UNIVERSITÀ DEGLI STUDI DI PADOVA

Facoltà di Ingegneria

Corso di Laurea di Ingegneria Informatica

Sencha Touch: framework per lo sviluppo di web application su piattaforma "mobile"

RELATORE
Prof. Carlo Fantozzi

TESI DI LAUREA DI
Giovanni Melai
Matr. N. 509736

Anno Accademico 2011/2012

A Elisa e ai miei genitori, che mi hanno supportato per lungo tempo.

Indice

Capitolo 1

Introduzione

Nell'ultima decade abbiamo assistito ad un costante aumento del numero dei dispositivi mobili, che hanno la possibilità di connettersi a Internet, sfruttando la rete cellulare o il sempre maggior numero di hot spot Wi-Fi.

I primi tentativi di portare una connessione dati sul cellulare sono stati fatti con il protocollo WAP[1]; che non ha avuto però un grande successo a causa dei costi e della lentezza nella navigazione, inoltre non dava la possibilità di visualizzare le normali pagine web ed era necessario pertanto creare siti ad hoc.

In anni più recenti siamo passati al protocollo HTML anche sui dispositivi mobili. Abbiamo qualche esempio nei dispositivi Nokia e BlackBerry, che davano la possibilità di navigare in internet e scaricare le email, seppur con qualche limitazione.

La vera e propria evoluzione dei dispositivi mobili si é avuta nel 2007 con l'ingresso di Apple nel mondo dei cellulari. Nel gennaio di quell'anno é stato presentato il primo iPhone, che ha suscitato un grandissimo interesse da parte dei media e degli utenti. Le caratteristiche rivoluzionarie di questo dispositivo sono state lo schermo multi touch che copriva la maggior parte del telefono, l'interfaccia rivoluzionaria e la possibilità di essere sempre connessi a internet visualizzando pagine complete senza problemi. Il successo di questo dispositivo ha dato il via alla creazione di oggetti simili da parte di molte altre aziende, che hanno utilizzato e fatto evolvere una serie di sistemi operativi concorrenti.

L'anno successivo, sempre Apple, dopo aver rilasciato l'SDK per lo sviluppo sulla piattaforma iPhone, ha lanciato un negozio di applicazioni per dare la possibilità ai programmatori di vendere le proprie creazioni in tutto il mondo senza limiti geografici, l'AppStore. Tantissimi sviluppatori hanno cominciato a creare app per iPhone, dando il via ad un mercato che prima non esisteva. Visto il grandissimo successo dello store, anche molte software house hanno cominciato a fare il porting di applicazioni desktop o creato applicazioni ad hoc. I competitor di Apple, come Google e RIM, non sono rimasti a guardare e hanno aperto i loro store online, creando nuove opportunità per i programmatori, attratti dai grandi numeri che fino a quel momento solo il mondo della Apple poteva offrire.

Con la nascita di nuovi sistemi operativi si é creata però la necessità di utilizzare diversi linguaggi di programmazione e conoscere il funzionamento di diversi SDK con una consistente spesa in termini di tempo.

Questo può non essere un problema per un'azienda di grandi dimensioni, ma lo é sicuramente per il singolo sviluppatore che vuole portare avanti il suo progetto e vuole distribuirlo sull'intero mercato.

[1] Wireless Application Protocol.

Per venire incontro a queste necessità sono nati una serie di framework che danno la possibilità di creare web application[2] che possono girare su tutti i dispositivi di ultima generazione, grazie all'integrazione in questi ultimi di browser sempre aggiornati e pronti per le nuove tecnologie, ne sono alcuni esempi: Sencha Touch, Sproutcore, Titanium mobile e JQuery mobile. Alcuni di questi mettono a disposizione, oltre a classi per la gestione completa dell'applicazione, anche dei layout pronti per essere usati. In questa tesina verrà approfondito Sencha Touch, uno dei framework più completi che il mercato offra attualmente.

Nel primo capitolo di questa tesina sarà effettuata una panoramica del framework Sencha Touch e una digressione sui linguaggi usati (HTML5, CSS3 e JavaScript). A seguire una carrellata delle principali funzioni di Sencha Touch: gestione delle classi, accesso ai dati, pannelli e layout.

Nel secondo capitolo verrà creata una semplice applicazione. Vedremo quali sono i requisiti per cominciare a lavorare e quali linee guida seguire per sviluppare buone applicazioni.

Nel capitolo conclusivo parleremo di PhoneGap, un framework che ci permette di trasformare le web app in applicazioni native per i vari Store come AppStore di Apple o l'Android Marketplace.

[2]Una web application é sostanzialmente una applicazione, che invece di essere eseguita sul computer dell'utente é accessibile via web, di solito viene sviluppata in linguaggi suppotrari dai browser, per renderla il più possibile universale.

Capitolo 2

Sencha Touch: struttura generale

Il framework che andremo a vedere nel corso di questa tesina é sicuramente di grande interesse per gli sviluppatori che vogliono creare applicazioni che funzionino sulla maggior parte delle piattaforme attualmente in commercio. É possibile creare delle applicazioni di consultazione e interazioni con servizi esterni, in maniera abbastanza semplice, questo comprende una gran quantità di applicazioni che si trovano negli store delle varie piattaforme. Non é possibile creare giochi che sfruttino la grafica 3D, perché in quel caso l'interconnessione con l'hardware é fondamentale, e questa non può essere garantita da Sencha Touch.

Il punto di forza di questo sistema é la possibilità di scrivere l'applicazione in un solo linguaggio, rendendola disponibile su molte piattaforme e pensando a quante sono le piattaforme attualmente sul mercato non é cosa da poco.

Ad ogni modo é necessario valutare i pro e i contro dei vari sistemi, prima di decidere quale strada intraprendere per lo sviluppo di un'applicazione, non é detto che questo framework vada bene in ogni circostanza. Alcune funzioni non sono ancora allo stato dell'arte e alcune non sono presenti, quindi, se si ha la necessità di qualche funzione in particolare, bisogna controlloare a priori se viene svolta bene da Sencha Touch, altimenti é meglio considerare la possibilità di utilizzare un diverso sistema.

Andremo ora a vedere come é strutturato il framework e quali linguaggi sfrutta.

2.1 Struttura e linguaggi

2.1.1 Panoramica sul framework

Sencha Touch é la prima piattaforma espressamente sviluppata per creare app per dispositivi mobili con HTML5, sfruttando ampiamente le capacità di CSS3 e Javascript [3]. É molto potente, flessibile ed ottimizzato per i sistemi mobili e gli schermi multitouch [1]. La scelta di utilizzare HTML5 viene dalla possibilità di questo linguaggio di distribuire facilmente contenuti come audio e video, ma soprattutto poiché é possibile salvare dati offline. Inoltre con l'utilizzo di CSS3 é possibile modificare a proprio piacimento l'aspetto grafico dell'app senza la necessità di avere una quantità di immagini nelle librerie dei componenti: gli stili, i bordi, i gradienti, le ombre, le transizioni, i menu, i pulsanti e anche le animazioni sono tutti in CSS puro. Cosa molto interessante é che la visualizzazione di tutti i componenti é indipendente dalla risoluzione dello schermo, quindi l'aspetto delle app si potrà adattare a qualsiasi dispositivo. Inoltre le animazioni e le transizioni tra le pagine che compongono l'applicazione, sono anch'esse gestite in automatico da Sencha

Touch, consentendo di creare interfacce complesse che ricalcano molto fedelmente quelle delle applicazioni native.

Pensando ai device per cui verranno sviluppate le applicazioni, il metodo di input primario é sicuramente il "tocco": per questo motivo é fondamentale che vengano riconosciuti, senza errori, la maggior parte dei movimenti normalmente utilizzati per interagire con le app a cui l'utente é abituato. Quindi, oltre ai classici touchstart e touchend[1], riconosciuti dai browser integrati nei device, vengono aggiunti dal framework una serie di eventi personalizzati come: il tocco, il doppio tocco, il pinch e la rotazione, sono solo alcuni esempi. Questi permettono interazioni viste finora solo nelle app native.

Sencha Touch é un framework cross-platform rivolto a dispositivi touchsreen di ultima generazione. É compatibile con i sistemi operativi Apple iOS 3+, Android 2.1+ e Blackberry 6+. Insieme, questi Sistemi Operativi, coprono una grandissima fetta del mercato e del traffico mobile, motivi che lo rendono molto appetibile a chi ha necessità di creare delle app indirizzate al maggior numero possibile di utenti. Gli sviluppatori possono contare su temi grafici specifici per ognuno di questi OS. C'é inoltre l'intenzione di rendere il framework compatibile con altre piattaforme, grazie al fatto che sempre più browser utilizzano WebKit[2] come motore di rendering web.

Una delle cose da evidenziare, per quanto riguarda l'interconnessione del framework con servizi esterni, é sicuramente il pacchetto per la gestione dei dati. É infatti possibile inviare richieste a una grande quantità di fonti attraverso i protocolli Ajax, JSONP e YQL, instradando i dati ricevuti verso uno specifico componente di visualizzazione. Inoltre é possibile salvare tali dati per una successiva consultazione offline, grazie a localStorage.

La prima versione di Sencha Touch, rilasciata nell'ottobre 2010, ha riscosso molto successo tra gli sviluppatori per la facilità di utilizzo e per la possibilità di creare app multipiattaforma, condivisibili attraverso un normale web server. Questo successo ha convito i creatori di Sencha Touch a migliorare il framework, rendendolo più performante, aggiornando il sistema delle classi, aumentando e migliorando la documentazione e creando un tool per la creazione automatica e l'invio della app direttamente all'Apple Store. Queste migliorie si trovano nella versione 2.0 di Sencha Touch, rilasciata nell'ottobre 2011, che é ancora in Developer Preview: non sono quindi state completate tutte le nuove caratteristiche e funzionalità ed il sistema non é completamente stabile, ma questa nuova release merita particolare attenzione per le potenzialità di sviluppo futuro e verrà pertanto approfondita in questa tesina.

Tutte queste caratteristiche sono integrate in un framework veramente leggero: l'intera libreria pesa meno di 120 Kb ed é possibile diminuirne ancora le dimensioni disabilitando i componenti e gli stili che non vengono usati.

2.1.2 HTML5

HTML (HyperText Markup Language) é un linguaggio per la definizione di documenti ipertestuali, tipici del World Wide Web [5]. Per usufruire di contenuti in questo formato é necessario utilizzare un web browser (Internet Explorer, Firefox, Safari o Chrome, ne sono alcuni esempi). Tali programmi interpretano il codice all'interno di file con estensione .html, al fine di generare la pagina desiderata sullo schermo del computer.

[1] Sono le due interazioni più semplici che si possono avere con un touchscreen. Touchstart é l'evento che viene chiamato quando il dito tocca lo schermo, mentre touchend viene chiamato quando il dito viene tolto dallo schermo.

[2] WebKit é un motore open source per browser web, creato da Apple. Attualmente viene usato da molti browser sia in ambito desktop che in quello mobile.

La prima versione di HTML é stata progettata nel 1989 da Tim Barners-Lee [6], ricercatore al CERN di Ginevra, che era alla ricerca di uno strumento per condividere testi scientifici con colleghi in tutto il mondo. Tim Barners-Lee voleva uno strumento per scrivere ipertesti che fosse indipendente dal tipo di computer e dal sistema operativo usato. La prima versione dell'HTML era molto semplice e contemplava la sola possibilità di inserire testi e link che li collegassero tra loro.

Figura 2.1.1: Logo HTML5

La prima versione pubblica del linguaggio fu presentata nel 1995, grazie al lavoro del W3C[3], un consorzio di aziende del settore informatico che da allora si occupa di stabilire gli standard per il web[4].

Nel corso degli anni l'HTML ha subito una serie di aggiornamenti, per seguire l'evoluzione di Internet ed assecondare le richieste degli sviluppatori. Dalla versione HTML 2.0 del 1995, si é passati alla versione HTML 3.2 nel 1997 quando sono stati introdotti i comandi per la gestione delle tabelle, del flusso del testo intorno alle immagini e delle lettere ad apice e pedice, l'ultima versione ufficiale é HTML 4.01 presentata nel 1999, che include il supporto per i fogli di stile, l'internazionalizzazione, l'accessibilità delle pagine web per i disabili, i frame e un miglioramento della gestione delle tabelle e dei form [8].

Il diretto successore di HTML 4.01 é 'l'XHTML[5]: esso associa alcune proprietà dell'XML[6] con le caratteritiche dell'HTML. Un file XHTML é una pagina HTML scritta in conformità con lo standard XML. Questo linguaggio prevede che i tag HTML vengano utilizzati per descrivere solamente la struttura logica della pagina, mentre layout e resa grafica sono completamente lasciati ai fogli di stile [10]. L'XHTML é il linguaggio che viene attualmente considerato come standard dal W3C.

L'evoluzione di tutti i linguaggi visti finora é l'HTML5, attualmente in fase di definizione (Draft) presso il W3C. La definizione delle specifiche iniziali é stata fatta da un gruppo esterno al W3C: il WHATWG[7]. Entrambi i team, W3C e WHATWG, sono al lavoro per la stesura delle specifiche definitive.

L'HTML5 é un insieme di tecnologie che include tutte le più recenti novità relative allo sviluppo web e alla creazione di web applications [11]. Una delle caratteristiche più importanti e richieste dai web developer, é la divisione totale dei contenuti dal layout,

[3]World Wide Web Consortium.

[4]La mission del W3C é quella di portare Il World Wide Web al suo massimo potenziale sviluppando protocolli e linee guida che assicurino una crescita a lungo termine dell'Web [7].

[5]eXtensible HyperText Markup Language.

[6]eXtensible Markup Language: é un metalinguaggio di markup che definisce un meccanismo sintattico che consente di estendere o controllare il significato di altri linguaggi marcatori.

[7]Web Hypertext Application Technology Working Group.

attraverso l'uso massiccio dei CSS. Vediamo di seguito le caratteristiche di questo nuovo linguaggio.

2.1.2.1 Semantica per le pagine e un markup più facile

Grazie ad una serie di studi ci si é accorti che i web designer utilizzavano alcuni nomi di classi per indicare sempre gli stessi tipi di sezione . Si é quindi pensato che sarebbe stato utile introdurre dei nuovi tag per identificare gli elementi maggiormente usati, in modo da dare anche un valore semantico alle varie parti di una pagina. Sono così nati nuovi elementi come: <header>, <nav>, <footer>, <aside>, <article>.

Inoltre, per semplificare la fruizione e la creazione di markup é stata semplificata la parte di validazione. Il doctype per indicare che il nostro documento é una pagina HTML5, una volta molto lungo, é semplicemente:

```
<!DOCTYPE html>
```

2.1.2.2 Offline e Storge

Con HTML5 si sta cercando di trasformare Internet da un semplice catalogo di pagine di testo a un gigantesco archivio di applicazioni web, uno strumento dai molteplici utilizzi, che vanno oltre la consultazione di documenti. Per giungere a questo scopo ci sono due tecnologie, presenti in questo nuovo linguaggio, che si rivelano particolarmente utili:

- AppCache: per dire al browser quali file (HTML, immagini, CSS, JavaScript) salvare in locale così da consentire la navigazione e l'utilizzo dell'applicazione anche quando si è offline;
- Local Storage: un'evoluzione dei cookies, molto più facile da gestire e soprattutto con 5 megabyte minimi di spazio a disposizione per salvare dati direttamente dal browser, senza passare da un database centralizzato e quindi raggiungibile solo quando connessi.

2.1.2.3 API di accesso ai device

HTML5 vuole essere la tecnologia di sviluppo del futuro, ed essendo ormai assodato che una buona parte degli accessi al web non sono più fatti solo dai PC, ma dai dispositivi mobili come smartphone e tablet, non si può prescindere da una certa integrazione con questi sitemi. Per sfruttare al massimo le funzionalità di tutti questi device HTML5 prevede l'utilizzo di una serie di API per accedere a caratteristiche e dati specifici dei vari terminali: attualmente è già implementata e funzionante sulla maggior parte dei browser, telefonini e tablet la geolocalizzazione, ma sono già in fase avanzata di progettazione altre API per utilizzare ad esempio la fotocamera, la lista dei contatti o i dispositivi audio. Queste funzioni saranno rese disponibili man mano che verranno definite tutte le specifiche di HTML5, ancora in fase di bozza, e quando i browser si adegueranno a tali specifiche.

2.1.2.4 Web Sockets

Uno dei problemi più ardui che ogni developer affronta nella sua carriera è far comunicare il browser con il server senza dover ogni volta ricaricare la pagina, aprendo cioè una connessione diretta e aggiornabile a prescindere dal resto della pagina.

Web Sockets risolve proprio questo problema: questa tecnologia (non ancora del tutto diffusa e implementata per problemi legati alla sicurezza ma comunque già abbastanza

matura e funzionante, ad esempio, su iPhone) consente di creare un canale di comunicazione full-duplex tra il browser e il server, dando così la possibilità di "dialogare" in maniera semplice e user-friendly per creare, ad esempio, applicazioni di chat in realtime o altre applicazioni che richiedono uno scambio costante di informazioni fra client e server (o fra vari client, passando attraverso il server).

2.1.2.5 Multimedia

Due tag che si rivelano sicuramente molto interessanti nel panorama moderno sono <audio> e <video>: essi danno la possibilità di inserire in maniera facile e veloce file audio e video, ma cosa più importante non richiedono plugin esterni, come Flash o Silverlight. Essendo questi tag parte del codice vi é la possibilità di modificarli con facilità e in tempo reale attraverso JavaScript, cosa non semplicissima con i plugin esterni.

2.1.2.6 Grafica, disegno ed effetti 3D

Sono presenti due strumenti per disegnare: canvas, che crea bitmap, e SVG per il disegno vettoriale. Con il solo tag <canvas> e un po' di JavaScript é possibile creare delle animazioni impensabili fino a qualche tempo fa.

2.1.2.7 Web workers

JavaScript sta diventando un linguaggio sempre più complesso e potente, richiedendo in alcuni casi una quantità di risorse non indifferente. Purtroppo, questo vuol dire spesso rallentare tutto il browser, rendendo la navigazione nel resto della pagina o l'utilizzo di altre finestre quasi impossibile.

Per risolvere questo problema è stata inventata la tecnologia Web Workers, ovvero un sistema per "slegare" un'attività intensiva dal resto dell'interfaccia utente: in sostanza, si decide che una funzione deve essere eseguita senza che questa blocchi l'utilizzo del sito e dunque, mentre si attendono i risultati è possibile comunque utilizare le altre funzionalità della web app e navigare sulla pagina come se nulla stesse impegnando in maniera intensiva il processore e il browser.

2.1.3 CSS3

Il CSS[8] é un linguaggio che viene usato per formattare documenti creati in HTML, XHTML e XML. É nato e cresciuto, quasi di pari passo con l'HTML, infatti le prime specifiche CSS sono state rilasciate un anno dopo quelle della prima versione dell'HTML. Le regole per comporre i CSS sono contenute in una serie di direttive emanate a partire dal 1996 dal W3C. L'idea che sta alla base di questo linguaggio é molto semplice: separare il contenuto dalla presentazione.

Con i CSS é possibile modificare il colore del testo, i font, dimensionare a piacimento l'interlinea, spaziare lettere e parole, inserire decorazioni, impostare stili diversi per titoli e paragrafi. Non solo, é anche possibile posizionare i vari elementi di una pagina e gestire i margini e i bordi di tutti gli elementi. Si possono impostare colori di sfondo e inserire immagini, tutto questo modificando un unico file, che contiene le regole per l'intero sito.

Nel 1998 si era già arrivati alla versione CSS2, che non ha portato grossi stravolgimenti, ma molte aggiunte rispetto alla prima. Dopo questa non ci sono state altre versioni

[8] Cascading Style Sheets.

raccomandate dal W3C, anche se la maggior parte dei browser supporta anche le specifiche CSS2.1; questo fa anche capire come i produttori di browser non attendono che un linguaggio sia del tutto definito per cominciare ad implementarlo.

Negli ultimi tempi si é cominciato a parlare, insieme ad HTML5, di CSS3, evoluzione delle precedenti versioni. Le specifiche di questo linguaggio hanno tre caratteristiche fondamentali.

- Lasciano intatto il core del linguaggio: tutto quello che é valido e funzionante in CSS2 continua ad esserlo in CSS3.
- Vengono aggiunte moltissime proprietà: il loro numero passa dalle 120 del CSS2 alle 245 del CSS3, con la possibilità di un aumento tramite l'introduzione di nuovi moduli.
- Organizzazione in moduli: invece di proporre una specifica unica, con la conseguente difficoltà di tenerla aggiornata, il W3C ha organizzato la specifica in moduli, ciascuno dei quali copre una determinata area dei CSS. Ognuno di questi moduli ha vita propria e può essere definito e aggiornato indipendentemente dagli altri: in questo modo é più facile poter seguire l'evoluzione del web.

Allo stato attuale é possibile cominciare a lavorare con le funzionalità più mature e supportate dalla maggior parte dei browser. Alcune delle specifiche già implementate dai browser sono le seguenti:

- **multiple background** dà la possibilità di sfruttare più immagini per creare un determinato sfondo:
- **text-shadow** serve per apportare l'ombreggiatura al testo;
- **@font-face** permette di aggiungere un font specifico, anche se non é presente sul computer;
- **border-radius** grazie a questa proprietà é possibile creare in maniera sempplice ed intuitiva angoli arrotondati, cosa che in precedenza si doveva fare ricorrendo ad una serie di escamotage;
- **opacity** conferisce una determinata trasparenza ad un elemento.

É inoltre possibile servire fogli di stile ad hoc in base alle caratteristiche dei dispositivi, ed anche sfruttare metodi e tecniche per dare dinamicità alle pagine, come transizioni, trasformazioni e animazioni.

2.1.4 JavaScript

JavaScript é un linguaggio di programmazione orientato agli oggetti, che serve ad aggiungere interattività alle pagine web [15]. Spesso viene chiamato linguaggio di scripting, con la velata intenzione di asserire che é più facile creare uno script che un programma, ma in questo caso é difficile fare questa distinzione.

Nonostante il nome, JavaScript e Java hanno poco in comune. La somiglianza dei nomi é dettata, nemmeno a dirlo, dalla vittoria del marketing sulla sostanza. Molti anni fa, quando Netscape ha aggiunto alcune abilità base di scripting nel suo web browser Navigator, ha chiamato quel linguaggio LiveScript. Nello stesso periodo stava prendendo piede Java. Con l'uscita della seconda versione di Navigator, che faceva girare anche le

applet Java, Netscape ha rinominato LiveScript in JavaScript, sperando di sfruttare la luce riflessa che Java stava producendo in quel periodo. Il fatto che JavaScript e Java fossero linguaggi molto differenti non ha fermato i responsabili del marketing di Netscape.

Quando Microsoft ha visto che che JavaScript stava diventando popolare tra i web developer, ha capito che avrebbe dovuto aggiungere qualche forma di scripting in Internet Explorer. Avrebbe potuto adottare JavaScript, ma come spesso accade, Microsoft ha pensato di creare un suo linguaggio, che facesse le stesse cose di JavaScript, ma non fosse proprio uguale. Lo chiamò JScript. Naturalmente, nel tempo, questo ha provocato qualche problema ai web developer, che hanno sempre dovuto controllare la compatibilità dei loro script con Internet Explorer, dovendoli modificare perché le proprie pagine web fossero visualizzate bene in ogni situazione.

Ci sono molte cose che si possono fare con Javascript per rendere interattive le pagine web e fornire agli utenti un'esperienza migliore e più emozionante. É possibile creare interfacce utente interattive, che diano un feedback all'utente mentre esplora le pagine. Per esempio con il *rollover* é possibile evidenziare un bottone quando ci si passa sopra col mouse. É possibile controllare che gli utenti inseriscano informazioni valide nei form, e fare dei conti all'interno di essi, usando la potenza di calcolo del client senza sovraccaricare il server. Con Javascript é possibile creare pagine web personalizzate in base alle scelte dell'utente. Più in generale, é possibile impostare i cookie, creare pagine HTML al volo e applicazioni web-based.

La caratteristica principale di JavaScript é quella di essere un linguaggio interpretato: il codice non viene compilato, ma interpretato lato client. La sintassi é relativamente simile a quella del C, del C++ e di Java. Poiché il codice viene eseguito direttamente sul client e non sul server ha il vantaggio che anche con la presenza di script particolarmente complessi, non si ha un sovraccarico del server a causa delle richieste del client. Lo svantaggio é che nel caso di uno script particolarmente pesante, il tempo per lo scaricamento può essere abbastanza lungo, anche se con le connessioni attuali il problema é meno sentito.

2.2 Il Framework Sencha Touch

Sencha Touch é un framework con molte funzioni già pronte per essere usate. Per provare a capire il suo funzionamento possiamo guardare la figura 2.2.1. La base del sistema é 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 é stato creato Sencha Touch.

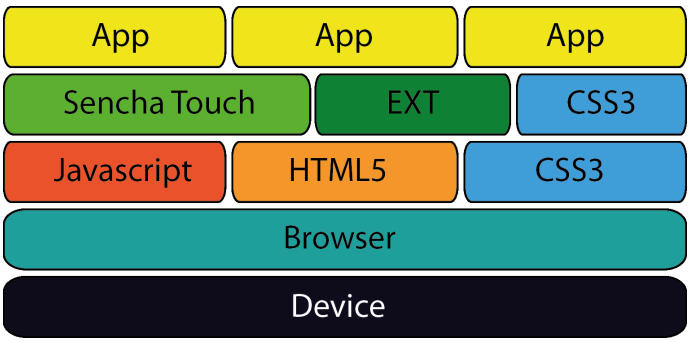

Figura 2.2.1: Funzionamento Sencha Touch.

Per creare le applicazioni andremo a lavorare su delle classi che ci vengono rese disponibili dal framework, useremo poco le normali funzioni di Javascript, perché esse sono inglobate nel framework. Ad esempio per lavorare sugli Array, Sencha Touch mette a disposizione una collezioni di metodi statici già pronti all'uso. Stessa cosa per gli oggetti, i numeri, le stringhe e così via. Oltre all'uso di Javascript, con qualche regola CSS3 é possibile migliorare la visualizzazione delle applicazioni.

Prima di andare a vedere nello specifico il funzionamento delle classi e delle funzioni più importanti, facciamo una piccola carrellata su quello che viene messo a disposizioni da Sencha Touch. Sono moltissime le cose già pronte per essere usate:

- **bottoni** - di diverse forme, colori e dimensioni;
- **form** - per l'inserimento dei dati da parte dell'utente, su di essi é anche possibile effettuare dei controlli senza sforzo di programmazione;
- **liste e liste nidificate** - per visualizzarle basta creare l'oggetto giusto e sono già come se le aspetterebbe l'utente;
- **icone** - una vasta selezione di icone già pronte all'uso;
- **toolbar** - già formattate e agganciabili ai vari lati dello schermo;
- **mappe** - é possibile sfruttare le mappe di Google senza fatica;
- **pop-up** - alert, action sheet e prompt pronti all'uso;
- **animazioni** - per passare da una view ad un'altra con varie transizioni;
- **touch** - molti tipi di gesture riconosciute automaticamente;
- **dati** - trasferire dati da server che comunicano attraverso JSON P, YQL e AJAX;
- **media** - caricare file audio e file video;
- **temi** - una serie di temi di visualizzazione pronti all'uso che copiano quelli dei vari sistemi operativi mobili;

Dal punto di vista pratico l'unico linguaggio, tra quelli visti, che andremo sicuramente ad utilizzare per la creazione di un applicazione in Sencha Touch é JavaScript, perché il codice HTML viene generato automaticamente e sono già presenti dei template in CSS3 che copiano le interfacce standard di iOS, Android e BlackBerry. Perciò, a meno di non voler creare un'interfaccia singolare, non é necessario avere una conoscenza particolareggiata di CSS3.

Come detto in precedenza Sencha Touch deriva da una versione desktop del framework, che si chiama Ext JS[9], e da questa prende la struttura delle classi. Infatti i nomi delle classi di entrambi i framework cominciano con Ext.

Andiamo ora a vedere in maniera più approfondita come funziona la versione 2.0 del framework.

[9]Ext JS é arrivato alla versione 4 e ha grosse somiglianze con Sencha Touch e naturalmente molte funzioni che non sono necessarie in un interfaccia per schermi touch.

2.2.1 Classi

Sencha Touch 2 usa lo stato dell'arte del sistema di classi sviluppato per Ext JS 4, la versione desktop da cui é nato Sencha Touch. Questo rende semplice la creazione di nuove classi in Javascript, fornendo ereditarietà, il caricamento automatico delle dipendenze, potenti opzioni di configurazione ed una serie di altre facilitazione che vedremo in seguito.

Una classe é un oggetto con alcune funzioni e proprietà legate ad essa. Di seguito l'esempio di una classe che rappresenta un animale, ne salva il nome e ha una funzione che lo fa parlare:

```
Ext.define('Animal', {
    config: { name: null },

    constructor: function(config) {
        this.initConfig(config);
    },

  speak: function() {
      alert('grunt');
    }
});
```

In *config* troviamo le variabili che vengono inizializzate di default alla creazione dell'oggetto. A seguire il costruttore e poi la funzione speak. Abbiamo così definito una classe che si chiama Animal, dove ogni animale ha un nome e può parlare. Vediamo ora come creare un'istanza di questa classe:

```
var bob = Ext.create('Animal', {name: 'Bob'});

bob.speak(); //alerts 'grunt'
```

Ecco creato un 'Animal' di nome Bob che visualizza un alert alla chiamata di *speak*. Ora che abbiamo una classe base, vediamo quali miglioramenti é possibile apportare. Potremmo cominciare dividendo gli animali in specie:

```
Ext.define('Human', {
  extend: 'Animal',

  speak: function() {
        alert(this.getName());
    }
});
```

Grazie al parametro *extend* abbiamo creato una nuova classe che eredita funzioni e configurazioni della classe Animal e la estende. Sovrascrivendo la funzione speak invece di un semplice 'grunt' la risposta sarà il nome impostato durante la creazione dell'oggetto.

Si può notare che é stata usata una funzione non definita in precedenza, getName(). Il sistema di classi crea in automatico delle opzioni di configurazione per le variabili e questi sono i metodi consigliati per gestire i dati nelle classi, in particolare crea:

- un getter, che ritorna il valore corrente, nel caso precedente getName();
- un setter, che modifica il valore della variabile e si invoca con setName();
- un applier, chiamato dal setter che da la possibilità di eseguire una funzione quando cambia una configurazione, nel nostro caso sarebbe applyName().

L'applier é una funzione che va definita insieme alla classe e viene chiamata automaticamente dal setter; se per esempio vogliamo chiedere la conferma della modifica del nome possiamo fare come segue:

```
Ext.define('Human', {
    extend: 'Animal',

    applyName: function(newName, oldName) {
        return confirm('Are you sure you want to change name to ' + newName
            + '?');
    }
});
```

In questo modo andiamo ad usare la funzione di conferma integrata nei browser, che apre una finestra con due possibili risposte, "Si" e "No". L'applier si comporterà in base alla risposta, modificando o meno la voce.

Con questo esempio abbiamo visto le funzioni base delle classi. Già così saremmo in grado di creare classi utili per creare applicazioni, ma ci sono ancora alcune cose da considerare e altri aiuti che ci vengono dal framework.

Spesso le classi dipendono da altre classi. Può capitare che per definirne alcune sia necessario usarne altre; bisogna però assicurarsi che le classi chiamate siano state caricate. Per fare ciò si usa la la keyword requires. Di seguito un esempio:

```
Ext.define('Human', {
  extend:'Animal',

  requires: 'Ext.MessageBox',

  speak: function(){
    Ext.Msg.alert(this.getName(), "Speaks...");
  }
});
```

Creando una classe in questo modo Sencha Touch controlla se la classe Ext.MessageBox é già stata caricata, se non lo é carica immediatamente la classe mancante attraverso AJAX, in questo modo il trasferimento avviene in background senza esplicito caricamento da parte dell'utente. Spesso capita che una classe invocata da noi abbia altre dipendenze: anche quelle vengono caricate in automatico. Una volta che tutte le dipendenze sono state soddisfatte é possibile usare Ext.create per creare istanze di una determinata classe. Questa catena di richieste é ottimale in fase di sviluppo, così non dobbiamo caricare a mano tutte le classi necessarie al nostro progetto, ma non va altrettanto bene quando dobbiamo condividere la nosta app, perché caricare tanti file uno a uno può risultare lento con una connessione internet.

Sarebbe utile avere la possibilita di caricare un solo file JavaScript contenente tutte le classi necessarie all'applicazione. Questo si può fare grazie a JSBuilder, un tool integrato in Sencha Touch 2, che analizza l'applicazione e crea un singolo file con tutte le classi dell'applicazione e le sole classi del framework che vengono usate. In questo modo viene caricato tutto una sola volta, a vantaggio della velocità di esecuzione.

2.2.2 Dati

Il pacchetto *data* é responsabile del caricamento e del salvataggio dei dati e la sua struttura rispecchia perfettamente il pattern MVC [10]. La maggior parte delle cose che può fare il pacchetto *data* vengono realizzate grazie alle tre classi seguenti:

- Model: rappresenta un'entità che viene usata nell'applicazione. Qualche esempio possono essere utenti, contatti e indirizzi. Possiamo dire che é l'involucro per i dati.

[10] Model view controller

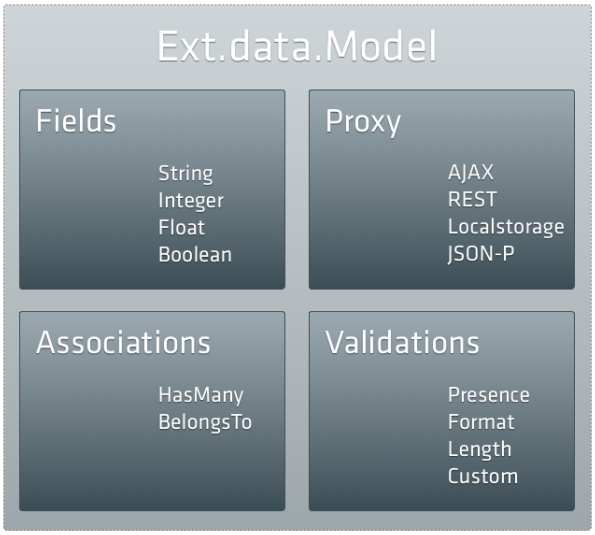

Figura 2.2.2: Struttura della classe Model

- Store: é una collezione di istanze di un modello (view). In generale é un array con alcune funzioni aggiuntive come l'ordinamento, la possibilità di filtrare le entry, il raggruppamento e la possibilità di lanciare eventi.
- Proxy: é responsabile del caricamento e del salvataggio dei dati (controller). Tipicamente viene creato un proxy AJAX che prende i dati da un server e li inserisce nello store.

Vediamo ora come si utilizza questo pacchetto. La prima cosa da fare é definire una classe derivata da *model*, che prepara la struttura dei dati che poi andremo a inserire. Un modello viene definito con un certo numero di *fields* (campi), che possono essere di tipo: *string*, *integer*, *float* o *boolean*, ma é comunque possibile definire altri tipi di dati grazie alla classe *Ext.data.Type*. Analizziamo di seguito come creare un modello per definire degli utenti:

```
Ext.define('User', {
    extend: 'Ext.data.Model',
    fields: [
        { name: 'id', type: 'int' },
        { name: 'name', type: 'string' }
    ]
});
```

Questa non é una classe, ma la definizione della struttura che useremo per immagazzinare i dati. I *fields* non vengono instanziati direttamente, ma quando viene chiamata una classe che estende *Ext.data.Model* vengono create delle istanze per ogni campo configurato nel modello. Il *model* é lo scheletro di quello che sarà poi il contenitore dei dati.

Una volta definito il modello bisogna creare il vero contenitore dei dati, questo é lo *store*. Lo *store* é fondamentalmente una collezione di istanze del modello. Insieme ad esso va configurato anche il *proxy*, che come abbiamo accennato prima é l'oggetto che riempie lo store, prendendo i dati da varie fonti, più in generale gestisce il trasferimento dei dati dalle fonti allo *store* e viceversa. Proprio in base ai dati che deve processare e alla fonte da cui deve ricavarli, va configurato in un modo o in un'altro:

```
Ext.create('Ext.data.Store', {
```

```
    model: 'User',
    proxy: {
        type: 'ajax',
        url : 'users.json',
        reader: 'json'
    },
    autoLoad: true
});
```

In questo esempio abbiamo creato uno *store* usanto il modello *User* definito prima e lo abbiamo impostato per interagire con un proxy AJAX, che attraverso l'URL user.json preleva i dati in formato JSON e li decodifica. É anche possibile definire il proxy insieme al modello, in modo che in tutti gli *store* creati usando quel *model* non sia necessario definire il *proxy* ogni volta.

É anche possibile popolare lo *store* con dati inline, quindi inserendoli direttamente nella fase di definizione, senza definire un *proxy*. Lo *store* converte gli oggetti che passiamo come dati in istanze del modello. Un piccolo esempio é il seguente:

```
Ext.create('Ext.data.Store', {
    model: 'User',
     data: [
        { id: '1', name: 'Ed' },
    { id: '2', name: 'Tommy' },
    { id: '3', name: 'Jamie' },
    ]
});
```

Un'altra caratteristica interessante dei *model* é che più modelli possono essere associati tra loro con le Associations API. In molte applicazioni capita che modelli diversi debbano interagire uno con l'altro e in generale i vari modelli di un'applicazione sono sempre legati tra loro (un po' come succede con le tabelle di un database). Per fare ciò si usano due metodi: *belongsTo* ed *hasMany*. Pensando alla struttura di un blog con utenti, articoli e commenti si può capire come usare queste relazioni. Ogni utente *hasMany* articoli e commenti. Ogni articolo *belongsTo* un utente e *hasMany* articoli. Infine ogni commento *belongsTo* utente e articolo.

L'ultima caratteristica che andiamo a vedere relativa ai modelli, é la validazione dei dati. Dopo aver definito i campi é possibile impostare alcuni controlli su di essi, vediamo di seguito quali sono le scelte:

1. **presenza** controlla che il campo abbia un valore;
2. **lunghezza** é possibile impostare un minimo e un massimo;
3. **formato** assicura che una stringa corrisponda ad un determinato formato di espressione (esempio classico potrebbe essere una data GG/MM/AAA);
4. **inclusione** per determinare un insieme di valori tra cui scegliere;
5. **esclusione** per determinare un insieme di valori che non é possibilie inserire, una sorta di blacklist.

Una volta determinati in un modello i valori da controllare, basta usare il metodo validate() su un oggetto che estenda quel modello. Il metodo validate() restituice un oggetto da cui é possibile sapere se ci sono errori e dove si trovano.

2.2.3 Componenti

Molte delle classi di visualizzazione con cui si interagisce in Sencha Touch sono componenti, che sono sottoclassi di *Ext.Component* e hanno tutti una serie di proprietà ereditate dalla superclasse, quindi possono:

- visualizzarsi in una pagina usando un template;
- rendersi visibili o nascosti in ogni momento;
- posizionarsi al centro dello schermo;
- abilitarsi e disabilitarsi, non é detto che un componente attivo sia anche visibile;
- comparire davanti ad altri componenti;
- cambiare dimensione/posizione sullo schermo con delle animazioni;
- agganciare altri componenti all'interno di essi;
- allinearsi con altri componenti.

Oltre ai componenti troviamo in Sencha Touch anche i contenitori. Questi hanno le stesse caratteristiche dei componenti e in aggiunta possono gestire e organizzare dei componenti figli. La maggior parte delle applicazioni hanno un unico *Container*, chiamato viewport, che occupa tutto lo schermo ed ha al suo interno i componenti che vanno a formare l'applicazione. É possibile specificare un layout nel *Container* per determinare come devono essere visualizzati i componenti sullo schermo.

Tutti i componenti possono lanciare eventi che é possibile intercettare per poter intraprendere delle azioni. Per fare ciò bisogna istanziare un listener alla creazione del componente. Per fare un esempio pratico consideriamo un campo di testo per il quale interessa sapere se viene modificato:

```
Ext.create('Ext.form.Text', {
    label: 'Name',
    listeners: {
        change: function(field, newValue, oldValue) {
            myStore.filter('name', newValue);
        }
    }
});
```

Ogni volta che il valore del campo cambia viene lanciato l'evento 'change' e viene chiamata la funzione che abbiamo inserito. In questo caso filtriamo uno *Store* in base al nome inserito nel campo, ma é possibile inserie qualsiasi altra funzione.

2.2.4 Pannelli

Andiamo ora ad esaminare la classe che ci permette di creare l'interfaccia della nostra applicazione, *Ext.panel*. Con questa classe possiamo istanziare i blocchi costruttivi di base, i *Panels*. Essi sono *containers* che possono visualizzare codice HTML e contenere altri elementi. Andiamo a vedere come viene creato un pannello:

```
var panel = Ext.create('Ext.Panel', {
    layout: 'hbox',

    items: [
        {
```

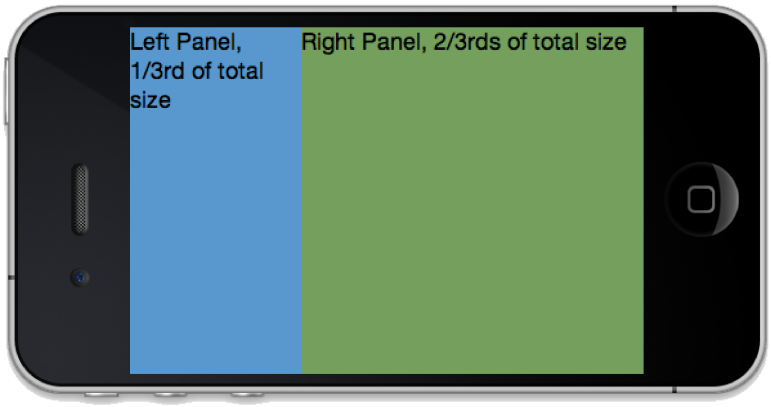

Figura 2.2.3: Immagine esempio hbox

```
            xtype: 'panel',
            flex: 1,
            html: 'Left Panel, 1/3rd of total size',
            style: 'background-color: #5E99CC;'
        },
        {
            xtype: 'panel',
            flex: 2,
            html: 'Right Panel, 2/3rds of total size',
            style: 'background-color: #759E60;'
        }
    ]
});

Ext.Viewport.add(panel);
```

Nell'esempio vengono creati 3 pannelli, il primo con la normale chiamata alla classe *Ext.create* e i due interni dichiarati inline usando un *xtype* [11]: in questo modo il framework crea in automatico i componenti. Per il primo pannello abbiamo specificato anche un layout, in questo caso 'hbox', che separa orizzontalmente il pannello padre con le proporzioni che vengono date nel parametro 'flex' dei figli, nel nostro caso risulterà 1/3 dello spazio per il primo pannello e 2/3 per il secondo. Possiamo vedere il risultato in figura 2.2.3.

Andiamo ora a vedere alcune utili funzioni che ci permettono di interagire con i componenti. In particolare useremo come esempio i pannelli.

Ogni componente ha una serie di opzioni di configurazione, che possono essere impostate alla creazione. Nella documentazione della classe si trovano tutte le opzioni disponibili. É anche possibile modificare una qualsiasi delle opzioni in un secondo momento, quando l'oggetto é già stato istanziato: in questo caso vengono in aiuto i getter e i setter, creati in automatico. Vediamo un piccolo esempio:

```
var panel = Ext.create('Ext.Panel', {
    fullscreen: true,
    html: 'This is a Panel'
});
panel.setHtml('Some new HTML');
```

[11] *Xtype* é un modo conveniente di creare un componente senza dover chiamare il metodo *Ext.create* specificando il nome intero della classe, come viene fatto per il primo pannello.

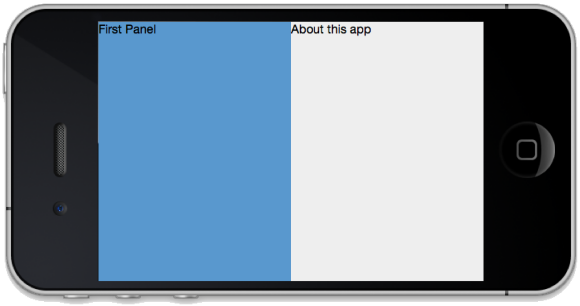

Figura 2.2.4: mainPanel e aboutPanel

In questo caso abbiamo modificato l'opzione html del pannello, grazie al metodo setHtml(). Passando al caso generale il framework crea le opzioni di configurazione a partire dal nome dell'opzione, quindi facendo un esempio per l'opzione defaultType verranno creati getDefaultType e setDefaultType e gli altri allo stesso modo.

Finora abbiamo creato tutti i pannelli in maniera statica, ma é anche possibile istanziare pannelli aggiuntivi durante l'esecuzione dell'applicazione. Vediamo un altro esempio:

```
var aboutPanel = Ext.create('Ext.Panel', {
    html: 'About this app'
});


var mainPanel = Ext.create('Ext.Panel', {
    fullscreen: true,
    layout: 'hbox',
    defaults: {
        flex: 1
    },
    items: {
        html: 'First Panel',
        style: 'background-color: #5E99CC;'
    }
});

mainPanel.add(aboutPanel);
```

Per primo viene creato il pannello aboutPanel che verrà visualizzato alla fine. Il secondo pannello che viene creato é il mainPanel, il pannello principale e viene visualizzato per primo, ha alcune opzioni di configurazione di cui abbiamo già parlato. Un'opzione che invece non abbiamo ancora visto é *defaults*: essa imposta le opzioni al suo interno come default per tutti i pannelli che verrano inseriti in mainPanel. Insieme al mainPanel viene definito un altro pannello che visualizza del semplice codice html. Arrivati a questo punto, quello che viene visualizzato sullo schermo é un pannello a tutto schermo con scritto 'First Panel'.

Nell'ultima riga viene chiamato il metodo add dell'oggetto mainPanel: in questo modo viene inserito al suo interno aboutPanel. Adesso quello che viene rappresentato sullo schermo sono due pannelli: il primo ha sempre la scritta 'First Panel', mentre il secondo ha la scritta 'About this app'; occupano esattamente metà schermo a testa, perché per default hanno entrambi l'opzione *flex* impostata a 1. Vediamo il risultati in figura 2.2.4.

Quando non si ha più la necessita di visualizzare il pannello aboutPanel é possibile rimuoverlo invocando il metodo *remove*.

2.2.5 Layout

Nei dispositivi touch siamo ormai abituati ad avere un determinato tipo di interfaccia. Certo non ci aspettiamo di poter vedere contemporaneamente molte finestre, come potremmo avere su un computer; ci attendiamo piuttosto di avere in primo piano solo la schermata che ci interessa per una determinata operazione, questo é quello a cui siamo abituati. Per questo motivo Sencha Touch mette a disposizione dei layout che rispecchiano quello che ci aspettiamo da una applicazione mobile. Ai fini pratici un layout é una proprietà che viene attribuita ad un panel per dirgli come deve essere visualizzato.

Le direttive relative al layout vengono usate per posizionare e dimensionare i *Components* dell'applicazione. Vediamo di seguito come sfruttarle.

2.2.5.1 Fit

Figura 2.2.5: Fit

Partiamo da quello più semplice. *Fit* fa sì che il componente figlio copra tutta la dimensione del contenitore padre. Vediamo un esempio in cui il pannello container ha le dimensioni 200px per 200px e viene dato il layout *fit* al suo unico figlio. Anche il figlio avrà dimensioni 200px per 200px.

```
var panel = Ext.create('Ext.Panel', {
    width: 200,
    height: 200,
    layout: 'fit',

    items: {
        xtype: 'panel',
        html: 'Also 200px by 200px'
    }
});
```

2.2.5.2 Card

Capita spesso di voler visualizzare più schermate di informazioni contemporaneamente, ma non é possibile a causa delle dimensioni degli schermi. Grazie al layout *cards* é possibile visualizzare, una alla volta, una serie di schermate. Per spostarsi tra le schermate si possono usare i TabPanel, barre orizzontali con tab multipli da cliccare, o i Carousels, pagine da far scorrere. Come nel caso precedente viene coperta tutta la dimensione del container, solo che vengono istanziate più schermate pronte ad essere visualizzate.

```
var panel = Ext.create('Ext.Panel', {
    layout: 'card',
    items: [
        {
            html: "First Item"
```

```
        },
        {
            html: "Second Item"
        },
        {
            html: "Third Item"
        },
    ]
});

panel.getLayout().setActiveItem(1);
```

In questo caso viene istanziato un pannello con tre schermate e usando il metodo *setActiveItem* si sceglie quale visualizzare, nell'esempio viene visualizzato il secondo (l'indice base é zero, quindi 1 é il secondo). Questo metodo viene chiamato dal TabPanel (figura 2.2.6) o dal Carousel (figura 2.2.7).

Figura 2.2.6: TabPanel

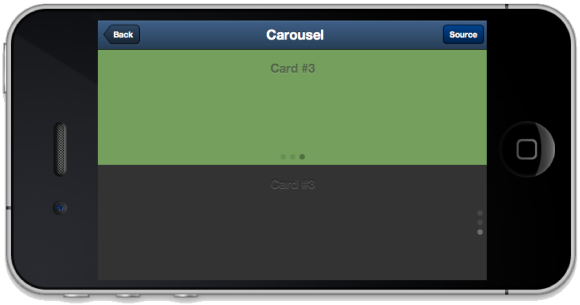

Figura 2.2.7: Carousel

2.2.5.3 HBox e VBox

Nel caso di schermi grandi, come potrebbero essere quelli dei tablet, la possibilità di dividere lo schermo in più sezioni offre un'ampia gamma di applicazioni. Questo non vuol dire che non si possa fare la stessa cosa su uno smartphone, ma sicuramente non avrebbe lo stesso effetto. Un esempio di divisione è quello che troviamo nei programmi di posta elettronica, dove vediamo da una parte l'elenco dei messaggi e dall'altra l'anteprima di messaggio.

In Sencha Touch questo effetto si ottiene con i layout *hbox* e *vbox* rispettivamente per la divisione in orizzontale e in verticale.

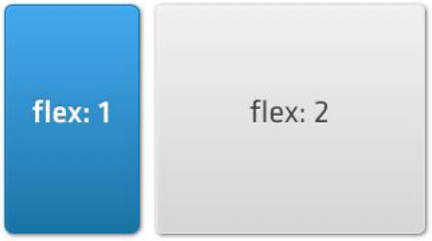

Figura 2.2.8: HBox

Figura 2.2.9: Dock top

```
Ext.create('Ext.Container', {
    fullscreen: true,
    layout: 'hbox',
    items: [
        {
            xtype: 'panel',
            html: 'message list',
            flex: 1
        },
        {
            xtype: 'panel',
            html: 'message preview',
            flex: 2
        }
    ]
});
```

Nell'esempio riportato sopra vediamo il funzionamento di *hbox*. Creiamo un container e al suo interno due pannelli. Come visto in precedenza la divisione dello schermo viene data dal parametro *flex* dei pannelli, il primo occuperà 1/3 dello schermo e il secondo 2/3 (vediamo un esempio grafico in figura 2.2.8). Allo stesso modo é possibile creare un container diviso in verticale, sostituendo il parametro *hbox* con *vbox*.

2.2.5.4 Docking

All'interno di uno qualsiasi dei layout visti fino a questo momento é possibile inserire un *dock*, un pannello che viene agganciato ad uno dei lati del contenitore, ridimensionando se necessario gli altri. Le possibili scelte per il parametro dock sono: top, right, bottom e left. Per fare un esempio basta modificare il layout precedente.

```
Ext.create('Ext.Container', {
    fullscreen: true,
    layout: 'hbox',
    items: [
        {
```

```
            dock: 'top',
            xtype: 'panel',
            height: 20,
            html: 'This is docked to the top'
        },
        {
            xtype: 'panel',
            html: 'message list',
            flex: 1
        },
        {
            xtype: 'panel',
            html: 'message preview',
            flex: 2
        }
    ]
});
```

É stato inserito un pannello con il parametro *dock* impostato a top: viene così agganciato alla parte alta dello schermo e costringe gli altri panneli ad adattarsi alla nuova disposizione (vediamo un esempio in figura 2.2.9).

2.2.5.5 Eventi

Essendo Sencha Touch un framework specificamente creato per i dispositivi mobili moderni, che hanno come metodo di input lo schermo, esso deve riconoscere il maggior numero possibile di gestures. Questo avviene grazie ad una classe interna che normaliza il comportamento dei vari browser, evitando che si propaghino eventi inaspettati. In pratica blocca tutti gli eventi di default e risponde in un determinato modo, uguale per tutte le piattaforme.

La classe che riconosce le gestures é *Ext.event.recognizer.Recognizer* a cui non si ha accesso diretto, perché é una classe privata del framework, ma può essere usata attraveso un'altra classe *Ext.event.Event*, che ci permette di rilevare molti eventi, come: *DoubleTap*, *Drag*, *Swipe*, *Rotate* e altri a cui siamo abituati.

Sencha Touch sfrutta già questo sistema in automatico all'interno delle sue classi, ma é possibile inserire, nei pannelli creati, dei Recognizer personalizzati per modificare il comportamento di default. Questi sono i soli eventi riconosciuti dal framework, quindi per effettuare operazioni é necessario l'input dell'utente.

Capitolo 3

Sencha Touch: come realizzare un'applicazione

In questa ultima parte vedremo cosa ci serve per creare una piccola applicazione di block-notes seguendo il paradigma MVC, consigliato dagli sviluppatori Sencha così da rendere più facile gestire lavori di grosse dimensioni.

3.1 Requisiti

La prima cosa da fare é scegliere quale versione di Sencha Touch vogliamo scaricare. Le funzionalità delle diverse versioni sono le medesime, la scelta dipende dal tipo di uso che si vorrà fare dell'applicazione. La divisione viene fatta in base alla licenza d'uso.

- La licenza commerciale, diventata da poco gratuita, é indicata per chi vuole creare applicazioni proprietarie senza distribuire e condividere il codice sorgente. É possibile creare un numero illimitato di applicazioni, senza dover pagare in base al numero di utenti o di programmatori. Con questa versione é possibile comprare dei pacchetti di assistenza, per farsi aiutare nello sviluppo da programmatori esperti.
- La licenza commerciale OEM, a pagamento, da usare se si vuole creare una propria SDK, o in application builder basati sul framework.
- La licenza open source, naturalmente gratuita, va bene per chi vuole creare applicazioni open source sotto la licenza GNU GPL v3. La regola più importante da rispettare é che bisogna rilasciare l'intero codice sorgente dell'applicazione all'utente, in modo tale che possa modificarla in base alle sue necessità. Questa é la versione che userò per l'applicazione di seguito descritta.

Come già anticipato analizzeremo la versione 2.0 del framework, ancora in developer preview, quindi non ancora stabile, ma adatta a questa specifica esigenza.

All'interno della cartella che contiene il framework é presente tutta la documentazione, consultabile anche offline.

Di per sè il framework potrebbe funzionare anche senza un web-server, in quanto il codice é tutto JavaScript, ma per alcune funzioni di interazione con servizi esterni, é necessario che risieda in un server che gli permetta di comunicare attraverso AJAX. Il modo più semplice per avere un web-server sul proprio computer é scaricare un pacchetto già pronto con tutto il necessario, come WAMP o MAMP.

Ultimo requisito: per visualizzare il lavoro svolto é necessario un web browser moderno come Chrome, Safari; o quello degli smartphone e tablet con iOS o Android. Per gli

screenshoot presenti in questa tesina ho usato il simulatore iOS, presente nel pacchetto di sviluppo XCode per Mac.

3.2 NotePad

3.2.1 I file base

Il team di sviluppo di Sencha Touch consiglia di seguire una determinata struttura per le applicazioni; un piccolo insieme di convenzioni e classi che semplifica la creazione ed il mantenimento delle app, in particolar modo quando si lavora in gruppo.

La prima cosa da fare é creare la struttura della cartella che conterrà l'applicazione: all'interno di questa bisogna inserire una copia della cartella contenente il framework e due file.

- index.html - un semplice file html che carica il framework e l'applicazione.
- app.js - il file in cui viene definito il nome dell'applicazione, l'icona della home, e cosa deve fare l'applicazione all'avvio.
- touch - la cartella Sencha Touch così com'é scaricata dal sito. In realtà in vari tutorial viene consigliato di mettere un symbolic link alla cartella posizionata da un'altra parte, in modo tale che in caso di aggiornamenti sia possibile aggiornare il framework solo in un punto per più applicazioni e lo spazio occupato non sia elevato.

Cominciamo a vedere il file index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>NotePad</title>
    <link rel="stylesheet" href="touch/resources/css/apple.css" type="text/
        css">
  <link rel="stylesheet" href="app.css" type="text/css">

  <script type="text/javascript" src="touch/sencha-touch-debug.js"></script
      >
    <script type="text/javascript" src="app.js"></script>
</head>
<body></body>
</html>
```

Questa é una pagina HTML5 molto semplice (si può notare il Doctype semplificato rispetto alle precedenti versioni di HTML): per prima cosa carica i CSS che intendiamo usare e il file JavaScript di Sencha Touch, successivamente il file della nostra applicazione, app.js. Tutto ciò viene fatto nell'header della pagina; il body rimane vuoto, perché verrà popolato in seguito da Sencha Touch. Normalmente non sono necessarie ulteriori modifiche a questo file, a meno che non si debba aggiungere qualche template in CSS o qualche file JavaScript. In questo caso usiamo i CSS di Sencha che replicano l'interfaccia di iOS più un file app.css, in cui possiamo inserire i nostri CSS personalizzati. Per quanto riguarda il framework carichiamo il file sencha-touch-debug.js, che ci permette di caricare dinamicamente tutte le classi richieste dal programma e stampare sulla console gli errori, molto utile in fase di sviluppo. Come si può vedere dalla figura 3.2.1 sono presenti anche altre build del framework: quelle contrassegnate con "all" hanno al loro interno tutte le classi del framework, sono quindi pesanti, e hanno molte classi che non verranno usate in una

specifica applicazione. Vedremo più avanti come creare una custom build che racchiuda al suo interno solo le classi richieste.

Name	Type	Loader	Minified	Comments	Debug	Compat	Usage
sencha-touch-debug.js	Core	✔		✔	✔		Use when developing your app locally
sencha-touch.js	Core	✔	✔				Use in production with a custom build
sencha-touch-all.js	All		✔				Use in production if you don't have a custom build
sencha-touch-all-debug.js	All			✔	✔		Use to debug your app in staging/production
sencha-touch-all-compat.js	All			✔	✔	✔	Use to migrate your 1.x app to 2.x

Figura 3.2.1: Le builds del framework.

Nello sviluppo di Sencha Touch 2 gli sviluppatori hanno impiegato molte forze per migliorare l'organizzazione con il pattern Model View Controller, perciò faremo uno sforzo maggiore per capire come sfruttare questo sistema. La prima cosa da fare é creare una divisione dei file in cartelle: nella root dell'applicazione avremo, quindi, una cartella app con al suo interno le cartelle in cui posizionare rispettivamente i file con i model, le view, i controller e gli store.

Ecco come appaino i percorsi relativi:

- app/controller
- app/model
- app/store
- app/view

Usando poi una semplice convenzione é possibile richiamare i vari file che ci interessano, per esempio per selezionare la view app/view/Main.js useremo la stringa 'NotePad.view.Main' (dove NotePad é il nome dell'applicazione) e così per tutte le altre classi.

Passiamo ora al file app.js, dove risiede il codice vero e proprio per avviare l'applicazione.

```
Ext.Loader.setConfig({ enabled: true });

Ext.application({
    name: 'NotePad',

    launch: function() {

    }
});
```

Prima di tutto abilitiamo il loader che carica in automatico le dipendenze di cui avremo bisogno. Passiamo poi alla classe *Ext.application* che crea una variabile globale chiamata 'NotePad'; tutte le classi dell'applicazione saranno sotto questo unico namespace in modo tale di limitare la possibilità di collisione con altre variabili globali. Quando la pagina é pronta e tutto il codice Javascript é stato caricato, viene chiamata dal framework la funzione *launch* che é in esso definita, da quel momento é possibile eseguire il codice che fa funzionare l'applicazione.

3.2.2 La main view

La prima cosa da fare é creare una view che venga caricata all'avvio dell'applicazione. Andiamo quindi a creare il file app/view/Main.js:

```
Ext.define('NotePad.view.Main',{
      extend: 'Ext.Panel',
      id: 'main',

      config: {
        fullscreen: true,
        layout:{
          type: 'card'
        },
        html: 'Questa é la viewport che conterrà tutta l\'applicazione',
      }
});
```

In questo modo definiamo una view che estende la classe Ext.Panel, gli assegniamo un id per poterla identificare in seguito, qualche opzione per la configurazione: fullscreen indica che deve coprire tutto lo schermo e layout come devono essere organizzate le view che andremo a inserire, infine inseriamo del codice html da visualizzare. Al momento però la view é solo definita, ma non inizializzata, per fare ciò andiamo ad aggiungere la chiamata per la creazione della view nel file app.js che viene così modificato:

```
Ext.application({
    name: 'NotePad',

  models:[],
  views: ['Main'],
  controllers[],
  stores:[],

    launch: function() {

    Ext.create('NotePad.view.Main');

    }
});
```

Nell'intestazione di questo file vanno dichiarate tutte le classi che creeremo, che siano model, view, controller o store. In questo modo verranno caricati i file relativi all'avvio dell'applicazione, per cominciare inseriamo la view Main.

A questo punto andando con un browser compatibile nella cartella dove risiede l'applicazione otterremo il risultato in figura 3.2.2.

Figura 3.2.2: La Viewport.

3.2.3 Le view di lavoro

Per strutturare un'applicazione che funga da block notes servono fondamentalmente due view: la prima per visualizzare l'elenco delle note e la seconda per leggere e modificare le note.

Cominciamo con la prima, sarà formata da un panel con al suo interno una toolbar in alto ed una lista delle note. Sotto andiamo quindi a definire una nuova view, che chiameremo notesListContainer.js, vediamo il codice:

```
Ext.define('NotePad.view.NotesListContainer',{
      extend: 'Ext.Panel',

      config:{
        id: 'notesListContainer',
        layout: 'fit',

        items: [

            {
               xtype: 'toolbar',
              id: 'notesListToolbar',
              docked: 'top',
              title: 'Le mie Note',

            },

            {
               xtype: 'list',
              id: 'notesList',
```

```
            }
        ]
    }
});
```

In questo modo abbiamo inserito nella view, con gli xtype, la toolbar e la lista. Per visualizzare il risultato andiamo a modificare il file Main.js, sostituendo la riga "html: 'Questa é la viewport..." il seguente codice.

```
items:[
  Ext.create('NotePad.view.NotesListContainer'),
]
```

Adesso nella view Main viene creata e visualizzata la view NotesListContainer. Possiamo vedere il risultato finale in figura 3.2.3.

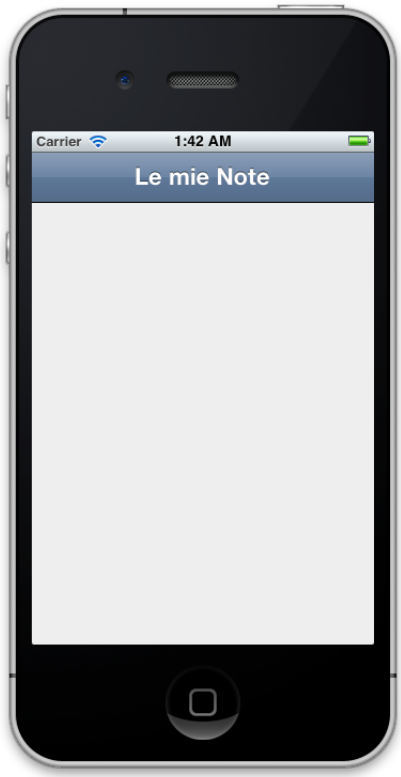

Figura 3.2.3: NotesListContainer.

3.2.4 Il model e lo store

Per visualizzare elementi nella lista é necessario definire il modello dei dati che verranno rappresentati, ciò va fatto in un file che chiameremo Note.js, da posizionare in app/model. Come per le view andremo ad utilizzare la classe Ext.define, vediamo il codice.

```
Ext.define('NotePad.model.Note',{
  extend: 'Ext.data.Model',

  config:{
    fields:[
      { name: 'id', type: 'int'},
```

```
      { name: 'date', type: 'date', dateFormat: 'c'},
      { name: 'title', type: 'string'},
      { name: 'text', type: 'string'}
    ],
    validations:[
      { type: 'presence', field: 'id'},
      { type: 'presence', field: 'title', message: 'Inserire un titolo per
          la nota'}
    ]
  }
});
```

Il modello viene definito estendendo la classe Ext.data.Model. Abbiamo inserito quattro campi: id, data, titolo e testo. Inoltre abbiamo aggiunto un controllo per essere sicuri che siano presenti id e titolo, se così non fosse verrà restituito un errore. Il model definisce solo come i dati devono essere salvati, ma per il salvataggio vero e proprio abbiamo bisogno di uno store. Come nei casi precedenti andiamo a definire lo store nella posizione giusta, app/store/NotesStore.js

```
Ext.define('NotePad.store.NotesStore',{
  extend: 'Ext.data.Store',

  config: {
    model: 'NotePad.model.Note',

    sorters:[{
      property: 'date',
      direction: 'DESC'
    }],
    proxy: {
      type: 'localstorage',
      id: 'notes-store'
    }
  }
});
```

Ecco definito uno store partendo dal modello creato in precedenza, con una proprietà per l'ordinamento in base alla data di creazione della nota e un proxy che salva i dati grazie al servizio localstorage del browser. A questo punto abbiamo uno store pronto per esserre popolato.

La prima cosa da fare é collegare la lista allo store e impostare cosa visualizzare. Ciò può essere fatto modificando l'xtype che genera la lista all'interno della view NotesList Container, che verrà modificata in questo modo:

```
var store = Ext.create('NotePad.store.NotesStore');
store.load();

Ext.define('NotePad.view.NotesListContainer',{
      extend: 'Ext.Panel',

      config:{
        id: 'notesListContainer',
        layout: 'fit',

        items: [
            {
               xtype: 'toolbar',
              id: 'notesListToolbar',
              docked: 'top',
              title: 'Le mie Note',
```

```
            },
            {
               xtype: 'list',
              id: 'notesList',
              store: store,
              itemTpl: '<div class="item-title">{title}</div>' +
                 '<div class="item-text">{text}</div>'

            }
        ]
      }
});
```

Per prima cosa creiamo un'istanza dello store con Ext.create e lo carichiamo subito in memoria, store.load(). Una volta caricato lo Store andiamo a impostarlo come risorsa nella lista con l'opzione di configurazione "store: store", infine bisogna dire alla lista come utilizzare i dati presenti nello store, atteaverso l'opzione "itemTpl: ...", in questo caso visualizzeremo il titolo e una parte della nota. Assegnando una classe ai che conterranno il titolo e il testo della nota é possibile modificarne la visualizzazione agendo sul file app.css.

Per controllare il lavoro fatto finora dobbiamo caricare dei dati nello store, ma al momento non ci é possibile agire dall'interfaccia, possiamo però sfruttare una funzione degli store che ci permette di caricare dati inline. Torniamo perciò al file in cui é definito lo store e aggiungiamo un campo all'interno del config.

```
...

  config: {
    ...
    // Da cancellare quando verrà attivato il salvataggio dei dati nello
        store
    data: [
          { id: 1, date: new Date(), title: 'Nota di prova', text: 'Ecco un
              po\' di testo per la nota' },
    ]
  }
```

A questo punto se visualizziamo la lista, dopo aver fatto qualche modifica al file app.css il risultato sarà quello in figura 3.2.4.

3.2.5 I bottoni

Andiamo ora a completare la prima view, manca il bottone per creare le note, che dovrà essere posizionato all'interno della toolbar in alto. Grazie agli xtype il lavoro é molto semplice basta aggiungere il codice seguente direttamente all'interno della toolbar.

```
...
{
  xtype: 'toolbar',
  ...
  items:[
    {xtype: 'spacer'},
    {   xtype: 'button',
      id: 'newNoteButton',
      action: 'createnewnote',
      iconCls: 'add',
      iconMask: true,
      ui: 'action',
```

Figura 3.2.4: Lista con nota di prova.

```
    }
  ]
...
```

L'xtype "spacer" serve per spostare il pulsante a destra. Il secondo xtype é invece quello che crea il bottone, a cui assegnamo un id e una delle icone messe a dispososizione dal framework, inoltre con "ui" diamo invece una forma particolare al bottone. Un campo importante da inserire é "action", servirà più avanti per interagire col pulsante. Il risultato in figura 3.2.5.

Passiamo ora a creare la seconda view, quella per la modifica e la visualizzazione delle note. Avendo già visto come si crea una view blocco a blocco, la second la vediamo nella sua forma finale.

```
Ext.define('NotePad.view.NoteEditor',{
      extend: 'Ext.form.Panel',

      config:{
        id: 'noteEditor',

        items: [
          // toolbar in alto con i pulsanti home e salva
          {
             xtype: 'toolbar',
            id: 'noteEditorTop',
            docked: 'top',
            title: 'Modifica la nota',
            items: [
              {
                xtype: 'button',
```

Figura 3.2.5: Bottone per creare le note.

```
            action: 'back',
            text: 'Indietro',
            ui: 'back',
          },
          { xtype: 'spacer'},
          {
             xtype: 'button',
            action: 'savenote',
            text: 'Salva',
            ui: 'action',
          }
        ]
      },
      {
         xtype: 'fieldset',
        items:[

          //campo titolo della nota
          {
            xtype: 'textfield',
            name: 'title',
            label:'Title',
            required: true
          },
          //campo testo della nota
          {
            xtype: 'textareafield',
            name: 'text',
            label:'Note',
          },
```

```
            ]
          },
          //toolbar in basso con il pulsante cestino
          {
             xtype: 'toolbar',
            id: 'noteEditorBottom',
            docked: 'bottom',
            items: [
              { xtype: 'spacer'},
              {
                iconCls: 'trash',
                iconMask: true,
                id: 'cancel',
                action: 'cancel'
              }
            ]
          }
        ]
      }
});
```

In questo modo abbiamo creato una view con al centro un form, coi campi titolo e nota, per l'inserimento dei dati e due toolbar, una in alto con i bottoni "Indietro" e "Salva" e una in basso con un bottone che ha l'icona del cestino. Andando a modificare la view Main possiamo caricare quella appena creata per vedere il risultato (figura 3.2.6).

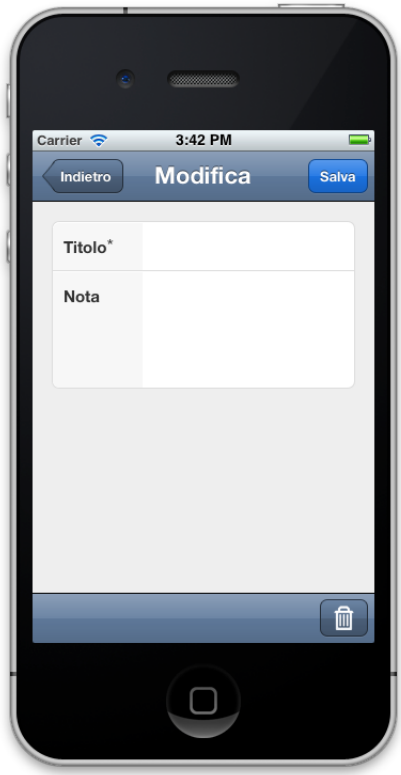

Figura 3.2.6: La view NoteEditor.

3.2.6 Il controller

Adesso che abbiamo le view pronte bisogna farle interagire tra loro e con l'utente. É arrivato il momento di inserire il controller, che gestirà tutte le azioni e gli eventi dell'applicazione. Per semplicità useremo un unico controller, ma in caso di applicazioni più complesse conviene dividere anche i controller in base alle necessità, per non ottenere file troppo complessi.

Cominciamo creando il file nella giusta posizione: app/controller/Main.js. La prima azione che ci interessa inserire é quella per creare una nuova nota, provocano il passaggio dalla view NotesListContainer a NoteEditor.

```
Ext.define('NotePad.controller.Main',{
  extend: 'Ext.app.Controller',

  config: {
    refs: {
      main: '#main',
      noteEditor: '#noteEditor',
    },
    control:{
      'button[action=createnewnote]': { tap: 'createNewNote'},
    }
  },

  createNewNote: function(){
    var now = new Date();
    var noteId = now.getTime();
    var note = Ext.create('NotePad.model.Note',
            { id: noteId, date: now, title: '', text: ''}
          );
    view2 = this.getNoteEditor();
    view2.setRecord(note);
    this.getMain().getLayout().setAnimation({type:'slide', direction:'left'
        , duration:200});
    this.getMain().setActiveItem(view2);
  },
});
```

Si comincia definendo il controller Main come estensione di Ext.app.Controller, poi passiamo alla configurazione, nel campo refs si inseriscono i riferimenti alle view che ci servono in base agli id, in modo da poterle richiamare facilmente. Grazie a questi é possibile avere i riferimenti diretti alle view già definite. Nel campo control vanno inserite le azioni che il controller deve agganciare, in questo caso prende carico del bottone con la action=createnewnote (impostata in fase di creazione dei bottoni) se viene cliccato e chiama il metodo createNewNote().

Vediamo cosa fa il metodo createNewNote(). Imposta i parametri di base per la nota, crea un Id univoco in base alla data e un modello di nota con titolo e testo vuoti, ma id e data compilati. A questo punto crea una variabile relativa alla view noteEditor ed inserisce in essa la nota base creata in precedenza. A questo punto imposta il tipo di animazione e la direzione da usare per il passaggio di view e infine, col metodo setActiveItem(), attiva la nuova view. Affinché tutto funzioni bisogna ricordarsi di inserire all'inizio del file app.js la chiamata per caricare il controller, a questo punto si presentrà come segue.

```
...
Ext.application({
  name: 'NotePad',

  models: ['Note'],
```

```
  stores: ['NotesStore'],
  controllers: ['Main'],
  views: ['Main', 'NoteEditor', 'NotesListContainer'],

  ...
});
```

Inseriamo ora nel controller la funzione per tornare alla view precedente col tasto Indietro. Il codice é molto semplice. In questo caso la funzione non é complicata, si deve solo impostare l'animazione da usare e chiamare il metodo getActiveItem() verso la prima schermata. Con qualche aggiunta al config otteniamo il seguente risultato.

```
Ext.define('NotePad.controller.Main',{
  extend: 'Ext.app.Controller',

  config: {
    refs: {
      main: '#main',
      noteEditor: '#noteEditor',
      notesList: '#notesListContainer',
    },
    control:{
      'button[action=createnewnote]': { tap: 'createNewNote'},
      'button[action=back]': { tap: 'backHome'},
    }
  },
  ...

  backHome: function(){
    this.getMain().getLayout().setAnimation({ type:'slide', direction: '
        right', duration: 200 });     this.getMain().setActiveItem(this.
        getNotesList());
  },


});
```

A questo punto abbiamo la possibilità di muoverci tra le due view senza problemi, ma non possiamo ancora salvare le nostre note.

Per prima cosa é necessario modificare il modulo di validazione all'interno del model Note per inserire il messaggio da visualizzare in caso di errore, così diventa più comoda la gestione degli errore.

```
Ext.define('NotePad.model.Note',{
  extend: 'Ext.data.Model',
  config:{
  ...
  validations:[
      { type: 'presence', field: 'id'},
      { type: 'presence', field: 'title', message: 'Inserire un titolo per
          la nota'}
    ]
  }
});
```

Andiamo ora a vedere come salvare effettivamente le note nello store. Torniamo a modificare il controller e inseriamo il control che aggancia il tap sul bottone salva, poi aggiungiamo la funzione che permette il salvataggio saveNote().

Per prima cosa inseriamo il riferimento alla lista delle note e il control sul bottone Salva.

```
    refs: {
      ...
      notes: '#notesList',
    },
    control: {
      ...
      'button[action=savenote]': { tap: 'saveNote'},
    }
```

Vediamo di seguito la funzione per il salvataggio della nota, da aggiungere sempre al controller.

```
saveNote: function(){
    newNote = this.getNoteEditor();
    currentNote = newNote.getRecord();
    newNote.updateRecord(currentNote);

    error = currentNote.validate();
    if (!error.isValid()) {
      currentNote.reject();
      Ext.Msg.alert('Attenzione!', error.getByField('title')[0].getMessage
          (), Ext.emptyFn);
      return;
    }

    list = this.getNotes();
    notesStore = list.getStore();

    if  (notesStore.findRecord('id', currentNote.data.id) === null) {
      notesStore.add(currentNote);
    }
    else {
      currentNote.setDirty();
    }
    notesStore.sync();
    notesStore.sort([{property:'date', direction: 'DESC'}]);
    list.refresh();
    this.getMain().getLayout().setAnimation({ type: 'slide', direction: '
        right', duration: 200 });
    this.getMain().setActiveItem(this.getNotesList());
},
```

Per prima cosa impostiamo la variabile newNote come riferimento alla view NoteEditor. Usiamo poi il metodo getRecord() del pannello form per ottenere un riferimento alla nota caricata nel form, col metodo updateRecord() trasferiamo i valori dal form all'oggetto currentNote, che é un modello di nota.

Per la validazione usiamo l'oggetto error generato dalla chiamata validate() su currentNote. In caso di errore la chiamata isValidate() restituirà true, verrà quindi eseguito il codice nell'if che crea un alert al centro dello schermo, col messaggio impostato precedentemente nel model, e verrà fermata l'esecuzione del metodo, vediamo il risultato in figura 3.2.7.

Se invece non ci sono errori l'esecuzione del programma continua e vengono creati due nuovi oggetti, uno riferito alla lista e uno allo store, quest'ultimo per leggere e salvare i dati da localstorage. A questo punto controlliamo che la nota non sia già presente nella cache gazie al metodo findRecord() dello store, in caso di risposta negativa la nota viene inserita nello store, se invece la nota é già presente viene contrassegnata come dirty, col metodo setDirty(), in questo caso verrà aggiornata la cache, coi dati modificati, alla successiva

Figura 3.2.7: Alert per titolo nota mancante

sincronizzazione dello store con la cache, cosa che viene fatta subito dopo col metodo sync(). In seguito viene riordinato lo store in base alla data col metodo sort().

Per finire aggiorniamo la lista col metodo refresh() e, come nei casi precedenti, effettuiamo lo slide per tornare alla view NotesListContainer.

Giunti a questo punto l'applicazione comincia già a prendere forma, però mancano ancora due funzioni, la possibilità di aprire una nota salvata e quella di eliminarla. Cominciamo dalla visualizzazione di una nota salvata.

Vogliamo che venga visualizzata la nota quando viene fatto il tap sull'elemento della lista. Per fare questo dobbiamo inserire un altro elemento di rilevamento nel controller, in particolare il seguente.

```
    ...
    control: {
      ...
      'notes': { itemtap: 'showNote'}
    }
```

Questo control rileva nella lista l'evento itemtap e chiama il metodo showNote, che vediamo di seguito.

```
  showNote: function(view, index, target, record){
    nota = this.getNoteEditor();
    nota.setRecord(record);
    this.getMain().getLayout().setAnimation({ type: 'slide', direction: '
        left', duration: 200 });
    this.getMain().setActiveItem(nota);
  },
```

Alla funzione passiamo i parametri del metodo itemtap() della lista. Creiamo un modello di nota e inseriamo i dati prelevati dall'elemento della lista, poi impostiamo lo slide e visualizziamo la nota in noteEditor.

Adesso manca solo la possibilità di cancellare una nota, cominciamo facendolo dalla view noteEditor premendo il bottone col disegno del cestino. Come negli altri casi dobbiamo mettere il control dell'evento.

```
    ...
    control: {
      ...
      'button[action=cancel]': {tap: 'cancelNote'},
    }
```

Vediamo il metodo cancelNote().

```
cancelNote: function(){
  var currentNote = this.getNoteEditor().getRecord();
  list = this.getNotes();
  notesStore = list.getStore();
  if (notesStore.findRecord('id', currentNote.data.id)){
    notesStore.remove(currentNote);
  }
  notesStore.sync();
  list.refresh();
  this.getMain().getLayout().setAnimation({ type: 'slide', direction: '
      right', duration: 200 });
  this.getMain().setActiveItem(this.getNotesList());
},
```

Il funzionamente é relativamente semplice. Creiamo i riferimenti alla nota corrente, alla lista e allo store. Cerchiamo nello store la nota in base all'id e la rimuoviamo. Sincronizziamo lo store con la cache, facciamo il refresh della lista e torniamo a visualizzare la lista delle note aggiornata.

Arrivati a questo punto l'applicazione ha tutte le sue funzionalità, é possibile creare note, visualizzarle, modificarle ed eliminarle. Possiamo però aggiungere un secondo modo per cancellare le note, quello a cui sono ormai abituati tutti gli utenti dei più moderni smartphone, ossia facendo lo slide col dito sopra un elemento della lista. Per agganciare questa azione dobbiamo modificare il control sulla lista ottenendo il seguente risultato.

```
    ...
    control: {
      ...
      'notes': { itemtap: 'showNote', itemswipe: 'cancelNoteSwiping'}
    }
```

E in fine il codice del metodo cancelNoteSwiping().

```
cancelNoteSwiping: function(view, index, target, record){
    list = this.getNotes();
    Ext.Msg.confirm("Conferma", "Sei sicuro di voler eliminare la nota?",
        function(e){if(e == 'yes'){
      notesStore = list.getStore();
      if (notesStore.findRecord('id', record.data.id)){
        notesStore.remove(record);
      }
      notesStore.sync();
      list.refresh();
        }
      }
    );
  }
```

In questo caso vogliamo che ci venga chiesta conferma prima di eliminare la nota, per raggiungere il nostro scopo usiamo il metodo Ext.Msg.confirm che visualizza un messaggio e due bottoni (vedi figura 3.2.8), yes e no, con la possibilità di eseguire una determinata funzione solo nel caso in cui venga premuto il tasto yes. Naturalmente vogliamo che venga eseguita la funzione per cancellare la nota, che é praticamente identica a quella vista in precedenza. In questo caso non serve cambiare view, perché siamo già in quella giusta.

Figura 3.2.8: Messaggio di conferma.

Giunti a questo punto l'applicazione, nella sua semplicità, é completa.

A questo punto sarebbe possibile fare la build dell'applicazione per creare un file unico con tutta l'applicazione e le classi del framework usate, in modo tale da snellire il pacchetto in produzione.

Capitolo 4

Conclusioni

Abbiamo descritto il framework Sencha Touch e creato un'applicazione basilare, ma abbastanza esplicativa di quello che é possibile fare con il framework. Tuttavia le potenzialità di Sencha Touch sono molto più estese, dalla visualizzazione di video al salvataggio di immagini, dall'interazione con svariati server all'uso delle mappe e molto ancora.

Sencha Touch può rivelarsi molto comodo per la creazione di progetti multipiattaforma. Le potenzialità sono elevate e il continuo sviluppo di questa piattaforma lascia ben sperare per il futuro, con la possibilità di avere sempre un maggior numero di funzioni integrate.

Nel periodo in cui ho lavorato allo sviluppo dell'applicazione ho avuto la necessità di frequentare il forum ufficiale di Sencha e non solo e devo dire che la comunità di sviluppo é abbastanza estesa, le discussioni sono aggiornate continuamente e i developer di Sencha sono i primi a rispondere alle richieste di aiuto. Come detto all'inizio ho usato la versione 2.0 del framework, che é ancora in fase di sviluppo quindi ci sono delle funzioni non implementate, ma sopratutto ci sono alcuni bug che devono essere ancora risolti. Durante lo sviluppo dell'applicazione é stata fortunatamente presentata la prima beta del framework, che ha risolto alcuni errori che avevo riscontrato inizialmente. Si tratta di un progetto di notevole interesse per le sue potenzialità e per il vasto numero di utenti che può raggiungere.

4.1 PhoneGap

Tutto quello che abbiamo visto finora funziona benissimo sui più recenti browser. È possibile fare delle applicazioni web molto interessanti, ma per sfruttare appieno le capacità dei più recenti smartphone e tablet, bisogna poter accedere ai vari sensori e alle funzioni messe a disposizione, cosa non possibile lavorando dal browser. Per avere pieno accesso al dispositivo le applicazioni native potrebbero essere la risposta, ma in questo caso si perde tutto il lavoro fatto con Sencha e diventa necessario creare un'applicazione per ogni piattaforma. Un lavoro molto impegnativo per lo sviluppatore.

Esiste un framework, non legato a Sencha, che riesce a far coesistere lo sviluppo in un solo linguaggio con la possibilità di creare applicazioni native per ogni dispositivo: questo é PhoneGap. Tale framework é basato su HTML5, CSS3 e JavaScript, esattamente i linguaggi che servono a Sencha Touch per lavorare. Concettualmente questo framework fa da ponte tra le API dei vari dispositivi e l'applicazione creata con Sencha: in questo modo é possibile sfruttare appieno le funzioni dei dispositivi come fotocamera, GPS, accelerometri e altri ancora. Inla figura 4.1.1 é possibile vedere con quali sistemi é compatibile PhoneGap e quali features supporta per ogni dispositivo.

	iOS iPhone / iPhone 3G	iOS iPhone 3GS and newer	Android	OS 4.6-4.7	OS 5.x	OS 6.0+	WebOS	WP7	Symbian	Bada
ACCELEROMETER	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
CAMERA	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
COMPASS	✗	✓	✓	✗	✗	✗	✗	✓	✗	✓
CONTACTS	✓	✓	✓	✗	✓	✓	✗	✓	✓	✓
FILE	✓	✓	✓	✗	✓	✓	✗	✓	✗	✗
GEOLOCATION	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MEDIA	✓	✓	✓	✗	✗	✗	✗	✓	✗	✗
NETWORK	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NOTIFICATION (ALERT)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NOTIFICATION (SOUND)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NOTIFICATION (VIBRATION)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
STORAGE	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗

Figura 4.1.1: PhoneGap supported features

Elenco delle figure

Bibliografia

[1] Sencha Touch *Mobile JavaScript Framework*, http://www.sencha.com/products/touch/

[2] Sencha Touch 2 Developer Preview, http://www.sencha.com/blog/sencha-touch-2-developer-preview/

[3] Sencha Touch: un framework HTML5 per dispositivi mobili, http://www.melablog.it/post/11991/sencha-touch-un-framework-html5-per-dispositivi-mobili

[4] Sandro Paganotti, Sencha touch: applicazioni mobile HTML5 senza sforzo, http://javascript.html.it/articoli/leggi/3703/sencha-touch-applicazioni-mobile-html5-senza-sforzo/

[5] HTML, http://www.w3.org/TR/html5/

[6] Storia del WWW, nascita di HTML, il W3C, http://universityblog.splinder.com/post/20391609/storia-del-wwwnascita-htmlil-w3c

[7] W3C Mission, http://www.w3.org/Consortium/mission.html

[8] Leopoldo Saggin, Generalità sull'HTML, http://fog.bio.unipd.it/corso_html/tags.html

[9] XML, http://www.w3.org/XML/

[10] XHTML, http://www.w3.org/TR/xhtml1/

[11] Silvio Porcellana, Che cos'é HTML5, http://www.html5today.it/tutorial/che-cos-html5

[12] CSS, http://www.w3.org/TR/CSS/

[13] Cesare Lamanna, Simone D'Amico, Guida CSS3, http://css.html.it/guide/leggi/208/guida-css3/

[14] Gaelle Guelton, Proprietà CSS3 alla prova, http://www.onsitus.it/css3/

[15] Tom Negrino - Dori Smith, Javascript: Visual QuickStart Guide (8th Edition), Peachpit Press, 2011

[16] Javascript, https://developer.mozilla.org/en/JavaScript/About_JavaScript

[17] How PhoneGap Works, http://phonegap.com/about

[18] Sencha Touch, Documentation, http://docs.sencha.com/touch/2-0/

[19] WampServer, `http://www.wampserver.com/en/`

[20] MAMP, `http://www.mamp.info/en/mamp/index.html`