# UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Facoltà di Ingegneria "Enzo Ferrari" – Sede di Modena

Corso di Laurea Specialistica in Ingegneria Informatica


# Progettazione e Sviluppo di un'Applicazione per Dispositivi Mobili basati sul Sistema Operativo iOS per la Gestione e Pianificazione di Attività


**Relatore:**

Prof.ssa Sonia Bergamaschi

**Correlatore:**

Prof. Riccardo Martoglia

**Tesi di Laurea di:**

Giovanni Simonini



Anno Accademico 2010/2011

**Ringraziamenti**

*Ringrazio primi tra tutti la Prof.ssa Sonia Bergamaschi e il Prof. Riccardo Martoglia per avermi seguito in questo periodo di tesi, visto che senza di loto, ovviamente, non sarei qui a scrivere questi ringraziamenti.*

*Per quanto riguarda il sostegno morale, ringrazio mia madre che mi ha sempre sostenuto, spronato e soprattutto sopportato, appoggiandomi incondizionatamente in ogni mia decisione. Ringrazio mio padre, al quale in fin dei conti credo faccia piacere avere un figlio ingegnere, anche se così non potrà più far finta di lamentarsi delle tasse universitarie. Ringrazio la nonna, vera motivatrice della famiglia, il cui sostegno è stato fondamentale in questi anni. Ringraziamento, purtroppo postumo, anche al signor zio Gherardo.*

*Ringraziamento speciale, alla quale devo una decina di punti del voto finale di laurea, il sergente di ferro, la mia ragazza, Giulia (Bio) che mi ha sempre spronato a pretendere il massimo da me.*

*Ringrazio tutti i miei amici di sempre, e quelli che ho conosciuto strada facendo. Uno speciale ringraziamento anche ai proprietari ea ai frequentatori della casa del seme, come dimenticarvi…*

*Grazie a tutti.*

**Parole Chiave:**
*iOS*
*Appcelerator*
*Titanium*
*JavaScript*
*SQLite*

**INDICE**

**PARTE I**

**Introduzione** ..... 3

**1 – Studio di Fattibilità di una App IOS per il Supporto Intelligente all' Agenda Personale: Descrizione Generale e Funzionalità** ..... 5
**1.1 - Campo di utilizzo** ..... 5
**1.2 - Descrizione generale dell'App** ..... 8
**1.3 – Confronto con prodotti simili sul mercato** ..... 10

**2 - Stato dell'Arte** ..... 11
**2.1 - Dispositivi Mobili: Smartphone e Tablet** ..... 11
**2.2 - Sistemi Operativi per Dispositivi Mobili** ..... 12
2.2.1 - Distribuzione del mercato dei sistemi operativi mobili ..... 13
2.2.2 - Sistemi Open source vs Proprietari ..... 13
**2.3 - Le applicazioni per Dispositivi Mobili: le App e gli strumenti per svilupparle** ..... 14
**2.4 - Statistiche del mercato delle App** ..... 15

**3 - Strumenti per lo Sviluppo delle App in iOS.** ..... 17
**3.1 - iOS SDK e Cocoa Touch** ..... 17
**3.2 - Multiple Phone web-based Framework** ..... 18
3.2.1 - Classificazione ..... 19
3.2.2 – Confronto tra cross-platform frameworks ..... 20
3.2.3 - Appcelerator Titanum ..... 23

**4 - Sviluppo di un App con Appcelerator Titanium** ..... 26
**4.1 - Struttura di un progetto in Titanium** ..... 26
**4.2 - Com'è generata l'applicazione: pre-compiler e front-end-compiler** ..... 27
**4.3 - JavaScript in Titanium** ..... 29
4.3.1 - Tecniche di Programmazione JavaScript per Titanium ..... 30
4.3.2 - Librerie aggiuntive JavaScript per Titanium ..... 32
**4.4 - Storing Data in Titanium** ..... 34
4.4.1 - App Properties ..... 34
4.4.2 - SQLite in Titanium ..... 35
4.4.3 – Filesystem ..... 36

**PARTE II**

**5 – Progettazione App per Gestione e Pianificazione di Attività CalPlanner** ..... 38
**5.1 – Definizioni** ..... 38
**5.2 – Requisiti Funzionali** ..... 39
**5.3 – Progettazione Database** ..... 44
5.3.1 – Modello E-R ..... 44
5.3.2 – Studio dei Dati Derivati ..... 46
**5.4 – Struttura Modulare dell'Applicazione** ..... 49
5.4.1 – Interfaccia Utente ..... 51
5.4.2 – Logica Funzionale e Gestione dei Dati ..... 53
**5.5 – Problematiche Progettuali** ..... 54
5.5.1 – Date, Orari e Standard Impiegati per la Memorizzazione ..... 54
5.5.2 – Modulo Calendario per Titanium ..... 55

5.5.3 – Realizzazione della Vista Totale ..... 57
**6 – Realizzazione del Progetto ..... 59**
**6.1 – Il Database ..... 59**
6.1.1 – I Trigger ..... 60
6.1.2 – SQL Query Per la Verifica di Disponibilità Temporale ..... 61
**6.2 – Il Progetto in Titanium ..... 62**
6.2.1 – Struttura del Progetto ..... 63
6.2.2 – Implementazione delle Librerie ..... 65
6.2.3 – Implementazione della Vista Totale ..... 73
**6.3 – User Interface ..... 75**
6.3.1 – Moduli delle Componenti Grafiche Personalizzate ..... 76
6.3.2 – Le Finestre delle Funzioni ..... 77
**Conclusioni e Sviluppi Futuri ..... 86**
**Appendice A: Utilizzo dell'ereditarietà in JavaScript ..... 90**
**Bibliografia ..... 92**

# Introduzione

Il mercato degli Smartphone e Tablet è nato e si è evoluto negli ultimi anni. La diffusione di questi dispositivi mobili è stata esponenziale e non accenna a diminuire.

Il loro successo è dovuto anche alle "app"; infatti ogni produttore di sistemi operativi per dispositivi mobili mette a disposizione strumenti software per lo sviluppo di programmi appositi per le proprie piattaforme mobili; così è nato parallelamente a quello dei devices anche un mercato per le loro app.

Smartphone e Tablet vengono sempre più impiegati anche come agende personali digitali, dove gli utenti memorizzano i loro impegni e che vengono utilizzate per pianificare attività ed eventi, magari con l'integrazione con strumenti *on the cloud* come iCloud o Google Calendar, accessibili da qualsiasi dispositivo, mobile o fisso.

L'idea che sta alla base dell'app **CalPlanner** presentata in questa tesi è nata dall'esigenza di classificare gli eventi presenti su un calendario digitale e verificare il rispetto di alcuni vincoli per la rendicontazione delle attività di docenti e ricercatori universitari. Esigenze che si presentano anche in altri contesti lavorativi, soprattutto legati alla pianificazione di attività.

Partendo da questa idea si è voluto realizzare un'app che offrisse strumenti per il mantenimento di statistiche e viste riassuntive per la pianificazione e la rendicontazione delle attività basate su classi di eventi. Inoltre fornisce gli strumenti per affiancare l'utente nella gestione del proprio tempo, offrendo funzionalità ausiliarie che lo guidano nella pianificazione delle proprie attività, coerentemente con le proprie esigenze e con gli impegni che già gestisce con altri strumenti, ad esempio, calendari digitali come iCal e Google Calendar.

Per lo sviluppo dell'app CalPlanner è stato scelto di impiegare tecnologie nuove, che si basano su linguaggi tipici del web dinamico: JavaScript, Pyton, HTML e CSS. In particolare si è scelto Appcelerator Titanium che consente di sviluppare con un unico framework app sia per dispositivi mobili basati su iOS che Android (i due sistemi operativi per dispositivi mobili più diffusi).

La tesi è organizzata in due parti.

Nella prima parte, è descritto lo scenario di utilizzo dell'applicazione CalPlanner e ne è data una descrizione generale. Inoltre, vengono evidenziate le differenze con applicazioni simili già esistenti sul mercato (cioè possibili concorrenti). In seguito, sono presentati gli strumenti software utilizzati per la realizzazione del progetto, le tecniche di programmazione impiegate e le motivazioni che hanno portato a queste scelte. In particolare l'ultimo capitolo della prima parte è interamente dedicato alla programmazione in Titanium e JavaScript, che costituiscono il cuore degli strumenti software adottati.

Nella seconda parte, sono descritti formalmente i requisiti dell'app e del suo database. Sono definite le caratteristiche e funzionalità dei moduli necessari alla realizzazione del progetto e sono motivate le scelte progettuali, tenendo conto anche delle problematiche emerse in ogni fase (sia in quella di progettazione che di realizzazione) e delle soluzioni adottate. Infine, è descritta l'implementazione dell'app CalPlanner vera e propria in Titanium: librerie sviluppate, i moduli aggiuntivi e l'interfaccia utente.

L'ultimo capitolo è dedicato alle conclusioni ed ai possibili sviluppi futuri del progetto CalPlanner. In particolare, le note conclusive elencando gli obiettivi raggiunti e le valutazioni sulla piattaforma Titanium. Negli sviluppi futuri, sono proposte le funzionalità che permetterebbero l'utilizzo dell'app per la pianificazione di eventi ed attività in un contesto multiutente. Viene inoltre proposta una breve analisi di come possa essere estesa la app per essere utilizzato come strumento di ausilio per la pianificazione di strategie da parte della forza vendita di un'impresa. Questo scenario di utilizzo è stato suggerito dal Prof. Gianluca Marchi, docente della facoltà di economia Marco Biagi dell'università Modena, che ringrazio vivamente.

# 1 – Studio di Fattibilità di una App IOS per il Supporto Intelligente all'Agenda Personale: Descrizione Generale e Funzionalità

In questo capitolo sono descritte le esigenze che l'applicazione per il supporto intelligente all'agenda personale si propone di soddisfare. Come prima cosa sarà descritto il campo di utilizzo, cioè lo scenario generale all'interno del quale l'app si colloca e le problematiche tipiche di esso. Con il termine app ci si riferisce ad un'applicazione software per dispositivi mobili (Smartphone e Tablet). In seguito saranno presentate le funzionalità; per ultimo verrà fatto un confronto con prodotti già esistenti sul mercato, infine verrà mostrato un confronto con alcune applicazioni che si inseriscono nello stessa categoria.
Lo scopo del capitolo è, comunque, di dare una descrizione generale dell'applicazione; le descrizioni funzionali e le scelte progettuali per realizzarla saranno invece oggetto dei capitoli successivi.

## 1.1 - Campo di utilizzo

L'applicazione **CalPlanner** si colloca nella categoria delle "utility" per dispositivi mobili iOS.

Mette a disposizione dell'utente strumenti per la gestione ed organizzazione di eventi ed attività, ed offrire una visualizzazione personalizzata di statistiche ed informazioni relative ad essi. Può essere sincronizzata con il calendario nativo del dispositivo mobile sul quale è installata e mantiene un proprio database di eventi e attività.

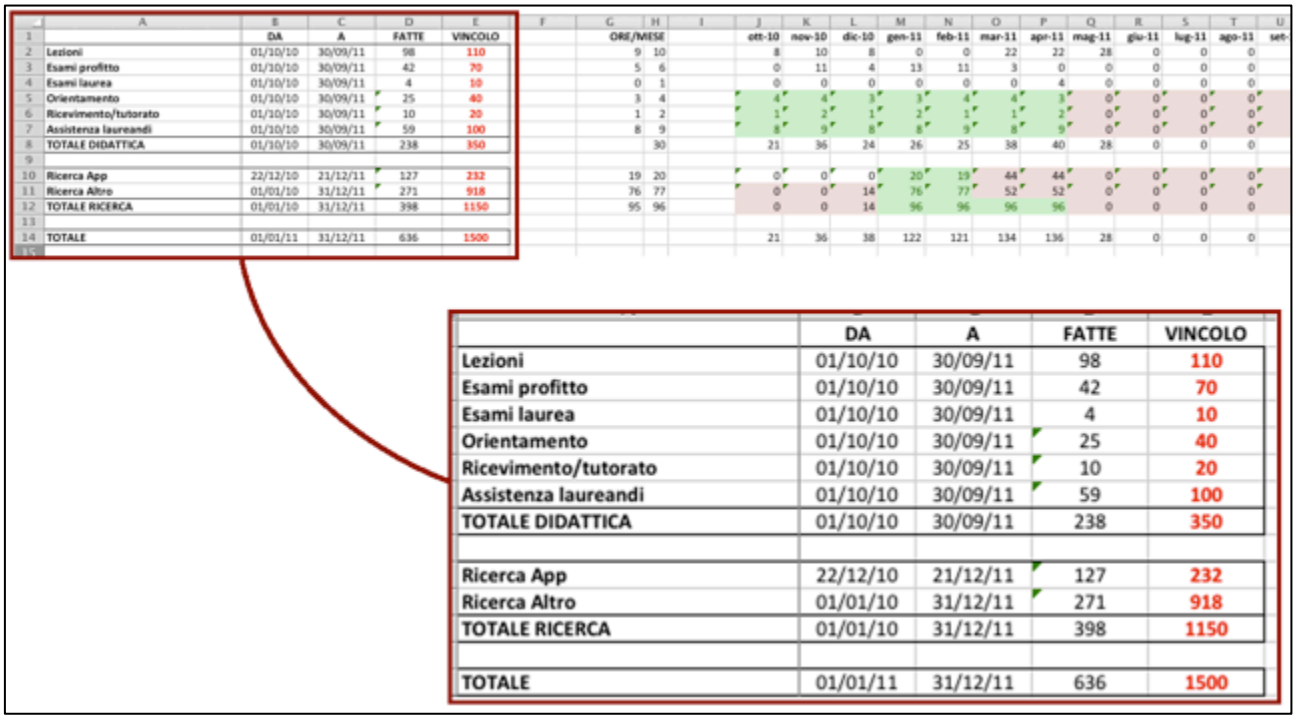
CalPlanner prevede l'esistenza di *tipi di evento*: classi di attività, eventi o impegni; ogni evento memorizzato nel calendario è un'istanza di una determinata classe di *tipo di evento*. Ogni *tipo di evento* prevede determinati vincoli definibili dall'utente al momento della creazione della classe, o modificabili successivamente. Questi vincoli possono riguardare il numero minimo o massimo di ore previste per una determinata attività in un determinato arco temporale; il numero di ore totali da raggiungere o il periodo di validità. L'applicazione deve anche impedire la sovrapposizione degli eventi laddove necessario. Di questi *tipi di eventi* vengono mantenute quindi statistiche sulle percentuali di completamento e di evasione dei vincoli.

L'applicazione vuole anche offrire strumenti per la consultazione rapida di determinati tipi di impegni presenti sui calendari nativi dei dispositivi (eventualmente sincronizzati con altri calendari come Google Calendar).

Nelle immagini (figura 1 e figura 2) viene mostrato un esempio del foglio excel dal quale si è preso spunto per la realizzazione di alcune funzionalità dell'applicazione. Questo foglio excel (foglio-excel) creato dal Dott. Riccardo Martoglia, viene utilizzato per mantenere il conteggio delle ore degli impegni accademici tipici di un docente o ricercatore universitario; prevede vincoli di controllo temporali ed ha una duplice funzione:

- Aiutare l'utente a ottimizzare l'organizzazione del proprio tempo.
- Fornire una vista facilmente consultabile e modificabile, riassuntiva di tutti gli impegni appartenenti ad un determinato arco temporale.

Questo strumento può essere molto utile per la compilazione di rendiconti tipici di un ateneo, richiesti ad esempio al termine di ogni anno accademico; ma potrebbe essere impiegato in ambito aziendale con le stesse finalità.



| | DA | A | FATTE | VINCOLO |
|---|---|---|---|---|
| Lezioni | 01/10/10 | 30/09/11 | 98 | 110 |
| Esami profitto | 01/10/10 | 30/09/11 | 42 | 70 |
| Esami laurea | 01/10/10 | 30/09/11 | 4 | 10 |
| Orientamento | 01/10/10 | 30/09/11 | 25 | 40 |
| Ricevimento/tutorato | 01/10/10 | 30/09/11 | 10 | 20 |
| Assistenza laureandi | 01/10/10 | 30/09/11 | 59 | 100 |
| TOTALE DIDATTICA | 01/10/10 | 30/09/11 | 238 | 350 |
| | | | | |
| Ricerca App | 22/12/10 | 21/12/11 | 127 | 232 |
| Ricerca Altro | 01/01/10 | 31/12/11 | 271 | 918 |
| TOTALE RICERCA | 01/01/10 | 31/12/11 | 398 | 1150 |
| | | | | |
| TOTALE | 01/01/11 | 31/12/11 | 636 | 1500 |

**Figura 1:** Un esempio del foglio excel per la gestione dei vincoli per rendiconti accademici. Nel dettaglio le righe e colonne adibite all'inserimento dei vincoli.

### *Foglio-excel*

In figura 2 viene mostrato la vista totale di *foglio-excel* presa di riferimento per la realizzazione della funzionalità *vista totale* dell'applicazione. La tabella è organizzata nel seguente modo:

- Righe: *tipi di evento* (classi di evento).
- Colonne: giorni e date di un determinato arco temporale.
- Celle: ore svolte di un determinato tipo di evento in una determinata data. Questo numero è modificabile.
- Ultima riga: totale delle ore per ogni giorno

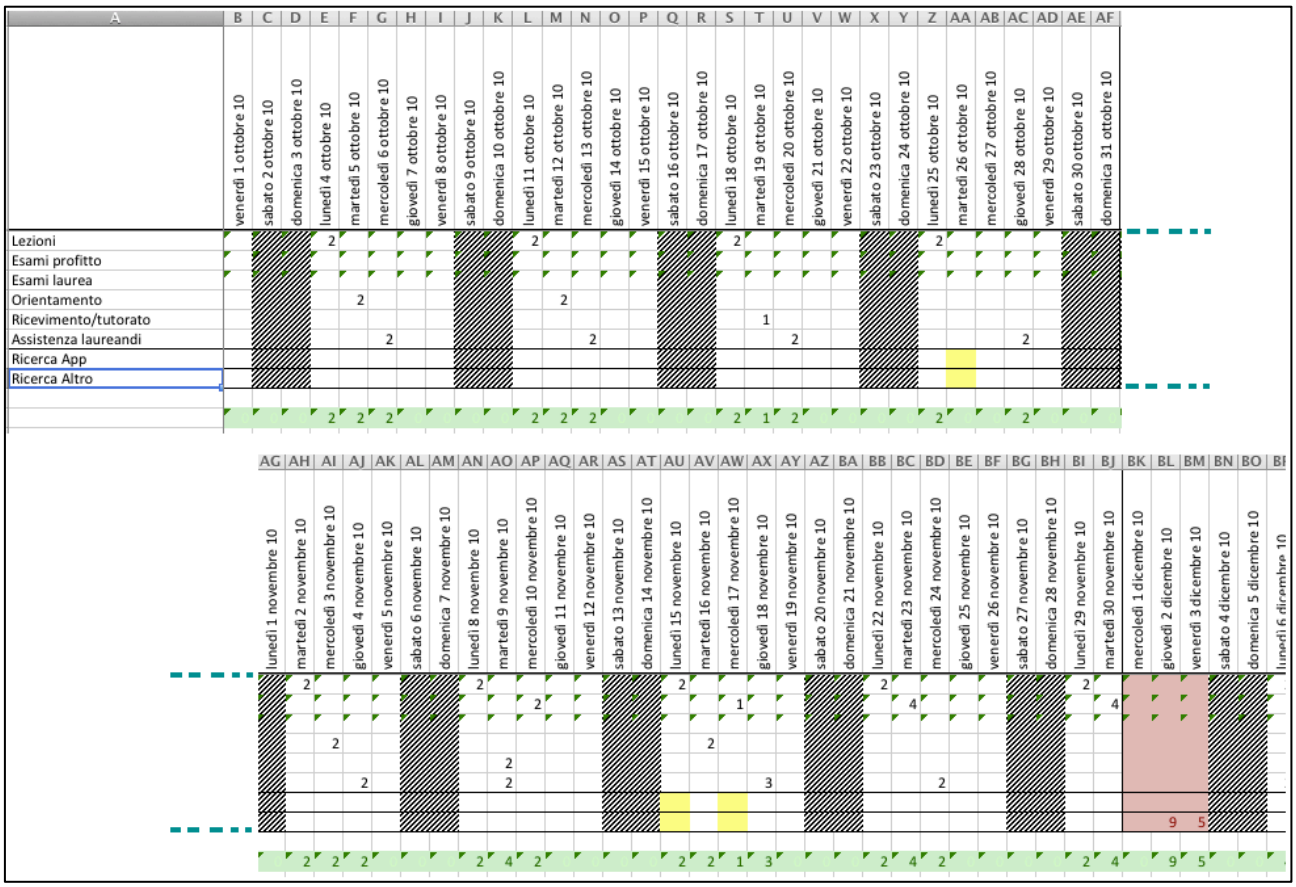


**Figura 2:** Viste riassuntiva consultabile e modificabile del foglio excel. Ogni cella contiene il numero di ore per il giorno indicato in colonna relativo al *tipo di evento* indicato nella riga; il numero all'interno della casella è anche modificabile, ed in caso di evasione di un vincolo viene segnalato visivamente nell'ultima riga (totali). Sono segnalati anche i giorni non lavorativi, nei quali on è possibile inserire ore di alcun tipo di evento, e sono riportati i giorni speciali, nei quali può essere inserito solamente un determinato tipo di evento.

## 1.2 - Descrizione generale dell'App

Lo scopo dell'app è di offrire funzionalità di ausilio al calendario già utilizzato dall'utente per memorizzare sul dispositivo mobile i propri impegni. Offre funzionalità di memorizzazione e sincronizzazione di nuovi eventi o gruppi di eventi, ma l'obiettivo finale è quello di integrarsi con altri strumenti, così da risultare meno invasivo ed adattarsi a differenti esigenze degli utenti.

Di seguito sono elencate le caratteristiche principali dell'applicazione:

- **Definizione di classi di eventi**
  L'app prevede la possibilità di definire *tipi di eventi*, cioè classi di attività sulle quali l'utente vuole imporre vincoli di controllo.
- **Mantenimento vincoli riguardanti le esigenze dell'utente**
  L'utente può imporre dei vincoli temporali per ogni *tipo di evento*. L'applicazione segnala quando questi vincoli sono rispettati e quando no.
- **Inserimento rapido eventi**
  L'utente può inserire rapidamente con l'apposita funzione un evento rapido appena terminato.
- **Importazione eventi da calendario**
  L'utente può importare eventi dal calendario nativo del dispositivo e farli confluire in un unico *tipo di evento*.
- **Consultazione rapida statistiche**
  Una sezione dell'app permette la consultazione rapida delle statistiche relative alle classi di eventi.
- **Vista totale di consultazione rapida**
  Con apposite viste l'utente può consultare una vista generata dinamicamente che contiene le ore dei tipi di eventi da lui indicati in un determinato arco di tempo. (vista *foglio-excel*)
- **Suggerimento di proiezioni per pianificazione attività**
  Con un'apposita funzione è possibile generare una vista analoga a "vista totale di consultazione rapida" per pianificare un'attività nel tempo coerentemente con gli impegni e i vincoli già esistenti. Impostando vincoli come: ore/settimana, ore/mese, fasce orarie, giorni della settimana, ore totali, ore minime e ore per evento, è possibile

generare una vista di un piano di suddivisione delle ore di una determinata classe di evento.

In figura 3 si può vedere una rappresentazione schematica delle funzioni di alcune funzionalità (*inserimento rapido* e *visualizzazione totale*).

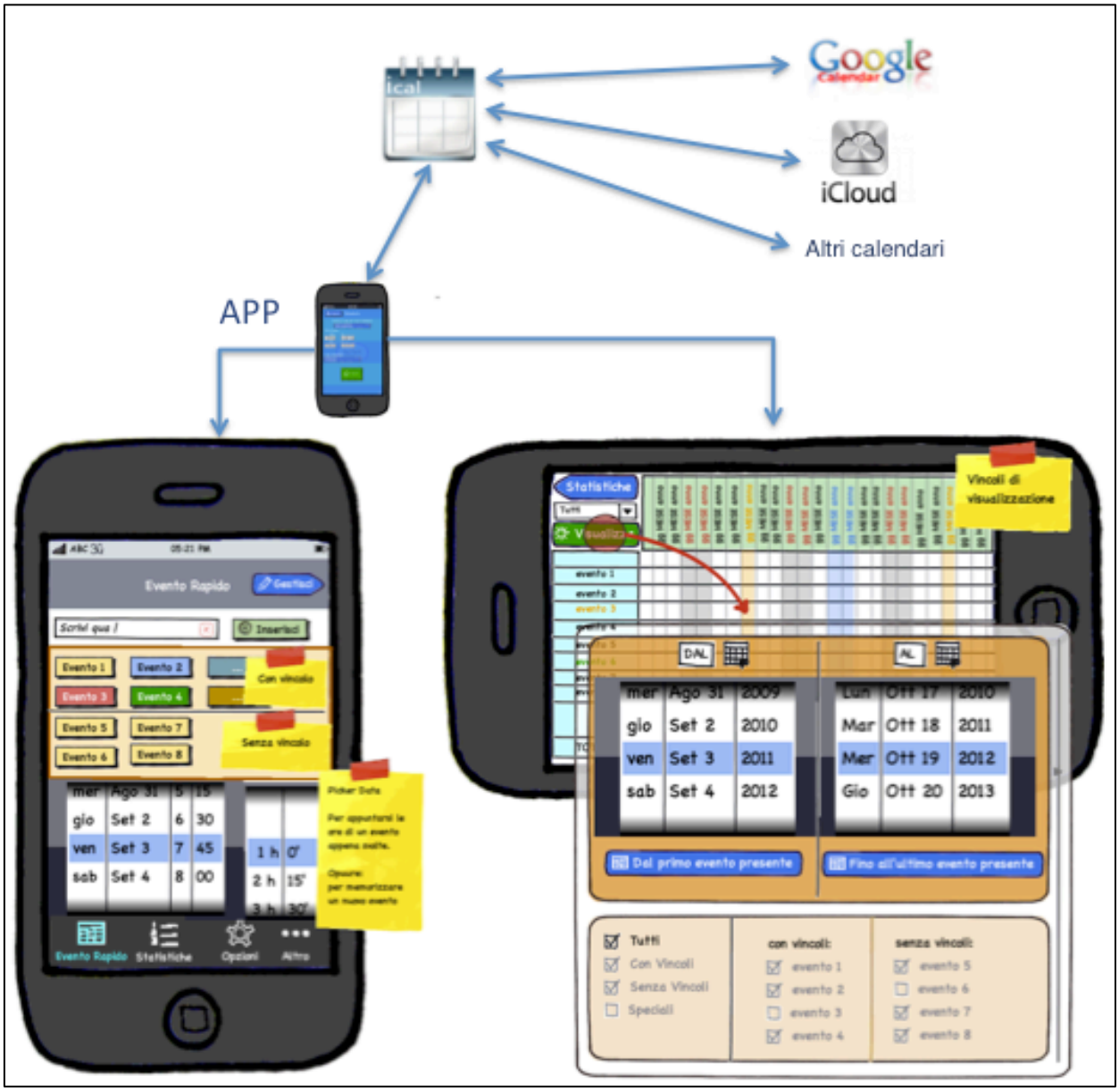



**Figura 3:** Rappresentazione schematica di alcune funzionalità dell'app. Si può sincronizzare con il calendario nativo del dispositivo; permette l'inserimento di eventi e il mantenimento di "classi di evento"; genera dinamicamente viste schematiche per la consultazione delle attività già svolte o da programmare.

## 1.3 – Confronto con prodotti simili sul mercato

Tutte le applicazioni sono classificate in base al loro campo di utilizzo e alle loro funzionalità. **CalPlanner** può rientrare nelle categorie:

- Utility
- Produttività
- (Economia)

Di seguito sono illustrate alcune applicazioni presenti su "App Store" che si collocano nelle stesse categorie e che presentano alcune funzionalità simili. Viene fatto un breve confronto evidenziando le differenze con CalPlanner.

***TimeTracker***

Questa app permette di creare dei progetti (attività), suddividerli in task mantenendo un contatore di ore per ogni tipo di attività. Non è possibile importare eventi dal calendario nativo. I vincoli imponibili sono limitati: data di inizio e data di fine. Non offre una vista simile a quella proposta in foglio-excel.

Non importando eventi da calendario non permette di creare programmazioni di eventi coerenti con gli impegni già programmati dall'utente con altri strumenti.

***Time Tracker***

Questa app è in grado di sincronizzarsi con il calendario nativo dei dispositivi mobili. Permette la definizione di attività (tipi di evento) che si possono ripetere nel tempo e ne mantiene un contatore di ore. Offre una vista a griglia totale delle attività presenti in memoria simile a quella di foglio-excel. Non permette però di far confluire eventi provenienti da calendario in determinate classi di eventi definite dagli utenti, in questo modo può mantenere le statistiche solo degli eventi creati dall'applicazione e non importati. Non permette l'imposizione di vincoli per le attività oltre all'assegnazione di una data di inizio e una di fine. Le viste che crea sono statiche, poco compatte e non modificabili: nelle celle viene indicato solamente il numero di ore svolte in una determinata data, ma questo numero non è modificabile.

***SmartPlans + SmartTime***

Applicazioni per la pianificazione e gestione di progetti. Appartengono alla stessa suite di applicazioni e possono essere sincronizzate tra di loro. Permettono la sincronizzazione con il calendario nativo del dispositivo (iCal). Prevedono la definizione di vincoli, così da mantenere statistiche relative alle attività gestite. Mantengono contatori di ore per ogni attività o progetto. Permettono di pianificare attività ed eventi tenendo conto degli eventi già presenti in iCal. Non permettono però di far confluire determinati eventi di calendario già esistenti in nuovi progetti/attività. Offrono viste ed interfacce utente per statistiche e vincoli, ma non simili a quella di fogli-excel.

# 2 - Stato dell'Arte

In questo capitolo è mostrato lo stato dell'arte dei dispositivi mobili, smartphone e tablet, e dei loro sistemi operativi.

## 2.1 - Dispositivi Mobili: Smartphone e Tablet

Con dispositivo mobile si intendono tutti quei dispositivi portatili di ridotte dimensioni dotati di un proprio sistema operativo e batterie che permettono il loro utilizzo in mobilità. Nella più larga accezione del termine possono rientrare in questa categoria anche i computer portatili, ma nell'uso comune del termine si identificano in questa categoria: smartphone, PDA e tablet.

Sono generalmente dotati di piccole tastiere integrate ed uno schermo lcd. Quest'ultimo può essere anche di tipo touch-screen e sostituire in questo caso il tastierino. La grandezza degli schermi generalmente varia dai 2/3 pollici di alcuni smartphone fino ad un massimo 14 pollici per i tablet . Esiste tuttavia una vasta gamma di prodotti sul mercato che può rientrare solo in parte in questa classificazione, essendo questo un settore relativamente giovane (la prima

versione di iPhone è stata presentata a metà del 2007, iPad all'inizio del 2010; la prima release di Android è del 2008) ed in continua evoluzione.

Dal punto di vista della dotazione hardware possono presentare varie configurazioni che permettono:

- connessione a reti mobili (GPS/GPRS/EDGE/UMTS/HSDPA/ HSUPA): smartphone e alcuni tipi di tablet.
- connessione a reti LAN (Local Area Network, tecnologie 802.11 g/b): smartphone e tablet.
- Connessione a reti PAN (Personal Area Network, tecnologia bluetooth): smartphone e tablet. Per la connessione a dispositivi esterni (auricolari, stampanti, altri dispositivi mobili).
- Memorizzazione di dati in memoria flash: sia gli smartphone che i tablet sono dotati di memoria integrata che può andare dai 256MB ai 64GB; generalmente espandibili con schede di memoria esterne (SD, microSD).

Caratteristica distintiva è anche quella di avere sistemi operativi molto più evoluti rispetto ai classici telefoni cellulari che permettono a sviluppatori di terze parti, e non solo ai produttori dei dispositivi, di implementare applicativi installabili.

## 2.2 - Sistemi Operativi per Dispositivi Mobili

I sistemi operativi di smartphone e tablet si sono evoluti nel tempo di pari passo con la dotazione hardware dei dispositivi mobili integrando nel tempo sempre più funzionalità e diventando sempre più simili ai sistemi operativi dei personal computer general purpose.

Attualmente i sistemi operativi più diffusi nel mercato mondiale sono:

- Android (Google)
- iOS (Apple iPhone, iPad, iPod Touch)
- Symbian (Nokia)
- RIM BlackBerry (Research In Motion)
- Windows Phone 7 (Microsoft)

### 2.2.1 - Distribuzione del mercato dei sistemi operativi mobili

Di seguito, in figura 4, viene mostrata la suddivisione del mercato dei sistemi operativi installati sui dispositivi mobili. Android e iOS si sono affermati negli ultimi due anni come i sistemi operativi più diffusi, a scapito soprattutto di Symbian, fino a pochi anni fa leader del mercato.

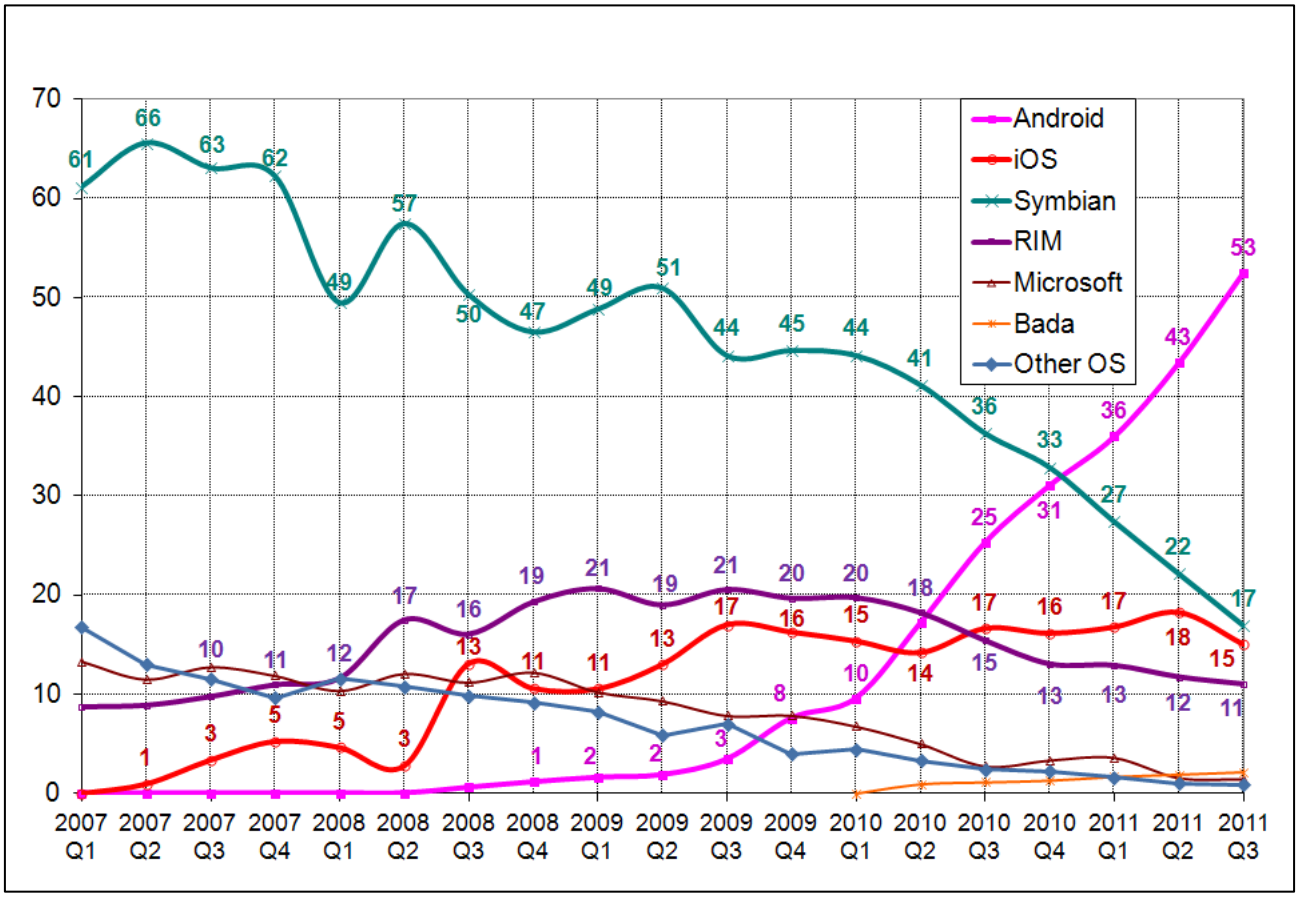



**Figura 4:** Asse delle ascisse: quadrimestri; asse delle ordinate: percentuale si suddivisione del mercato. Distribuzione sul mercato dei dispositivi mobili (smartphone, PDA e tablet) dei principali sistemi operativi mobili. Come si può osservare dal grafico è in costante ascesa la diffusione di Android a scapito soprattutto di Symbian; iOS invece conserva negli ultimi anni costantemente la sua quota di mercato come RIM. I dati relativi a Microsoft si riferiscono al sistema operativo "windows mobile", attualmente Microsoft ha rilasciato il nuovo sistema operativo "windows phone 7"; quest'ultimo è anche diventato il sistema operativo principale dei dispositivi Nokia, la quale a sua volta ha abbandonato lo sviluppo del sistema Symbian. [1][2].

### 2.2.2 - Sistemi Open source vs Proprietari

Alcuni sistemi operativi sono stati sviluppati dagli stessi *vendor* dei dispositivi mobili sui queli sono installati; sono proprietari e si trovano solo su quei dispositivi, non possono essere installati su dispositivi di altre case produttrici. E' questo il caso di iOS e Black Berry OS; mentre la politica opposta è stata adottata invece da Android.

***iOS e BlackBerry OS.***

I codici sorgenti di questi software sono protetti dal copyright e non sono modificabili da sviluppatori di terze parti. Sono pubbliche solamente le APIs con le quali si possono sviluppare applicazioni che devono essere però sottoposte alla verifica delle case produttrici del sistema operativo per essere poi approvate e distribuite tramite i canali di diffusione ufficiali.
Questi sistemi operativi possono essere installati solamente sui dispositivi prodotti dagli stessi vendor: Apple per iOS, installabile solo sui dispositivi prodotti da Apple basati sui processori di tipo ARM; RIM per BlackBerry OS.

***Android***

Android è un sistema operativo open source; questo significa che chiunque può modificarlo per creare delle nuove versioni del software. E' stato sviluppato da Google a partire dal kernel *linux* e si può trovare installato su dispositivi mobili di differenti case produttrici; le architetture di processori supportate sono: ARM, MIPS e x86.

## 2.3 - Le applicazioni per Dispositivi Mobili: le App e gli strumenti per svilupparle

Tutti i sistemi operativi mobili mettono a disposizione delle APIs (*Application Programming Interfaceses*) per vari linguaggi di programmazione che permettono a sviluppatori di terze parti di creare applicazioni installabili: le "app".
Le APIs permettono di sfruttare le varie funzionalità che mettono a disposizione i dispositivi mobili e i loro sistemi operativi come:

- Accesso a reti mobili ed internet
- Invio e ricezione di SMS/MMS
- Fotocamera
- Strumenti di geolocalizzazione (GPS e basata su IP)
- Gestione file multimediali
- Web browsing ed email

- Accesso e gestione della memoria
- Gestione dell'interfaccia utente (Cocoa Touch in iOS; Graphical in Android)
- Supporto per SQLite

I linguaggi di programmazione per lo sviluppo di app varia tra sistemi operativi:

- iOS: Objective-C
- Android: C, C++, Java
- BlackBerry OS: Java

Ogni vendor mette inoltre a disposizione una serie di strumenti software (SDK, *Software Development Kit*) che possono includere:

- Gestione delle APIs
- Editor grafici / plug-in per editor di terze parti
- Strumenti per test e debugging
- Emulatori per computer
- Librerie specifiche per dispositivi mobili

## 2.4 - Statistiche del mercato delle App

I canali di distribuzione delle app sono "*digital stores*": piattaforme digitalizzate online per create appositamente per la distribuzione di software per dispositivi mobili.
Gli stores ufficiali sono:

- App Store (Apple, iOS)
- Android Market (Google, Android)
- App World (RIM, BlackBerry)

Qualche dato riguardante i download di applicazioni da questi stores:

***App Store***

Rilasciato il 10 giugno 2008, ha raggiunto il primo miliardo di app scaricate nel giugno del 2009 e nell'ottobre del 2011 ha superato i 18 miliardi di download.[3][4]

***Android Market***

Rilasciato il 22 ottobre 2008, ha raggiunto il primo miliardo di download nell'agosto del 2010 e nel dicembre 2011 ha superato i 10 miliardi di download.[5][6]

Il fatturato di App Store nel 2011 per Apple è stimato intorno ai 3 miliardi di dollari; mentre per Android Marker di 425 milioni di dollari. [7]

Di seguito, in figura 5, vengono mostrati i ricavi relativi al mercato della applicazioni, ed una stima della sua espansione nei prossimi anni. [8][9]

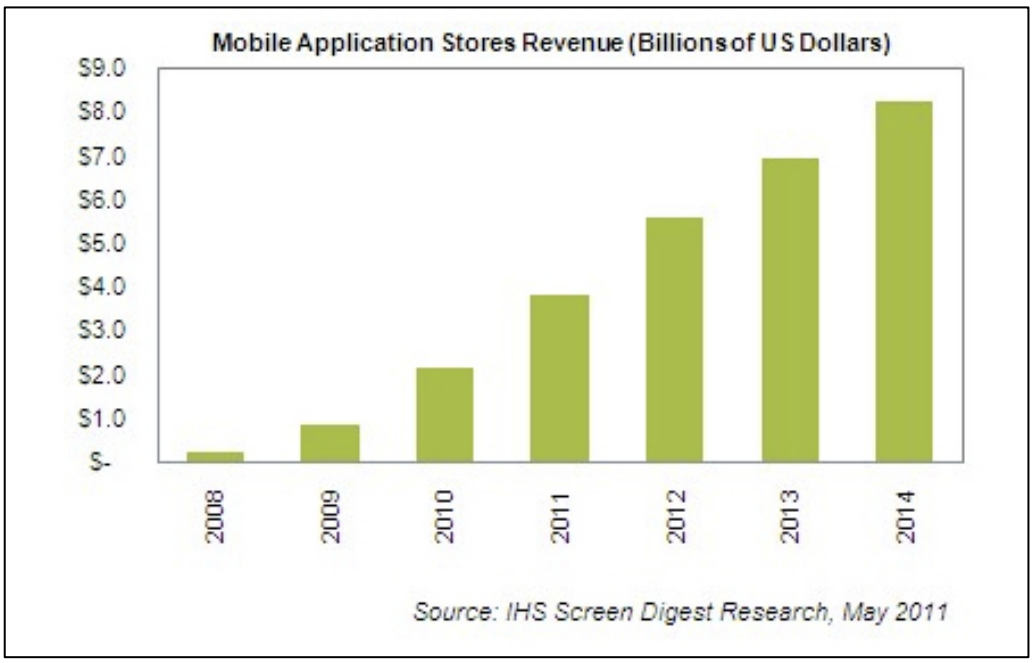



**Figura 5:** Rendimento dei principali applicaitione stores per dispositivi mobili, e stima dell'espansione dei ricavi. [8]

# 3 - Strumenti per lo Sviluppo delle App in iOS.

Per sviluppare applicazioni per dispositivi mobili basati sul sistema operativo iOS è necessario l' iOS SDK (*software development kit*) di Apple. Attualmente alla versione 5.1 e supporta solo Mac intel con sistema operativo Snow Leopard (o superiore); attualmente non esiste una versione ufficiale per altri sistemi operativi come Microsoft Windows o Linux.

## 3.1 - iOS SDK e Cocoa Touch

L'implementazione delle tecnologie impiegate in iOS può essere vista come un insieme di livelli di APIs mostrati in figura 6. Il livello più alto, API Cocoa Touch, è stato sviluppato per offrire agli sviluppatori strumenti di alto livello rendendo il più possibile trasparente il funzionamento dei livelli sottostanti.

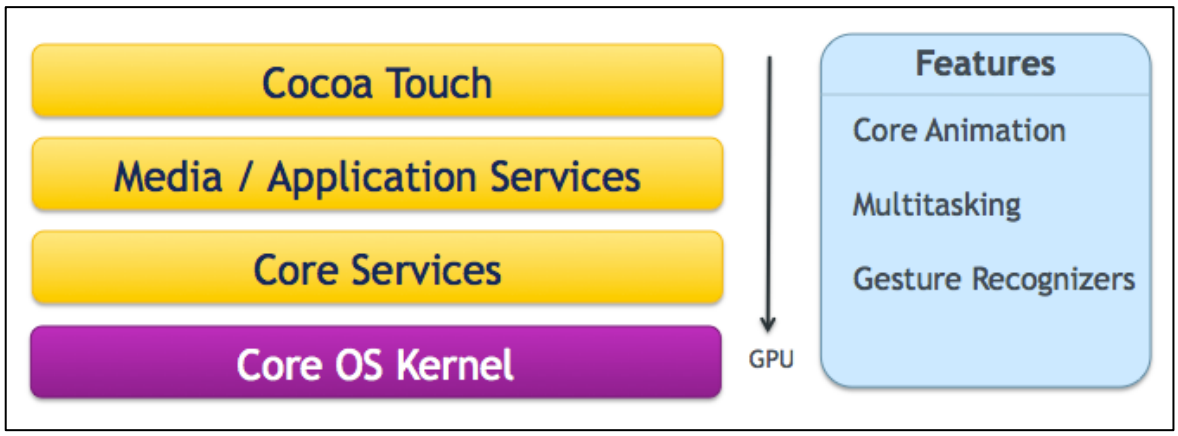



**Figura 6:** Rappresentazione dello stack delle APIs di iOS. Cocoa Touch introduce il livello di astrazione più alto con il quale vengono controllate tutti i livelli sottostanti. [9]

iOS SDK include anche tutta una serie di strumenti e risorse necessarie per lo sviluppo di applicazioni iOS. Apple ha rilasciato un apposito *framework*, una directory che contiene librerie condivise dinamicamente e le risorse necessarie per utilizzarle (come file di *header*, immagini, *helper application*). Nel progetto di sviluppo dell'applicazione è sufficiente fare il *link* a questa libreria per avere accesso a tutte le APIs di iOS.

Nell'SDK di iOS sono presenti anche altri strumenti:

- Xcode Tools:
  - Xcode: un IDE (*Integrated Development Environment*) che permette la gestione di un progetto di un'applicazione con strumenti per l'editing, la compilazione ed il debugging.
  - Altri strumenti: per il debugging, permettono di accedere alle informazioni di *runtime* dell'applicazione.

- iOS Simulator:

  Un'applicazione per Mac OS X che permette di simulare device iOS

- iOS Developer Library:

  Una raccolta di librerie che contengono guide di riferimento per la programmazione in questo ambiante di sviluppo.

## 3.2 - Multiple Phone web-based Framework

Sviluppare per una o per l'altra piattaforma mobile ha i suoi vantaggi e svantaggi sia dal punto di vista dello sviluppo, sia da quello del ritorno economico e della copertura di mercato.
Si prendano ora come riferimento i due principali sistemi: Android e iOS; e gli strumenti, le tecnologie ed i linguaggi di programmazione impiegati nella catena di sviluppo per le loro applicazioni. Quando si sviluppano applicazioni per questi sistemi è necessario pianificare il progetto in funzione del sistema operativo target come prima cosa. Non si può prescindere da questo, per scrivere applicazioni in codice nativo devono essere impiegate APIs differenti ed SDK differenti.
Esistono tuttavia strumenti che possono introdurre un livello ulteriore di astrazione nel paradigma di programmazione di applicazioni per dispositivi mobili: i "*Multiple Phone web-based Framework*" . L'astrazione a questo livello comporta alcuni vantaggi e altri svantaggi che devono essere valutati nelle prime fasi di progettazione delle applicazioni, è fondamentale per il programmatore avere coscienza dei limiti e delle potenzialità di questi strumenti.

***Vantaggi:***

- Permettono di sviluppare applicazioni per dispositivi mobile con linguaggi tipici del web (come JavaScript, PHP, Ruby, HTML e CSS). Sono linguaggi molto più semplici e dinamici rispetto ad objective-C e Java
- Sono *cross-platform*: si può sviluppare un'applicazione per più sistemi operativi contemporaneamente, utilizzando codice comune. Dove possibile viene riutilizzato lo stesso codice per sistemi differenti; dove necessario, ad esempio molte componenti UI (User Interface) sono differenti tra iOS ed Android. vengono riscritte utilizzando però lo stesso linguaggio di programmazione, inserendo degli *controlli condizionali* all'interno dello stesso codice sorgente (che rimane lo stesso per entrambe le piattaforme).

***Svantaggi:***

- Gli strumenti messi a disposizione (editor grafici, debugger, IDE) sono generalmente più limitati rispetto a quelli ideati per lo sviluppo nativo delle applicazioni.
- Le APIs sono più limitate rispetto a quelle degli framework ufficiali: per guadagnare semplicità con l'astrazione ad un livello superiore si ha una parziale perdita di controllo sui livelli inferiori (ad esempio sui controlli UI).
- Se si vuole estendere il framework è necessario utilizzare le APIs di basso livello.
- Prestazioni inferiori rispetto allo sviluppo di applicazioni in codice nativo.

### 3.2.1 - Classificazione

Possiamo suddividerli in due categorie:

***Embedded Dynamic Websites***

Questo tipo di strumenti permette di creare applicazioni il cui risultato finale agli occhi dell'utente è di una classica applicazione per dispositivi mobili; in realtà quello che si ha è però una serie di pagine web dinamiche che possono avere accesso ai controlli UI e ad altre risorse dei dispositivi *embedded*, caricate in memoria persistente nei dispositivi.

Un esempio molto diffuso è PhoneGap, che permette di realizzare applicazioni per varie piattaforme utilizzando HTML5, CSS3 e JavaScript.

Questo tipo di *Multiple phone web-based Applicaiton Framework* permette di creare applicazioni ibride che sono l'evoluzione per dispositivi mobili delle web-application, sono create con le stesse tecnologie, sono a tutti gli effetti delle pagine dinamiche, ma non richiedono un browser web e s'integrano maggiormente con le piattaforme sulle quali sono eseguite, inoltre le performance, così, sono notevolmente migliori.

***Native Language Application Development***

Questi tipi di strumenti mettono a disposizione APIs che svolgono il compito di ponte tra le APIs native dei sistemi operativi e le tecnologie web impiegate (come: JavaScript, php, ruby, HTML e CSS). Si possono così utilizzare vere componenti native UI e non solo pagine generate dinamicamente con HTML e CSS.

Appcelerator Titanium appartiene a questa categoria di framework.

### 3.2.2 – Confronto tra cross-platform frameworks

Esistono numerose piattaforme per lo sviluppo di applicazioni per dispositivi mobili che basate sulle tecnologie web. Dopo una ricerca ed una valutazione globale delle piattaforme più diffuse e complete, basata anche sulle esigenze del progetto, sono state individuati i seguenti frameworks:

- Titanium (Appcelerator)
- Rohodes (Rhomobile)
- MoSync
- PhoneGap

Volendole mettere a confronto per risaltarne i pregi ed i difetti, soprattutto in relazione alle esigenze del progetto, si possono prendere di riferimento i seguenti parametri:

- Componenti UI nativi.
- Accesso alle funzionalità native.
- Compatibilità con le piattaforme mobili (iOS, Android, BlackBerry, etc.).
- Performance generali
- Supporto per lo sviluppo: documentazione, materiale di esempio e *community.*

***Titanium***

| | |
|---|---|
| Compatibilità | Discreta. iOS, Android (versione bera per BlackBerry). |
| Componenti UI native | Ottima. Supporto per la maggior parte delle compontenti UI native. |
| Funzionalità native | Ottima. Asterazione JavaScript per le più comuni funzionalità dei dispositivi, compresa la gestione network di basso livello. |
| Performance generali | Ottime/Buone. Le performance dipendono molto dalle tecniche e strategie di programmazione adottate. Alto grado di libertà da questo punto di vista. |
| Supporto | Ottimo. Ottima documentazione e materiale di esempio; comunità molto attiva, anche per lo sviluppo di moduli aggiuntivi open-source. |

***Rhodes***

| | |
|---|---|
| Compatibilità | Ottima. iOS, Android, BlackBerry, Symbian, Windows Mobile |
| Componenti UI native | Discrete. Componenti *look-like*: non sono vere componenti native, ma danno all'utente la stessa sensazione. |
| Funzionalità native | Ottima. Permettono la gestione con astrazioni Ruby della maggior parte delle funzionalità native. |
| Performance generali | Buone/Discrete. La semplicità in termini di UI non native si paga nelle prestazioni, che risultano inferiori soprattutto in progetti complessi. |
| Supporto | Sufficiente. Documentazione mal organizzata e poco codice di esempio. |

***MoSync***

| | |
|---|---|
| Compatibilità | Buono. iOS, Android, (BlackBerry in parte). |
| Compontnti UI native | Limitate. Solo in parte. |
| Funzionalità native | Buone. Accesso a molte funzionalità. Non supportato l'accesso al calendario nativo. |
| Performance generali | Ottime. |
| Supporto | Ottimo. Ottima documentazione e buona quantità di codici di esempio. |

***PhoneGap***

| | |
|---|---|
| Compatibilità | Ottime. Supporta una vasta gamma di piattaforme mobili tra le quali: iOS, Android, BlackBerry e Windows Mobile. |
| Compontnti UI native | No. |
| Funzionalità native | Ottime. Astrazioni JavaScript per accedere alle più comuni funzionalità dei dispositivi. |
| Performance generali | Ottime. L'esecuzione delle applicazioni risulta fluida ed il *look and feel* per l'utente è paragonabile a quello di un'applicazione nativa. |
| Supporto | Buono. Documentazione esaustiva e comprendente adeguato codice di esempio. |

Con queste piattaforme sono utilizzati gli stessi linguaggi e gli stessi strumenti per sviluppare applicazioni *cross-platform*; ciò non implica che il codice sia "scritto una volta ed eseguito ovunque": come già detto nel capitolo 2.2 i vantaggi sono la semplificazione del codice, la riduzione dei tempi di sviluppo e l'utilizzo degli stessi strumenti e tecnologie per piattaforme diverse; ma molta parte del codice deve essere scritta ad hoc per ogni piattaforma mobile.

Un altro fattore da considerare in fase di scelta è, allora, anche la piattaforma mobile principale per la quale si vuole sviluppare. Nel progetto di questa tesi si è voluto dare la priorità ad iOS. Da questo punto di vista, attualmente, le due scelte migliori sono: PhoneGap e Appcelerator Titanium. Quest'ultimo ha il vantaggio di permettere lo sviluppo di applicazioni native, con il conseguente miglioramento delle prestazioni di esecuzione; che si nota tanto più l'interazione con funzionalità native dei dispositivi è frequente.

Titanium permette anche l'estensione con moduli scritti direttamente in linguaggio nativo e gestisce la generazione di oggetti proxy per la loro integrazione all'interno di progetti. La possibilità di estendere le APIs può risultare vantaggioso a lungo termine.

La scelta è stata per questi motivi Appcelerator Titanium.

### 3.2.3 - Appcelerator Titanum

Appcelerator Titanium è una piattaforma software di sviluppo open source, con la quale è possibile creare applicazioni multipiattaforma per dispositivi mobili utilizzando JavaScript, HTML e CSS. Con Titanium è possibile creare, eseguire e preparare per la distribuzione applicazioni native per dispositivi basati su iOS ed Android (esiste una versione beta per il supporto anche di BlackBerry OS) utilizzando APIs JavaScript.

In figura 7 viene mostrato lo stack della struttura di un'applicazione realizzata con Titanium.

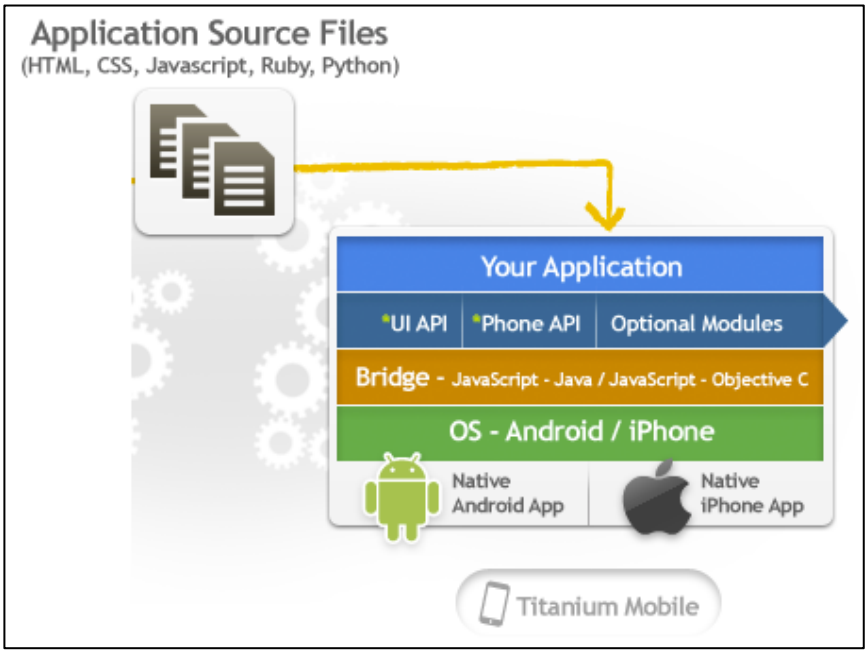



**Figura 7:** Stack delle tecnologie impiegate in Titanium Mobile: l'applicazione viene scritta dallo sviluppatore in JavaScript (possono essere incluse pagine HTML e CSS) utilizzando le APIs messe a disposizione dal framework ed eventuali moduli aggiuntivi. Titanium svolge poi una funzione di bridge Javascipt . Java o JavaScript – Objective-C per generare l'applicazione installabile.

***Codice nativo e codice JavaScript***

Le applicazioni create con Titanium risultano essere un ibrido tra le applicazioni *web-based* e quelle native: sono scritte in JavaScript, ma grazie alle APIs di Titanium si integrano con componenti native dei sistemi operativi. Il codice JavaScript non viene però "tradotto" in codice Objective-C (nel caso di iOS) o Java (nel caso di Android); viene incluso in fase di pre-compilazione in apposite variabili prima della generazione di sorgenti binari installabili sui dispositivi (o emulatori); in questo modo viene poi interpretato a *run-time* quando necessario sfruttando i *JavaScript engine* dei dispositivi mobili: webKit per iOS e V8 per Android.
Mediante le APIs Titanium e moduli aggiuntivi vengono però controllate veri elementi nativi, ai quali corrisponde codice sorgente nativo che deve essere compilato.

***Componenti***

Titanium è composto da:

- Titanium SDK tools
- Titanium mobile APIs
- Titanium Strudio
- Native Modules
- Appcelerator cloud service

**Titanium SDK tools**

Comprende una serie di script scritti in *python* e strumenti di supporto per l'integrazione con gli SDK nativi di iOS ed Android. Questi strumenti permettono di combinare il codice sorgente JavaScript, un interprete JavaScript e tutte le altre risorse incluse nel progetto, generando un'applicazione installabile su un emulatore od un dispositivo. Titanium Studio si occupa dei vari aspetti del processo rendendo il tutto trasparente al programmatore.

**Titanium mobile APIs**

Le Titanium mobile APIs sono APIs JavaScript che permettono l'accesso a centinaia di componenti nativi grafici (*User Interface*) e non.

**Titanium Strudio**

Titanium Studio è un ambiente di sviluppo integrato (IDE) sviluppato a partire dall'IDE Aptana.

**Native Modules**

Sono moduli aggiuntivi che estendono le APIs di Titanium. Possono essere sviluppati anche da terze parti.

**Cloud service**

Appcelerator mette anche a disposizione una serie di servizi analitici per gli sviluppatori che permettono di mantenere statistiche riguardanti le proprie app. Ha anche creato un online-store nel quale è possibile distribuire (gratuitamente o a pagamento) moduli aggiuntivi per Titanium.

# 4 - Sviluppo di un App con Appcelerator Titanium

## 4.1 - Struttura di un progetto in Titanium

In figura 8 è mostrato un tipico esempio di un progetto in Titanium. La cartella *Resources* è quella nella quale vengono salvati tutti i file necessari all'applicazione: i file contenenti codice; le immagini e contenuti multimediali in generale; pagine html o xml; database; librerie; file generati dall'applicazione durante l'esecuzione. Il file *tiapp.xml* è il file xml contenente le direttive per la compilazione del progetto, come ad esempio: la versione di SDK da utilizzare; la piattaforma mobile target (iOS o Android); moduli esterni da includere; direttive per la risoluzione dello schermo o l'orientamento (*landscape o portrait*); etc. Infine il file *app.js* è il file di bootstrap; ogni progetto deve averne uno ed è il primo file JavaScript caricato quando l'app viene lanciata.

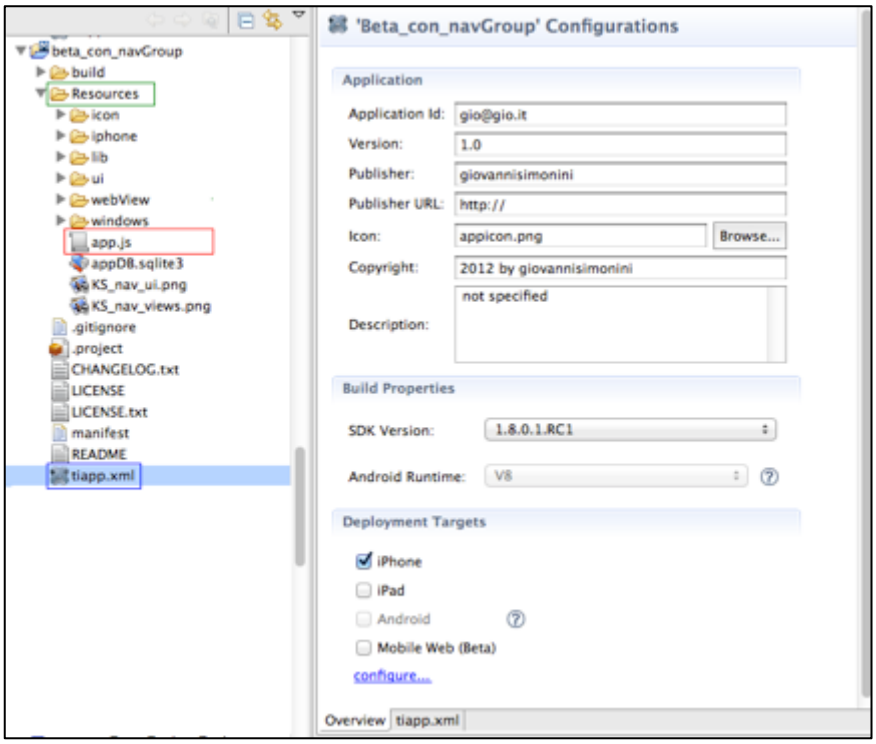



**Figura 8:** Esempio di struttura di un progetto in Titanium Studio. In verde è evidenziata la cartella delle risorse, root delle risorse dell'app; in blu il file di configurazione xml del progetto, il cui contenuto è visibile nella parte destra dell'immagine; in rosso è indicato il file JavaScript *app.js*, all'avvio della compilazione è il primo file JS che viene ispezionato e il primole cui istruzioni vengono interpretate.

## 4.2 - Com'è generata l'applicazione: pre-compiler e front-end-compiler

Creato il progetto, inserite le risorse nell'apposita directory e dichiarato le opportune variabili nel file *xml* di configurazione, la compilazione e la trasformazione da codice in app nativa viene gestita totalmente da Titanium rendendo ogni passaggio totalmente trasparente allo sviluppatore e questo è uno dei grandi vantaggi della piattaforma.

Per approfondire il funzionamento di quello che accade nei "livelli sottostanti", vengono ora illustrati i passaggi che Titanium gestisce automaticamente e che quindi permettono questo livello di astrazione.

Innanzitutto, tra gli strumenti di Titanium sono inclusi: un *pre-compiler* (pre-compilatore), un *front-end-compiler* e un set di script in *python* [9] necessari per l'interazione con gli SDKs nativi delle piattaforme per le quali si sta compilando l'applicazione. Quando l'applicazione è pronta per essere compilata, come prima cosa il codice JavaScript viene elaborato e reso più compatto.

Il *pre-compiler* analizza staticamente il codice e costruisce una gerarchia di dipendenze di tutte le APIs Titanium utilizzate dall'applicazione. Il compito del *front-end-compiler* è di creare successivamente dei blocchi di files "nativi", detti *stubs (mozziconi) files*. Questi *stubs files* includono codice nativo (generato a partire dalle dipendenze ottenute in fase di pre-compilazione), un progetto per l'SDK nativo (in iOS è un *xcode project*), e del codice di controllo necessario per compilare il tutto nel compilatore nativo della piattaforma (iOS o Android)[9]. Infine Titanium invoca il compilatore della piattaforma specifica (per iOS è xcodebuilder) che compila l'applicazione nativa.

Dalla parte del sistema, iOS, il codice JavaScript (*Base64 encoded*, quindi trasformato in codice binario) viene inserito all'interno di variabili in un file C, e compilato anch'esso. Da notare come il codice JavaScript non venga trasformato in codice Objective-C, ma rimanga a tutti gli effetti codice JavaScript; questo codice viene poi interpretato durante l'esecuzione dell'app mediante l'interprete *JavaScriptCore* (interprete JavaScript incluso in iOS). Anche in Android avviene un procedimento analogo; in questo caso l'interprete JavaScript può essere Rhino o V8. In entrambi i casi, il codice originale JavaScript non è più riconducibile ad una forma leggibile (dall'uomo). In figura 9 è illustrato lo schema riassuntivo dei passaggi che Titanium effettua per generare codice binario eseguibile.

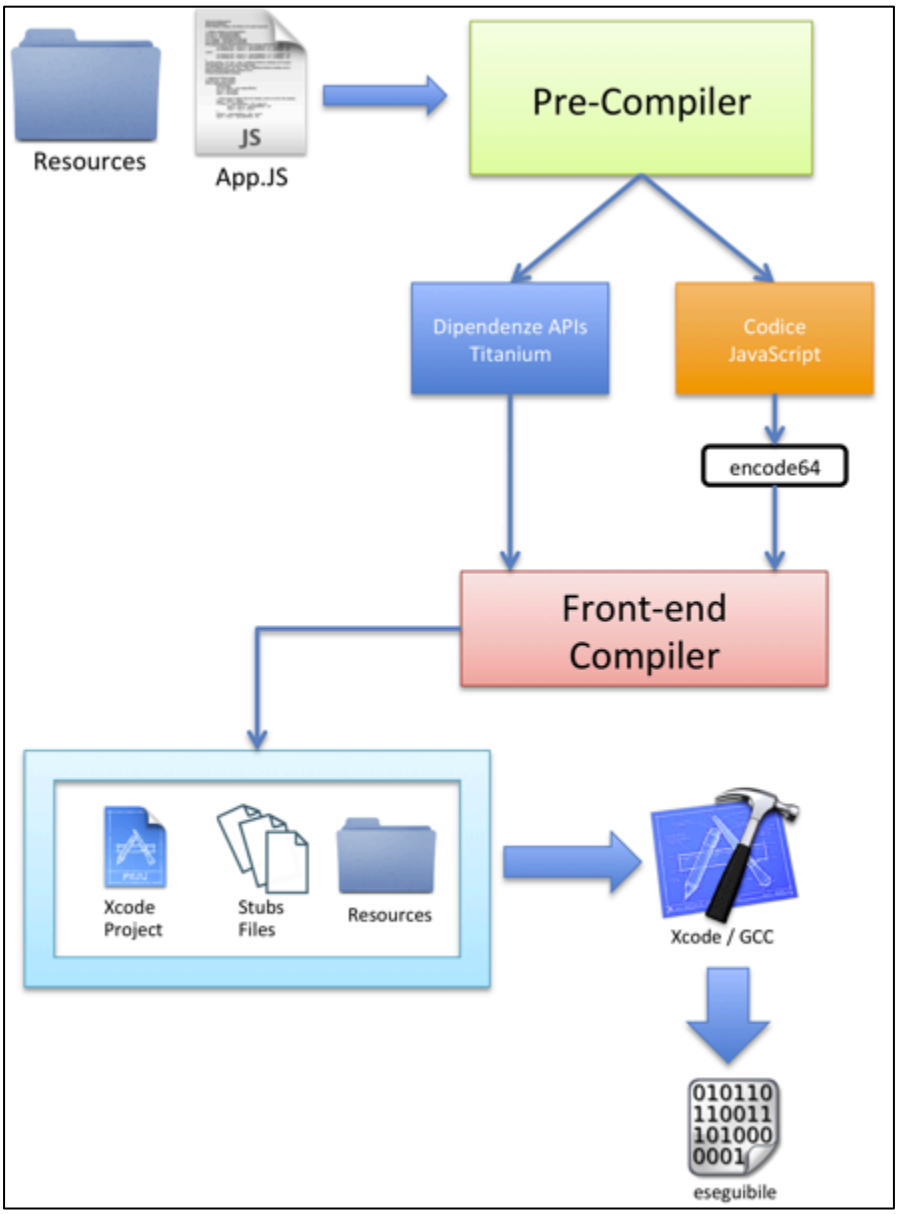



**Figura 9:** Schema delle fasi che vengono eseguite dal framework Titanium durante la compilazione di un progetto per iOS. I fiiles JavaScript contenenti il codice vengono processati dal pre-compiler che genera una gerarchia delle dipendenze relative alle APIs Titanium utilizzate e riordina il codoice JavaScript. Il front-end compiler si occupa poi di generare il progetto XCode. Vengono generati anche gli stubs files, che comprende tra le altre tutte le classi relative agli oggetti specificati nella gerarchia di APIs (componenti UI, database, etc.); comprendono anche altri oggetti che racchiudono al loro interno variabili contenente il codice JavaScript dell'applicazione.

In figura 10 è mostrato invece lo schema dell'esecuzione di un'app sviluppata con Titanium. Si evidenzia come il codice JavaScript che controlla la logica dell'applicazione non venga tradotto in codice nativo, ma venga eseguito all'interno dello JavaScript Engine (webKit in iOS). Tutti gli altri componenti sono invece nativi.

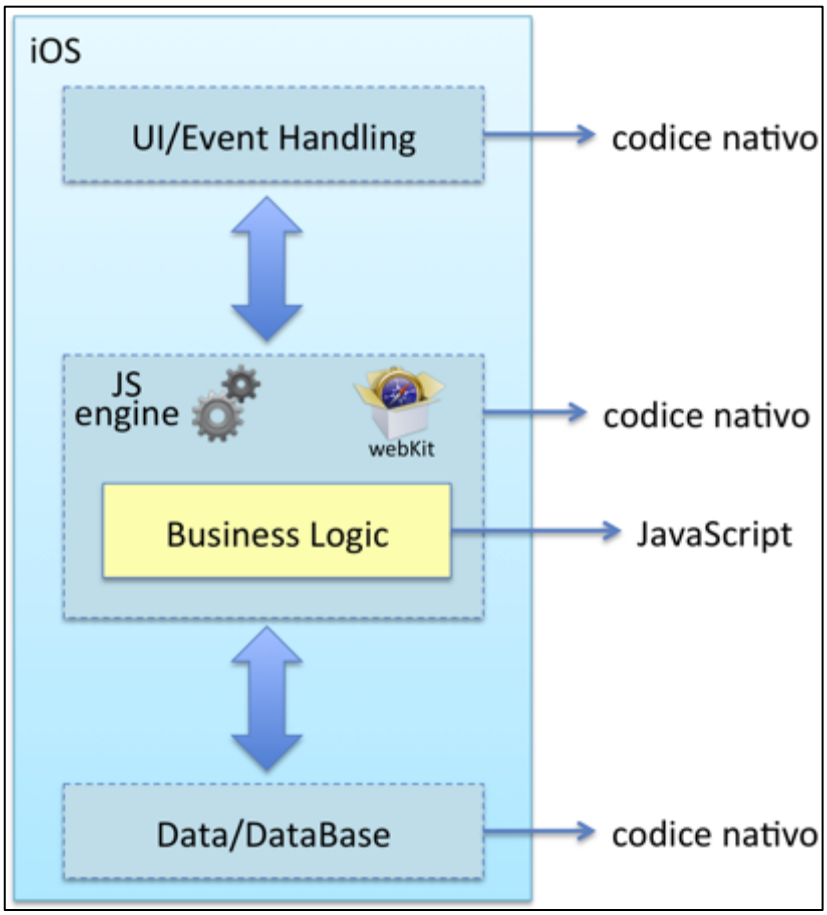



**Figura 10:** Schema dell'esecuzione run-time di un'app sviluppata con Titanium. Tutti i componenti sono nativi; la business logic è implementata in JavaScript ed è eseguita all'interno dello JavaScript Engine che traduce i comandi in comandi "nativi" tramite le APIs Titanium.

## 4.3 - JavaScript in Titanium

In Titaium il linguaggio di programmazione utilizzato per sviluppare applicazioni è JavaScript, linguaggio di scripting orientato agli oggetti. Così, per ogni applicazione sviluppata in Titanium, come per ogni applicazione JavaScript, esiste un oggetto chiamato *global object*. Ogni variabile dichiarata nel *global scope* diventa una proprietà del *global object*. In Titanium il *global scope* coincide con lo spazio di visibilità del file *app.js*; così ogni variabile dichiarata in questo file ha visibilità globale nell'applicazione.

Ogni funzione ha inoltre il proprio contesto di esecuzione di riferimento, se questo contesto è quello di *app.js* allora viene chiamato *global context*.

E' importante in fase di sviluppo di un progetto tenere conto di questi aspetti di tecniche di programmazione, per evitare di generare *global pollution* (troppe variabili ed oggetti mantenuti in memoria) ed interferenze tra variabili.

### 4.3.1 - Tecniche di Programmazione JavaScript per Titanium

Titanium permette di includere in un progetto altre pagine JavaScript oltre ad *app.js* con un proprio *URL* che faccia riferimento ad un file presente nella directory *Resources* (è fortemente sconsigliato far riferimento ad un file da un URL remoto). Il file viene associato ad una *window* ed il suo codice viene eseguito in un *sub-context* separato da quello globale di *app.js*.
I *sub-context* non possono accedere alle variabili globali, sono totalmente isolati, e sono eseguiti in un proprio *thread* distinto da quello principale dell'applicazione. Esistono però meccanismi che permettono il passaggio di dati tra *context*, ma permettono solo di passare dati serializzati (utilizzando lo standard JSON) e non la condivisione di oggetti.

La pratica migliore, in Titanium, di realizzare applicazioni modulari, object-oriented e prevenire *global pollution,* è utilizzare le *closure* e i moduli *CommonJS*.

#### *Closures in JavaScript*

Una funzione *closure* (chiusura) è una funzione *autochiamante*, e tutto ciò che viene dichiarato in essa ha visibilità solamente al suo interno. Di seguito un esempio di *closure* in JavaScript:

```
/*--------inizio closure----------------------------*/
(function() {
  var varPrivata = 'Qua è visibile';
})();
/*--------fine closure-----------------------------*/
alert(varPrivata); //non visibile nel global scope
```

La variabile *varPrivata* viene creata solo nel momento in cui viene eseguito il codice relativo alla *closure* ; e quando la funzione è terminata la memoria può essere liberata. Vengono solo mantenute quelle variabili (o oggetti) che sono stati memorizzati all'interno di variabili del *global scope* e le risorse a loro necessarie. Un esempio di quanto appena descritto:

```
var Counter = (function() {
  var privateCounter = 0;
  function changeBy(val) {
    privateCounter += val;
  }
  return {
    increment: function() {
      changeBy(1);
    },
    decrement: function() {
      changeBy(-1);
    },
    value: function() {
      return privateCounter;
    }
  }
})();

alert(Counter.value()); /* Alerts 0 */
Counter.privateCounter // Non è visibile invece qua
```

In questo esempio si può osservare come il contatore (variabile *privateCounter*) sia dichiarato all'interno della *closure*, non sia visibile dall'esterno, ma non venga eliminato dalla memoria in quanto necessario all'oggetto *Counter*.

### Vantaggi dell'uso delle *closure* in Titanium

Questa tecnica di programmazione può essere molto utile per lo sviluppo di applicazioni per dispositivi mobili in Titanium, poiché permette una gestione migliore delle risorse (soprattutto memoria RAM) allocate durante l'esecuzione.

JavaScript è un linguaggio snello e versatile, semplice e potente al tempo stesso; si addice perfettamente alle esigenze di Titanium (semplificare la programmazione di app per dispositivi mobili), ma nasce per il web ed i browser. Un'applicazione, o anche solo una *window*, in Titanium può diventare molto onerosa dal punto di vista della memoria necessaria all'esecuzione, e la memoria nei dispositivi mobili è limitata, non paragonabile a quella dei personal computer. Affollare quindi il *global scope* con variabili potrebbe essere inevitabile, con la conseguente degenerazione dell'allocazione di memoria. Utilizzando invece tecniche come quella delle *closure* tutto questo può essere evitato, limitando all'essenziale l'occupazione di memoria.

***Moduli CommonJS***

La modularità del codice JavaScript può essere rafforzata dividendo la logica dell'applicazione in più file JavaScript e oggetti. Con la definizione di moduli *CommonJS* è possibile definire moduli JavaScript con un ben definita interfaccia pubblica.
Nella pratica un modulo CommonJS viene definito in un proprio file, il cui codice viene eseguito quando richiesto secondo il modello *sandbox*, separato dal contesto globale; e permette l'esportazione di oggetti secondo le direttive espresse nel file stesso.

***Programmazione Object-oriented in Titanium***

JavaScript è un linguaggio *prototipale*, non prevede la definizione di classi ed eredità come altri linguaggi (come Java), ma solo oggetti istanziabili. Questi oggetti possono avere metodi ed attributi, possono essere estesi e clonati. Gli oggetti creati mediante le APIs Tianium non sono però veri e propri oggetti JavaScript.. In particolare, non possono essere estesi con attributi e metodi. Sono degli oggetti *proxy*: possiedono un'interfaccia che permette di controllare oggetti nativi, svolgendo una funzione di proxy.
Un ragionevole approccio per creare componenti personalizzate è quello di associare allora un oggetto proxy ad un normale oggetto JavaScript. Utilizzando opportune tecniche di programmazione è possibile simulare anche l'esistenza di classi ed ereditarietà[10]. Un metodo è proposto in APPENDICE A.

### 4.3.2 - Librerie aggiuntive JavaScript per Titanium

Di seguito sono illustrate le caratteristiche di due librerie di funzioni JavaScript che sono state impiegate nel progetto: DateJS e jQuery.

***DateJS***

DateJS è una libreria JavaScript open-source per la gestione delle date. L'integrazione in Titanium mette a disposizione nuovi metodi per eseguire operazioni per l'elaborazione di oggetti *Date*. Offre al programmatore strumenti utili per la formattazione, per il parsing e l'elaborazione di questo tipo di dati.

Permette al programmatore di utilizzare funzioni facilmente concatenabili, scritte in modo da permettere una programmazione che si avvicina al linguaggio naturale ( 'zucchero sintattico'): sono costrutti sintattici che non aggiungono niente alle funzionalità, ma ne rendono più facile l'utilizzo per i programmatori. Mette a disposizione anche metodi di conversione tra standard di date; ad esempio permette di convertire da ISO-8601 (formato utilizzato da SQLite) a RFC 3339 (formato utilizzato da JavaScript).

Alcuni esempi dell'utilizzo di DateJS sono i seguenti:

```
// Che giorno è il prossimo giovedì
Date.today().next().thursday();

// Aggiungi 3 giorni da oggi
Date.today().add(3).days();

// Che data era 3 giorni fa
(3).days().ago();

// Conversione stringhe in date
Date.parse('today');
Date.parse('t + 5 d'); // oggi più 5 giorni
Date.parse('next thursday');
Date.parse('February 20th 1973');
Date.parse('Thu, 1 July 2004 22:30:00');
```

***jQuery***

L'obiettivo di questa libreria open-source è di semplificare la programmazione lato client di pagine HTML. Fornisce metodi e funzioni per gestire gli aspetti grafici e strutturali di una pagina HTML manipolando ad esempio il *Document Object Model*, controllando gli stili degli elementi (CSS), semplificando la gestione degli eventi.

Titanium permette la gestione di *web view* integrate all'interno dell'applicazione; con la possibilità di generarle dinamicamente, e per questo può essere impiegata jQuery.

## 4.4 - Storing Data in Titanium

Titanium fornisce diverse APIs per salvare dati nel dispositivo mobile. Queste APIs permettono di accedere a funzionalità native e sono suddivisi in 3 categorie: *proprietà, database* e *filesystem*. Per ognuna di queste categorie Titanium prevede uno specifico oggetto per accedervi:

- *Titanium.App.Properties* per il salvataggio di *setting* dell'applicazione
- *Titanium.Database* che permette di accedere e gestire database SQLite3
- *Titanium.Filesystem* che fornisce i metodi per la manipolazione di file e directory.

### 4.4.1 - App Properties

L' API *Titanium.App.Properties* permette di memorizzare semplici associazioni *key-value*. E' possibile memorizzare stringhe , interi, boolean, array e in generale tutti i dati che possono essere serializzati con JSON [11]. E' un ottimo strumento per memorizzare pochi dati di dimensioni limitate, molto veloce ed efficiente. Se però è necessario memorizzare dati con relazioni o di grandi dimensioni è meglio utilizzare database e filesystem.

In figura 11 è mostrato un confronto tra le prestazioni dei due metodi. Il graficco mette in relazione i tempi impiegati con le due differenti strategie di memorizzazione per effettuare un loop di operazioni. Le operazioni compiute sono una memorizzazione di un dato, il recupero da memoria del dato stesso ed un'espressione condizionale.

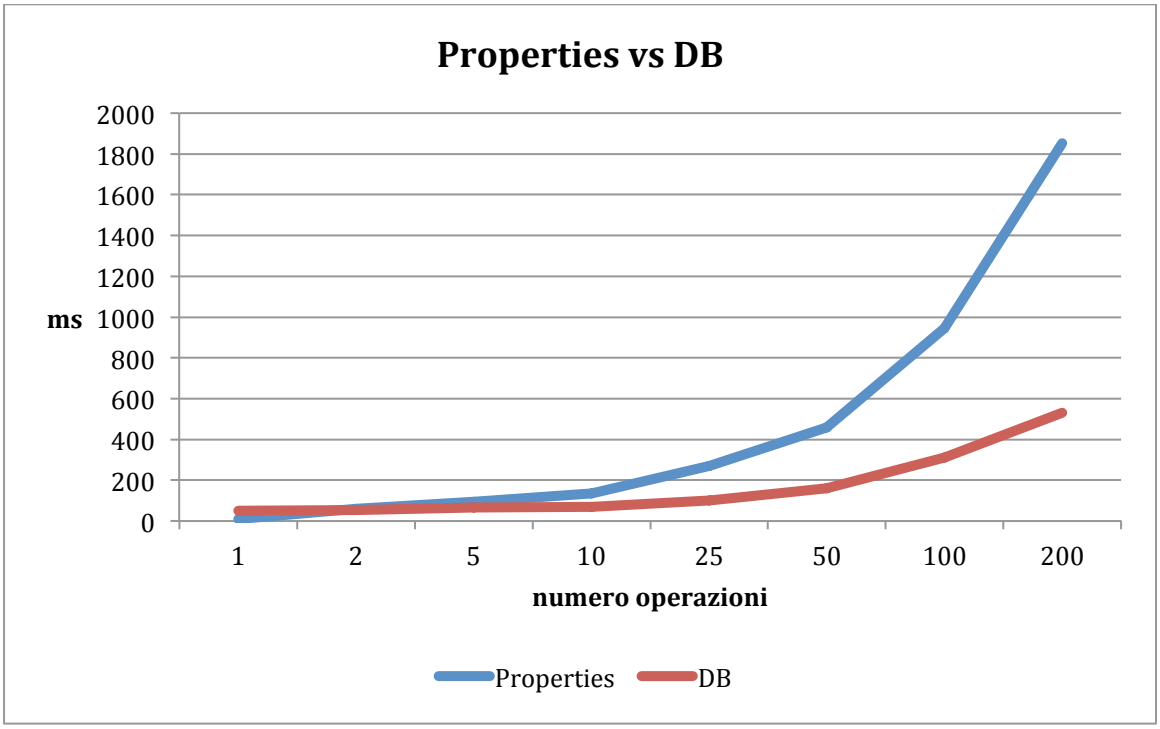



**Figura 11:** Confronto tra le prestazioni dei due diversi metodi di memorizzazione: Properties e database. Il primo caso risulta conveniente per l'inserimento o recupero di un solo dato alla volta; mentre il secondo per quantità di dati maggiori. Oltre alla memorizzazione viene considerato anche un'espressione condizionale: utilizzando la logica relazionale del database e le funzionalità del RDBMS queste vengono risolte più velocemente. I dati fanno riferimento al simulatore di XCode, per i dispositivi mobili gli ordini di grandezza dei tempi possono essere anche più grandi di un fattore 10.

Le prestazioni migliori con l'utilizzo delle properties si hanno quando deve essere memorizzato un solo dato alla volta; nel caso dell'accesso al database è sempre presente un tempo di accesso iniziale ad esso costante, uguale sia per un inserimento (o recupero) singolo che per un loop.

Questo metodo di memorizzazione è impiegato quando i dati da memorizzare sono in numero limitato, non cambiano di frequente, sono di dimensioni ridotte e non richiedono una logica relazionale per essere gestiti; un esempio di utilizzo è la memorizzazione di proprietà relative all'applicazione che vengono ricaricate ad ogni suo avvio, come le impostazioni che l'utente modifica durante l'utilizzo.

#### 4.4.2 - SQLite in Titanium

SQLite è un *Relational Database Management System*(RDBMS) basato su SQL, supportato da Apple, Android e RIM e utilizzato per la memorizzazione dei dati locali nei loro dispositivi mobili.

In un progetto Titanium i dati di un database SQLite vengono memorizzati tutti in un unico file di testo nella directory *Resources*.
Titanium mette a disposizione un apposito modulo per la gestione di database; invocando questo modulo viene restituito un oggetto *Titanium.Database.DB* che possiede tutti i metodi necessari a: installazione, apertura di un database; esecuzione di query; gestione dei risultati di una query.
Viene impiegato questo metodo quando ci sono molti dati e sono in relazione tra di loro, quando vi è necessità di una logica relazionale per inserirli, ordinarli, ritrovarli o elaborarli.

#### 4.4.3 – Filesystem

Con le APIs di Titanium è possibile anche accedere in lettura e scrittura al filesystem dei dispositivi. L'accesso è limitato alle risorse dell'applicazione stessa, non è possibile accedere a quelli di altre applicazioni o del sistema operativo. Questa non è una limitazione di Titanium, ma è un'imposizione di Apple. In Android invece è possibile accedere anche alle risorse del sistema.
Si utilizza questo metodo di memorizzazione soprattutto per file di grandi dimensioni (come contenuti multimediali: immagini, file audio etc.) e soprattutto per dati che possiedono già un proprio formato (una propria estensione).

# PARTE II

# 5 – Progettazione App per Gestione e Pianificazione di Attività CalPlanner

In questo capitolo sono descritti formalmente le funzionalità ed i requisiti dell'applicazione definiti in fase di progettazione. In seguito sono illustrati i moduli funzionali e la loro progettazione.

## 5.1 – Definizioni

Di seguito sono elencate alcune definizioni e convenzioni semantiche di termini utilizzate nei capitoli successivi per la descrizione delle funzionalità dell'applicazione.

***Evento Singolo***

Appuntamento singolo. Possiede: un nome di tipo; una data iniziale ed una finale; un orario di inizio ed uno di fine; altre informazioni facoltative. Ogni *Evento Singolo* è un'istanza di un *Tipo Evento*.

Sinonimi: incontro, appuntamento, impegno.

***Tipo Evento***

Ogni *Evento Singolo* deve appartenere ad un *Tipo Evento*. E' una classe generale di eventi definita dall'utente. *Tipo evento* può possedere dei vincoli definiti dall'utente come: ore minime, ore massime e data di scadenza.

Sinonimi: attività, classe di eventi.

***Proiezione Eventi***

Visualizzazione di eventi appartenenti ad un determinato *Tipo Evento* in un arco temporale. La vista è costituita da un array di tanti elementi quanti sono i giorni dell'arco temporale, ed in ogni elemento è memorizzato il numero di ore di *Tipo Evento* presenti nel database quel giorno.

## 5.2 – Requisiti Funzionali

Di seguito sono descritte le principali funzioni che mette a disposizione l'App. Ogni requisito avrà una struttura formata da 5 voci:

1. *Componenti coinvolte: componenti software o interazione dell'utente.*
2. *Input: dati necessari in ingresso alla funzione*
3. *Processo: azioni svolte dalle componenti*
4. *Errori: gestione e segnalazione di errori*
5. *Output: risultato della funzione*

Le componenti coinvolte sono:

- **Utente**: L'utilizzatore dell'app
- **Interfaccia Utente**: Interfaccia grafica per interazione dispositivo-utente
- **Database**: Database dell'app
- **iCal**: Calendario nativo del dispositivo

### *RF 1 – Inserimento Tipo Evento*

L'utente crea un nuovo tipo di evento da utilizzare nell'inserimento rapido di eventi. Rappresenta l'astrazione dei singoli eventi inseriti nel calendario.

| | |
|---|---|
| Componenti coinvolte | Utente<br>Interfaccia Utente<br>Database |
| Input | L'utente deve inserire un *nome* identificativo ed univoco per il tipo di evento, più altri campi facoltativi: luogo, ore totali, ore minime, ore massime, data inizio, data fine, descrizione generale. |
| Processo | Viene controllato che nel database non sia presente un *Tipo Evento* con lo stesso identificativo, in caso di successo viene creata una nuova classe *Tipo Evento*. Viene poi chiesta conferma all'utente generando un messaggio con un riepilogo dei dati. |

| | |
|---|---|
| Errori | Il nome per il *Tipo Evento* può essere già stato assegnato, viene generato un errore e l'operazione viene annullata. Un messaggio di viene mostrato all'utente e l'esecuzione può riprendere dall'inserimento di *Tipo Evento*.<br>In caso di presenza di vincoli si controlla la loro coerenza, in caso di fallimento viene generato un errore, l'operazione viene annullata e comunicata all'utente il motivo di tale fallimento. |
| Output | Messaggio di conferma con riepilogo delle caratteristiche del tipo di evento. |

***RF 2 – Inserimento di un Evento Singolo***

L'utente crea una nuova istanza di *Tipo* Evento da inserire nel calendario al termine di un'attività appena compiuta.

| | |
|---|---|
| Componenti coinvolte | Utente<br>Interfaccia Utente<br>Database |
| Input | Selezione del *Tipo Evento*. Selezione dell'ora di inizio o fine dell'evento singolo (impostata di default sull'ora corrente) e delle ore svolte. E' possibile eventualmente modificare l'orario di inizio dell'evento. Inserimento facoltativo di caratteristiche e di una descrizione. |
| Processo | Viene verificata la disponibilità effettiva delle ore selezionate nel database. Viene inserito un nuovo evento nel database con riferimento al *Tipo Evento* a cui è associato. |
| Errori | Se le ore selezionate sono occupate da un altro evento, viene generato un errore e suggerito uno slot temporale libero. |
| Output | Messaggio di conferma dell'avvenuto inserimento; riepilogo delle statistiche del *Tipo Evento* relativo. |

***RF 3 – Vista di Tipi Evento***

L'utente vuole visualizzare in una vista riassuntiva tutte le ore svolte in un determinato arco temporale. La vista comprende una tabella compatta dove le righe rappresentano i tipi di eventi, mentre le colonne i singoli giorni del periodo indicato. Ogni cella contiene il numero di ore di un determinato tipo di evento svolte in un determinato giorno.

| | |
|---|---|
| Componenti coinvolte | Utente<br>Interfaccia Utente<br>Database |
| Input | Utente seleziona i tipi di evento da visualizzare, data di inizio e di fine. |
| Processo | Per ogni tipo di evento sono recuperate tutte le istanze (gli eventi) dal database relative all'arco temporale indicato. È generato un array di n elementi; n è il numero di giorni del periodo temporale. In ogni elemento è memorizzato il numero di ore svolte quel determinato giorno di quel *Tipo Evento*.<br>Questi array sono processati e viene generata la vista totale. |
| Errori | Fallimento database. Segnalazione all'utente, stop dell'applicazione. |
| Output | Vista totale degli eventi per ogni *Tipo Evento*. |

***RF 4 – Proiezione di un Tipo Evento***

L'utente vuole visualizzare una possibile distribuzione di ore per una determinata attività con vincoli da lui specificati. Gli eventi non vengono memorizzati, solo mostrati in una "vista totale". A partire da questa vista totale modificabile si può successivamente inserire gli eventi nel calendario.

| | |
|---|---|
| Componenti coinvolte | Utente<br>Interfaccia Utente<br>Database |

| | |
|---|---|
| Input | Selezione o creazione di *Tipo Evento*. Numero ore da inserire, data inizio, data scadenza finale, giorni preferenziali per l'evento, ore minime/massime per settimana, fasce orarie. |
| Processo | Per ogni settimana vengono ricercate ore disponibili per quel dato evento compatibilmente con i vincoli degli altri eventi e dei vincoli indicati dall'utente. Se in una data settimana non si riesce a rispettare il vincolo si cerca di distribuire le ore in altre settimane rispettando i vincoli. La proiezione viene memorizzata solo se richiesto successivamente dall'utente. |
| Errori | Numero di ore disponibili non trovato per i vincoli definiti. Generazione di un messaggio di errore con informazioni sul fallimento dell'operazione. |
| Output | Vista totale con una proiezione delle possibili ore allocabili. |

***RF 5 – Modifica Tipo Evento***

L'utente modifica un *Tipo Evento*, o lo elimina. Le modifiche devono essere apportate a cascata anche sugli eventi che si riferiscono ad esso.

| | |
|---|---|
| Componenti coinvolte | Utente<br>Interfaccia Utente<br>Database |
| Input | Modifica campi relativi al tipo di evento |
| Processo | Le modifiche vengono valutate, se non viene violato nessun vincolo e non ci sono incompatibilità con altri eventi vengono memorizzate. |
| Errori | I nuovi vincoli interferiscono con altri eventi o non possono essere soddisfatti. Le modifiche vengono annullate e viene generato un messaggio di errore con riferimento ai vincoli disattesi. |
| Output | Messaggio di conferma per l'utente |

***RF 6 – Ricerca Eventi da iCal***

Ricerca dinamica di eventi da importare da iCal e sincronizzare con il calendario dell'app. Gli eventi vengono ricercati all'interno di un arco temporale indicato dall'utente. Gli eventi selezionati vengono fatti confluire in un tipo di evento scelto dall'utente.

| | |
|---|---|
| Componenti coinvolte | Utente<br>Database<br>iCal |
| Input | Data inizio, data fine, nomi di eventi di iCal, nome del tipo di evento |
| Processo | Sono ricercati gli eventi indicati dall'utente in iCal, vengono creati gli eventi corrispondenti sul calendario dell'applicazione ed associati ad un *Tipo Evento* specificato dall'utente. |
| Errori | Sovrapposizione con eventi già esistenti. Generazione di un messaggio di errore. |
| Output | Messaggio di conferma per l'utente |

***RF 7 – Visualizza Statistiche***

Vengono ricercati i dati necessari alla creazione di una vista totale degli eventi presenti in memoria.

| | |
|---|---|
| Componenti coinvolte | Interfaccia Utente<br>Database |
| Input | Data inizio, data fine, tipi di evento. |
| Processo | Vengono ricercati nel database tutti gli eventi dei tipi specificati e passati al modulo dell'interfaccia grafica che elabora una vista totale riassuntiva con indicazioni sullo stato di completamento per ogni *Tipo Evento*. |
| Errori | Fallimento del database. Segnalazione errore. |
| Output | Vista statistiche e stato dei *Tipi Evento*. |

## 5.3 – Progettazione Database

Di seguito (figura 12) viene mostrato lo schema E-R (Entity-Relationship) del database impiegato nella realizzazione del progetto.

Sono inclusi anche alcuni elementi che per ora non vengono utilizzati dalle funzionalità dell'applicazione come le *liste*; sono implementate a livello di database e librerie per possibili sviluppi futuri.

### 5.3.1 – Modello E-R

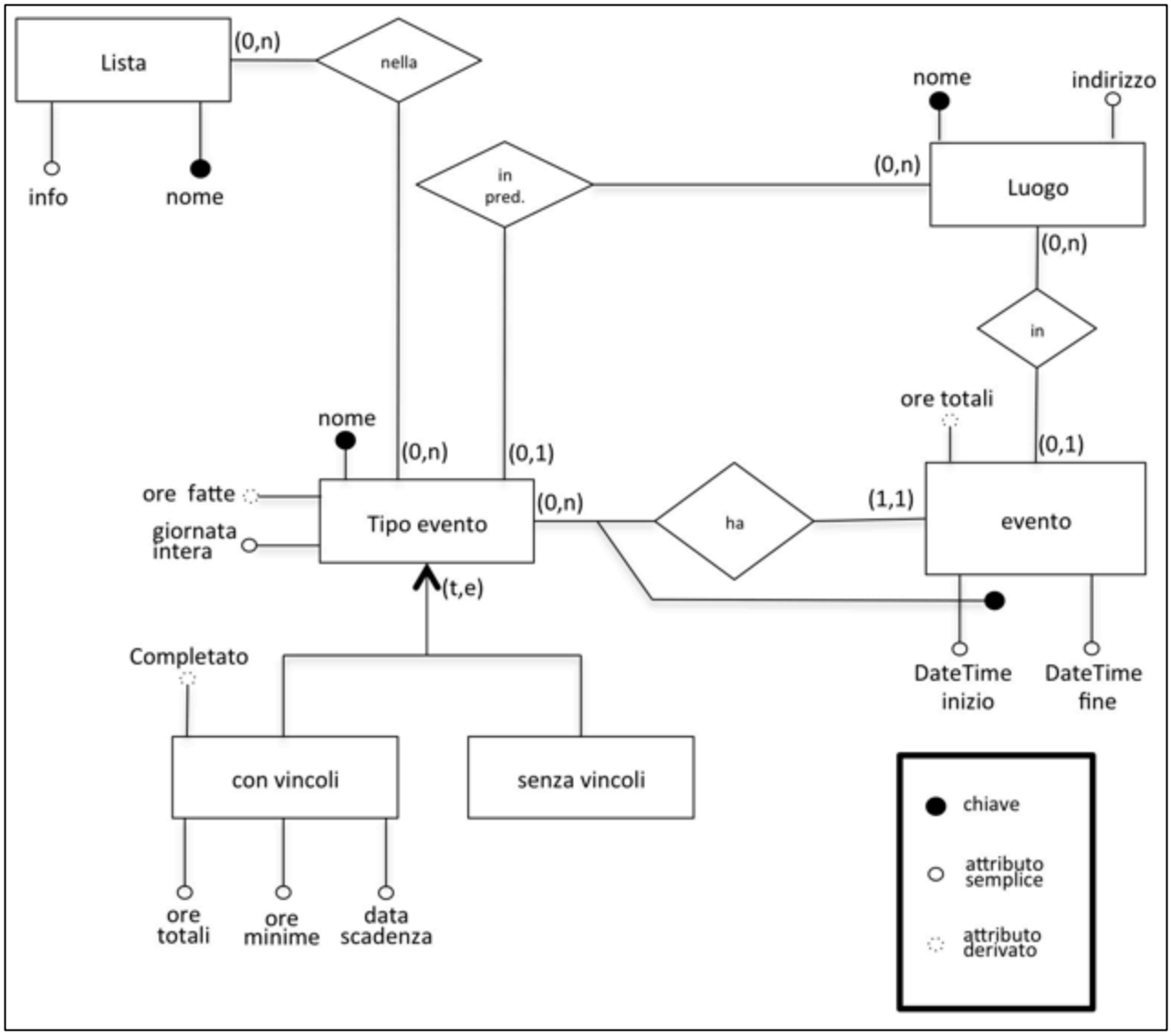



**Figura 12:** Modello Entità-Relazione del database dell'app.

***Dizionario Entità***

In questa tabella sono elencate le definizioni, gli attributi e gli identificatori di tutte le entità che compaiono nello schema E-R.

| **Entità** | **Descrizione** | **Attributi** | **Identificatore** |
|---|---|---|---|
| Tipo Evento | Evento generico, attività | Nome, ore_fatte, giornata_intera | Nome |
| Con Vincolo | Tipo di evento con dei vincoli | Eredita gli attributi da "*Tipo Evento*", più: ore_totali, ore_minime, data_scadenza | Nome |
| Senza Vincolo | Tipo di evento senza vincoli | Eredita gli attributi da "*Tipo Evento*" | Nome |
| Evento | Singolo evento | DateTime_inizio, DateTime_fine, ore, Nome_tipo | Nome_tipo, DateTime_inizio |
| Luogo | Lugo dove si svolge un evento | Nome, indirizzo | Nome |
| Lista | Lista di tipi di evento | Nome, info | Nome |

***Dizionario Relazioni***

In questa tabella sono elencate le associazioni presenti nel modello E-R e le loro descrizioni.

| **Relazione** | **Descrizione** | **Entità Coinvolte** | **Attributi** |
|---|---|---|---|
| Ha | Associa un evento al proprio tipo | Tipo Evento (0,n)<br>Evento (1,1) | - |
| In | Associa un evento al luogo in cui si svolge | Evento (0,1)<br>Luogo (0,n) | - |
| In predefinito | Associa un tipo di evento al proprio luogo predefinito | Tipo Evento (0,1)<br>Luogo (0,n) | - |
| Nella | Associa un tipo di evento alle liste a cui appartiene | Tipo Evento (1,n)<br>Lista (0,n) | (Valore_nella_lista) |

***Regole imposte dall'App***

| Regola | Descrizione |
|---|---|
| I | Per ogni tipo di evento le ore minime devono essere minori o uguali alle ore totali. |
| II | Un evento deve avere un'ora di inizio minore di quella di fine. |
| III | Un evento non può avere data di inizio superiore a quella di fine. |
| IV | Se non è specificato nell'inserimento dell'evento a quell'evento viene assegnato come luogo il luogo predefinito del suo tipo di evento di appartenenza. |

***Regole di derivazione***

| Regola | Descrizione |
|---|---|
| i | "ore_fatte" è calcolato come somma delle ore di eventi di quel determinato tipo. |
| ii | "ore" deve essere la differenza tra ora fine ed ora inizio. |

**5.3.2 – Studio dei Dati Derivati**

Considerando solamente la parte dello schema E-R che rappresenta le entità e le relazioni coinvolte nello studio del dato derivato "ore_fatte", si ottiene lo schema in figura 13.

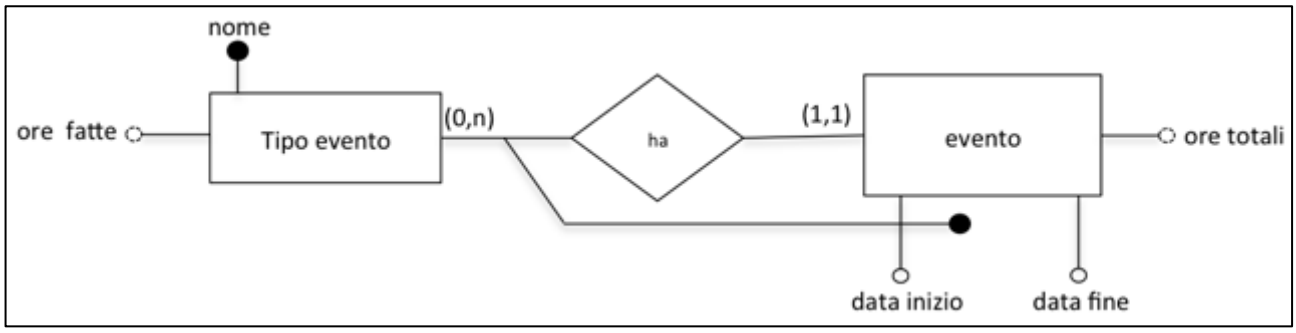



**Figura 13:** Schema E-R per lo studio del dato derivato "ore_fatte".

Le operazioni e i volumi di dati sono i seguenti:

**Operazione 1:** Dato il *nome* di un *Tipo Evento* visualizzare tutti i suoi dati. (O1)

**Operazione 2:** Inserimento di un nuovo *evento*. (O2)

Tavola dei volumi

| Concetto | Tipo | Volume |
|---|---|---|
| Tipo Eventi | E | $x$ |
| Evento | E | $y$ |

Tavola delle operazioni

| Operazione | Tipo | Frequenza |
|---|---|---|
| O1 | I | $f$ |
| O2 | I | $g$ |

- *x,y* sono i volumi delle tabelle *Tipo Evento* ed *Evento.*
- *f,g* sono le frequenze giornaliere espresse in: [ $\frac{num.\ esecuzioni}{giorno}$ ]

***Senza Dato Derivato***

| | Concetto | Accesso | Tipo |
|---|---|---|---|
| **Operazione 1**<br>Accessi Lettura: $\left[\left(2\ \frac{y}{x}\right)+1\right]$<br>Accessi Scrittura: 0<br>Frequenza: $\left[\left(2\ \frac{y}{x}\right)+1\right]f$ | Tipo Evento<br>Ha<br>Evento | 1<br>$y/x$<br>$y/x$ | L<br>L<br>L |
| **Operazione 2**<br>Accessi Lettura: 0<br>Accessi Scrittura: 2<br>Frequenza: $4g$ | Evento<br>HA | 1<br>1 | S<br>S |

**Totale:** $\left[\left(2\ \frac{y}{x}\right)+1\right]f+4g$

***Con Dato Derivato***

| | Concetto | Accesso | Tipo |
|---|---|---|---|
| **Operazione 1**<br>Accessi Lettura: 1<br>Accessi Scrittura: 0<br>Frequenza: $(1 * f)$ | Tipo Evento | 1 | L |
| **Operazione 2**<br>Accessi Lettura: 1<br>Accessi Scrittura: 3<br>Frequenza: $(7 * g)$ | Evento<br>HA<br>Tipo Evento<br>Tipo Evento | 1<br>1<br>1<br>1 | S<br>S<br>L<br>S |

**Totale:** $(f + 7g)$

Mettendo a confronto le due soluzioni, "senza dato derivato" e " con dato derivato" si ottiene che si ha convenienza nel mantenimento del dato derivato quando:

$$3g \leq 2 \frac{y}{x} f \qquad (1)$$

Con ipotesi, valutando un caso minimo di utilizzo dell'app:

- Volume di *Tipo Evento x* = 10 (10 tipi di eventi memorizzati nel database)
- Volume di *Evento* y = 40 (40 eventi memorizzati nel database, sottostimato)
- Funzione di visualizzazione delle statistiche (O1) di un solo tipo di evento al giorno: $f = 5$ (frequenza di utilizzo molto bassa, sottostimata: la visualizzazione della schermata totale riassuntiva comporterebbe una $f = 10$, con *x = 10*)
- Frequenza di inserimento (O2) di eventi: $g = 10$ ( inserimento di 10 eventi al giorno; frequenza di utilizzo alta, sovrastimata )

Determinerebbe una soluzione per l'equazione (1):

$$3 * 10 \leq 2 \frac{40}{10} * 5 \implies 30 \leq 40$$

Risultera conveniente mantenere il dato derivato "ore_fatte".

## 5.4 – Struttura Modulare dell'Applicazione

Per lo sviluppo dell'applicazione si è scelto di utilizzare la piattaforma Titanium (Appcelerator); per questo motivo la progettazione dell'app CalPlanner non poteva prescindere dalle funzionalità e caratteristiche messe a disposizione da questo framework.
Per sviluppare applicazioni in Titanum è bene conoscere e comprendere il funzionamento delle APIs messe a disposizione per poter effettuare scelte progettuali adeguate. Per questo, si è deciso di strutturare il progetto dell'app come mostrato in figura 14. Come mostrato, l'intero progetto è basato su moduli funzionali, in particolare si possono distinguere 3 macroblocchi logici:

- **Logica della gestione dell'interfaccia utente**:
  *User Interface Logic + Native User Interface + API Ti.UI*

- **Logica funzionale**:
  *Business Logic JavaScript + API Ti.App*

- **Logica di memorizzazione**:
  *Librerie JavaScript + Modulo Calendario + DataBase + API Ti.Database*

Le APIs Titanium si inseriscono trasversalmente in questo schema in quanto gestiscono ad ogni livello la traduzione dei comandi JavaScript in linguaggio nativo per i dispositivi mobili.

Si è così impostato il progetto su due fasi di sviluppo principali:

I. Sviluppo di librerie JavaScript per la gestione di dati persistenti. Includono anche la libreria open-source DateJS e mettono a disposizione funzioni per la comunicazione con il calendario nativo, con il database e sincronizzazione tra i due.
II. Sviluppo della logica dell'applicazione e gestione dell'interfaccia utente mediante JavaScript ed APIs specifiche sulla piattaforma Titanium.

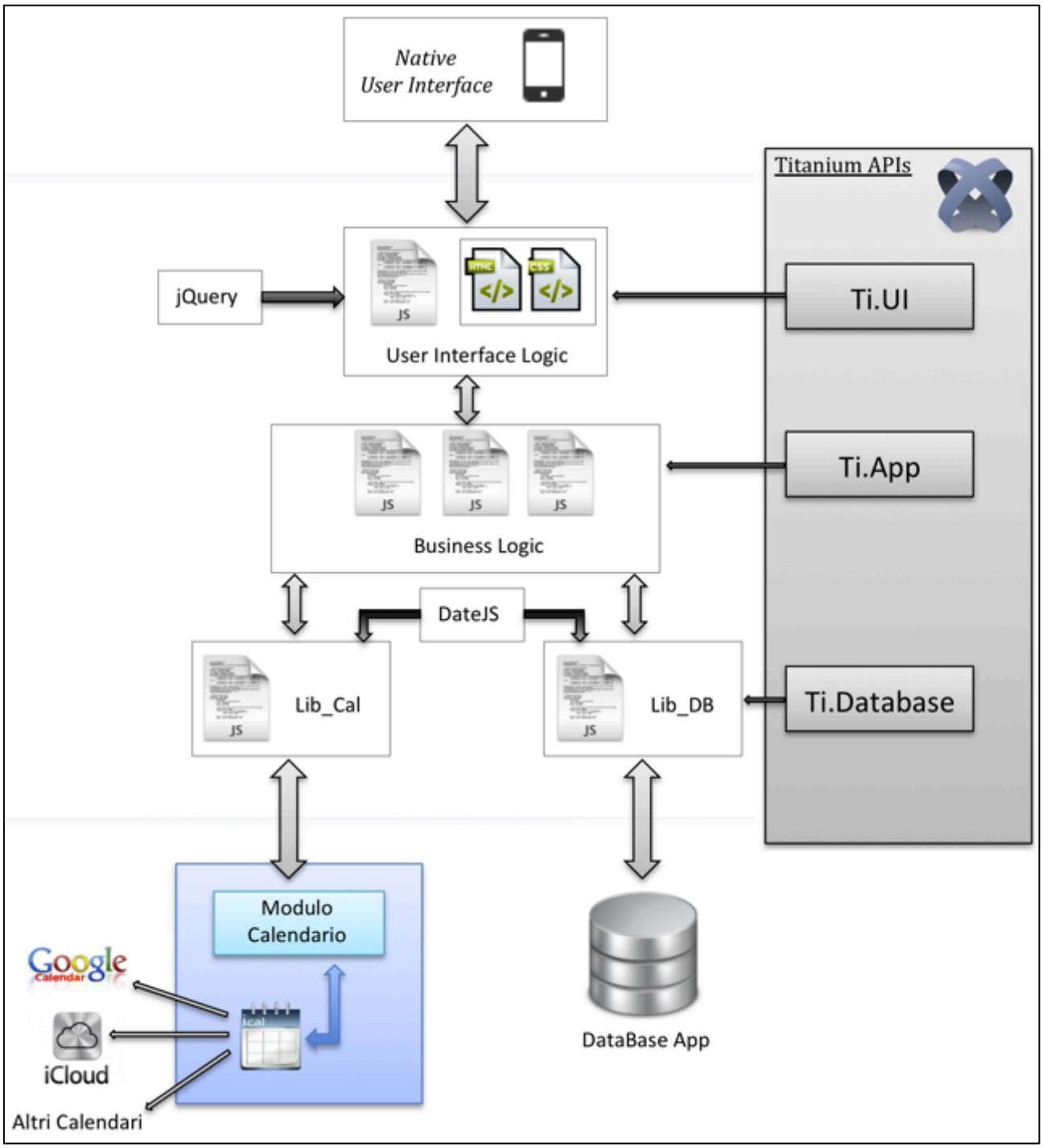



**Figura 14:** Schema dei moduli logici del progetto. Nell'immagine viene fatta una parziale suddivisione in 3 differenti blocchi, dall'alto: gestione dei controlli nativi dell'interfaccia utente del dispositivo; controllo della logica dell'applicazione, interamente sviluppato con JavaScript, HTML e CSS; gestione in linguaggio nativo della memorizzazione persistente (calendario nativo e database). Le APIs Titanium mostrate lateralmente sono parallele a questi 3 livelli in quanto ad ogni livello svolgono la funzione di ponte per la traduzione dei comandi JavaScript in comandi nativi per iOS.

### 5.4.1 – Interfaccia Utente

Nella progettazione delle applicazioni per dispositivi mobili ricopre un ruolo molto importante l'interfaccia grafica. Bisogna tenere conto di vari fattori, primi fra tutti: le ridotte dimensioni degli schermi e l'interfaccia touch screen che caratterizza tali dispositivi.
Il *look and feel* (tutto ciò che appare all'utente durante l'utilizzo dell'applicazione) è spesso di fondamentale importanza per le applicazioni di questo tipo, che devono risultare semplici ed intuitive. A questo proposito per lo sviluppo di CalPlanner sono stati seguiti in linea di massima i punti guida suggeriti dalla guida ufficiale di Apple Developer [12]. Seguendo queste linee guida i principali aspetti di un'applicazione di tipo *Utility*, come quella oggetto di questa tesi, da considerare sono:

- ***Integrità Estetica*:**
  L'applicazione deve essere semplice ed intuitiva anche dal punto di vista estetico. L'usabilità di un'applicazione per la produttività risulta migliore tanto più le funzionalità sono facilmente riconoscibili ed accessibili.

- ***Consistenza*:**
  - *Consistenza con altre app*: si deve cercare di utilizzare le stesse terminologie, gli stessi comandi e le stesse convenzioni di altre applicazioni; ad esempio usando le stesse convenzioni che sono impiegate in iCal.
  - *Consistenza all'interno dell'app*: utilizzando sempre le stesse terminologie e convenzioni grafiche per i medesimi concetti.

- ***Manipolazione Diretta*:**
  Permettere all'utente di utilizzare funzionalità tipiche dei dispositivi iOS come le *gesture*: pinch, scroll, long-tap etc. (cioè combinazioni di movimenti e click sullo schermo touch dei dispositivi mobili). Ad esempio permettendo l'interazione con gli oggetti delle viste e non solamente tramite bottoni o caselle di testo.

- ***FeedBack*:**
  L'app genera messaggi di conferma (o errore) al termine del completamento di un'operazione.

- ***Utilizzo di Metafore*:**
  Utilizzo di icone per rappresentare concetti riconosciuti dall'utente in queste categorie di applicazioni.

- ***Controllo dell'Utente*:**
  L'app non deve prendere decisioni definitive da sola, come la cancellazione di eventi o la loro modifica senza la conferma dell'utente che in ogni momento deve poter annullare l'operazione che sta eseguendo.

L'applicazione è basata su finestre, da ognuna delle quali è possibile accedere ad una funzionalità specifica. In iOS esistono due modi di strutturare il controllo di queste, utilizzando un ***Tab-Group*** o un ***Navigation-Group*.**
*TabGroup* è un oggetto messo a disposizione da iOS (in Titanium mediante l'apposita API) che permette di gestire le finestre parallelamente. Ad ogni finestra corrisponde una *tab*, ed in essa si ha accesso ad una determinata funzionalità.
L'alternativa, scelta per il progetto, è quella di avere un'unica finestra principale dalla quale è possibile scegliere un percorso di funzionalità mediante l'oggetto *Navigation-Group*. Questo oggetto permette di mantenere una gerarchia di finestre basata sull'ordine di apertura. Con questo tipo di approccio risulta più lineare l'utilizzo delle funzioni dell'applicazione, che richiedono spesso più passi con scelte condizionali per l'utente.
In questo modo non si perde l'ordine logico dell'esecuzione delle funzionalità che sarebbe più difficile mantenere con un'organizzazione a *tab*, nelle quali l'utente potrebbe passare da una funzionalità all'altra; per il tipo di applicazione sviluppata risulta anche più semplice ed intuitiva la navigazione tra le funzionalità dell'app.
Le funzionalità dell'applicazione, inoltre, si influenzano a vicenda; ad esempio se vengono inseriti, eliminati o modificati nuovi eventi o nuovi tipi di eventi anche le viste totali e le statistiche risentiranno di questi cambiamenti; lo stesso si può dire se viene sincronizzata un'attività con il calendario nativo. In queste situazioni è quindi necessario propagare le modifiche tra le finestre dell'app. Questo è possibile anche con l'utilizzo di un *Tab-Group*, ma in fase di sviluppo con Titanium si è rivelata una soluzione problematica: lo scambio di messaggi di sincronizzazione tra finestre del *Tab-Group* genera un maggior onere computazionale ed è più difficile da gestire a livello di codice JavaScript. Questi problemi non sono stati riscontrati con l'adozione di un modello a *Navigation-Group*.

Per i motivi citati si è allora scelta la soluzione con *NavigationGroup*.

Ad ogni *Navigation-Group* è associata una *root-window*, cioè la finestra principale dalla quale si ha accesso a tutte le funzionalità dell'app, ognuna delle quali associata ad un'icona. Le altre finestre, che partono dalla *root*, sono:

**Evento Rapido**
Da questa finestra consente l'inserimento di un nuovo evento associato ad un tipo di evento già esistente o ad uno creato sul momento.

**Statistiche**
Da questa finestra è possibile ritrovare in un'unica schermata tutte le statistiche relative ai tipi di eventi memorizzati nel database. E' anche possibile modificare o cancellare i singoli eventi.

**View**
Da questa finestra è possibile visualizzare la vista totale riassuntiva, o la vista proiezione.

**Tipi di Evento**
In questa finestra è possibile gestire i tipi di eventi, modificandone i vincoli.

**Opzioni**
Finestra delle opzioni dell'app. Da qui è possibile eseguire anche l'import (o export) degli eventi da calendario nativo.

### 5.4.2 – Logica Funzionale e Gestione dei Dati

La fase successiva alla definizione delle funzionalità dell'applicazione e delle sue caratteristiche a livello di interfaccia utente, è stata la progettazione logica funzionale dell'applicazione. Si è scelto di progettare l'app in modo modulare, creando due librerie principali che racchiudono tutte le funzioni JavaScript sulle quali si basa l'intera applicazione.

Queste due librerie si basano sulle APIs Titanium, e mettono a disposizione i metodi per la memorizzazione dei dati nel database dell'app CalPlanner e per la comunicazione con il calendario nativo.
Il resto della logica di applicazione è stato implementato in JavaScript, seguendo la struttura richiesta dai progetti in Titanium.

## 5.5 – Problematiche Progettuali

In fase di progettazione sono emerse delle problematiche che hanno richiesto una fase ulteriore di analisi per determinare una soluzione implementabile. Vengono di seguito espresse le problematiche e le soluzioni adottate.

### 5.5.1 – Date, Orari e Standard Impiegati per la Memorizzazione

SQLite adotta per la gestione degli elementi "data" lo standard ISO-8601[13], mentre JavaScript prevede l'esistenza di una classe specifica (Date) per la gestione di date ed orari (formato *date-time*). In JavaScript date ed orari vengono memorizzate in un oggetto con determinati attributi (definiti nelle specifiche ECMA [14]); in SQLite sono delle stringhe.
Secondo le specifiche ECMAScript, anche in JavaScript permette la rappresentazione del formato *date-time* come stringa. E' utilizzata in questo caso una semplificazione dello standard ISO 8601, che non è però compatibile con quello impiegato da SQLite. JavaScript è, però anche compatibile con altri standard e supporta i formati definiti dalle specifiche RFC-1123.

DateJS mette a disposizione una funzione di conversione (*toISOString*) che permettono di convertire le stringhe *date-time* di JavaScript in un formato compatibile per SQLite. Non è però implementata la funzione inversa di "*toISOString*", che è stata allora implementata ed inserita nelle librerie dell'applicazione. Inoltre, queste due funzioni non sono sufficienti perché gestiscono solamente il formato *date-time*, e non *date* e *time* separatamente. Per l'applicazione sono necessarie anche funzioni che permettano di distinguere la parte di informazione che si riferisce all'orario da quella che si riferisce alla data, indipendentemente

dallo standard utilizzato. Sono richieste anche funzioni per la compatibilità tra date in formato anglosassone (mese/giorno/anno) e formato classico (giorno/mese/anno); per questo motivo si devono utilizzare due versioni distinte di DateJS contemporaneamente (quella inglese e quella italiana).
Tutte queste funzioni sono state sviluppate ed inserite nelle librerie JavaScript dell'applicazione.

Per il funzionamento dell'app CalPlanner, sono stati implementati metodi che permettono l'arrotondamento degli orari: è utile in un'applicazione di questo tipo poter suddividere il tempo in slot temporali modulari (di default 15 minuti), così da rendere la ricerca di tempo libero per allocare nuovi eventi più semplice. Per lo stesso motivo sono state sviluppate funzioni ad hoc per la ricerca basate su query SQL e che verificano la disponibilità di un blocco di tempo in caso di inserimento di un nuovo evento.

### 5.5.2 – Modulo Calendario per Titanium

Titanium non mette a disposizione APIs per l'accesso al calendario nativo dei dispositivi mobili. Titanium permette però l'estensione mediante moduli aggiuntivi, di terze parti e open-source. Uno di questi è *Titanium-Calendar* (rilasciata sotto licenza libera Apache GPL v2 [15]; Appcelerator Titanium è rilasciato con questa licenza).
Questo modulo mette a disposizione funzioni per la memorizzazione e recupero da calendario nativo di oggetti di *Tipo Evento*.

Dal punto di vista implementativo consiste in un progetto XCode, generato secondo l'apposita procedura (controllata automaticamente dal framework Titanium). Lo sviluppatore deve occuparsi solo di scrivere il codice Objective-C per l'utilizzo delle APIs di iOS all'interno di file di tipo *modulo* e i corrispondenti *header*. Rispettando alcune convenzioni definite da Titanium è poi il framework Titanium stesso a generare il modulo includibile nel progetto dell'app[16].

Dal punto di vista funzionale il modulo *Titanium-Calendar* offre l'interfaccia JavaScript per le funzioni di memorizzazione e ricerca di eventi nel calendario. La funzione di ricerca restituisce un tipo di dato che in iOS è definito come *NSDictionary* [17], la cui trasposizione in

JavaScript risulta essere una struttura simile ad un array non ordinato di tutti gli eventi presenti. Un'altra caratteristica del modulo è di permettere l'accesso solamente al calendario di default del dispositivo mobile (iCal permette di mantenere più calendari contemporaneamente, solo uno dei quali può essere calendario di default per le applicazioni, in teoria non è però limitato l'accesso agli altri calendari).
Nelle librerie dell'app sono state sviluppate allora funzioni JavaScript per la gestione dei tipi di dato *NSDictionary*, per ricostruire l'ordine cronologico degli eventi elaborati dall'applicazione (funzioni di *sort*).

E' stato anche modificato il modulo calendario in modo da permettere la ricerca di eventi in tutti i calendari di iCal; per la memorizzazione si è scelto (almeno per ora) di utilizzare solo il calendario di default.

Anche il calendario Google Calendar è supportato. Il sistema iOS permette la sincronizzazione automatica con questo calendario che risulta, dal punto di vista della programmazione, un calendario di tipo iCal a tutti gli effetti (gestibile come un calendario nativo). Quindi, per permettere la sincronizzazione con questo calendario non si sono dovuti progettare moduli o funzioni ulteriori.

### 5.5.3 – Realizzazione della Vista Totale

Per la realizzazione della funzione *vista totale*, si è preso di riferimento il *foglio-excel* (figura ).

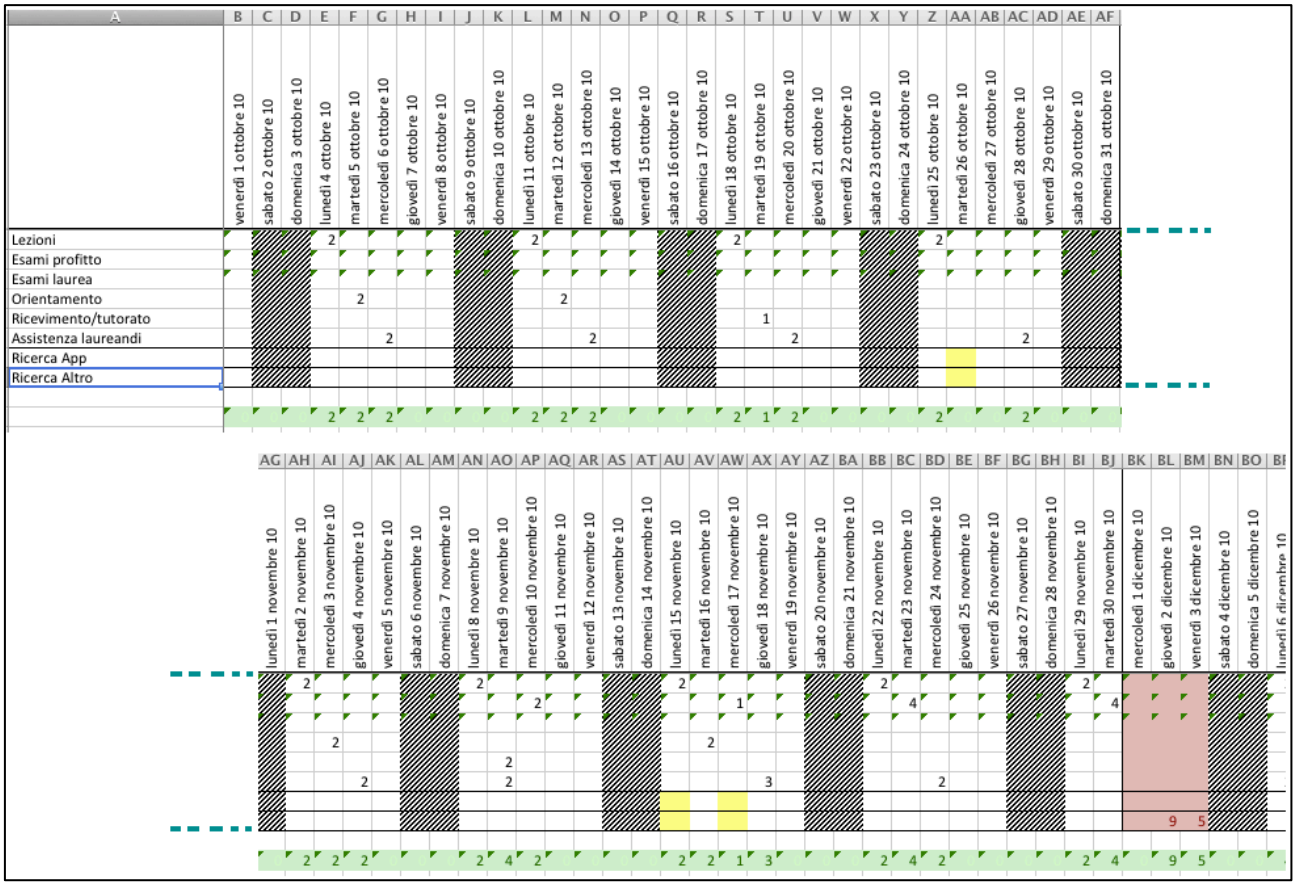



**Figura 15:** Viste riassuntiva consultabile e modificabile del foglio excel. Ogni cella contiene il numero di ore per il giorno indicato in colonna relativo al *tipo di evento* indicato nella riga; il numero all'interno della casella è anche modificabile, ed in caso di evasione di un vincolo viene segnalato visivamente nell'ultima riga (totali). Sono segnalati anche i giorni non lavorativi, nei quali on è possibile inserire ore di alcun tipo di evento, e sono riportati i giorni speciali, nei quali può essere inserito solamente un determinato tipo di evento.

Per generare dinamicamente una tabella di questo tipo le possibilità sono due:

I. Utilizzare componenti grafici messi a disposizione da Titanium come: *views, labels* e *buttons*. La creazione dinamica e la gestione degli eventi (*event handling*) viene gestita tramite le APIs Titanium.UI.

II. Impiegare una pagina HTML generata dinamicamente dall'applicazione utilizzando una pagina HTML preimpostata, un foglio di stile CSS e la libreria jQuery. Con questa soluzione solamente l' *event handling* è gestito da Titanium. Possono poi essere inseriti

nella stessa vista componenti nativi tramite le APIs, ma la tabella viene gestita come una vera e propria pagina web dinamica installata sul dispositivo.

Si è scelto di sviluppare la seconda, la *web view*, per i seguenti motivi prestazionali.

La pagina HTML ed il foglio di stile sono caricati in memoria all'avvio dell'app e sono poco onerosi dal punto di vista computazionale e delle dimensioni occupate. L'esigenza per l'utente è di generare tabelle scorribili dinamicamente, che coprano periodi di tempo molto lunghi, ad esempio un intero anno. Se si considera che possono essere visualizzati più tipi di eventi contemporaneamente, per avere una vista totale appunto, la tabella assume dimensioni elevate. Utilizzando oggetti nativi, creati con APIs Titanium, generandoli dinamicamente, il rischio è di incorrere in *memory lecks*, dovuti alla gestione automatica della memoria da parte del *garbage collector* di JavaScript [18] [19].

Lo svantaggio è di non avere componenti nativi integrati nella vista realizzata. Questo comporta una maggiore complessità dal punto di vista implementativo, soprattutto per la parte di gestione degli eventi. Sono state implementate funzioni apposite per l'interazione delle APIs Titanium con la web view, in modo da poter associare ad esempio anche a componenti HTML *event listener* di Titanium (con i componenti nativi non ce ne sarebbe stato bisogno).

# 6 – Realizzazione del Progetto

In questo capitolo sono trattati gli aspetti implementativi dell'app CalPlanner. Sono descritte le strategie e tecniche di programmazione utilizzate per lo sviluppo dei vari moduli: database, librerie e vista totale; infine è mostrata l'integrazione di questi con l'interfaccia utente.

## 6.1 – Il Database

Il database è stato realizzato utilizzando il tool software **SQLiteManager** seguendo lo schema relazionale definito in fase di progettazione. Questo software permette di eseguire su un database SQLite (versione utilizzata per l'app: SQLite3) qualsiasi espressione che rispetti la sintassi SQLite [20]. Offre anche tools grafici per la gestione di database: creazione, modifica e rimozione di tabelle; gestione dei trigger e viste; strumenti di analisi ed ottimizzazione; esecuzione di query [21].

In Figura 16 e 17 sono mostrate le tabelle del database di *Tipo Evento* ed di *Evento*. A fianco della colonna dei nomi degli attributi, per ogni attributo sono indicati: il tipo di dato, se è chiave, se è richiesto obbligatoriamente (per le chiavi è superfluo) ed un eventuale valore di default.

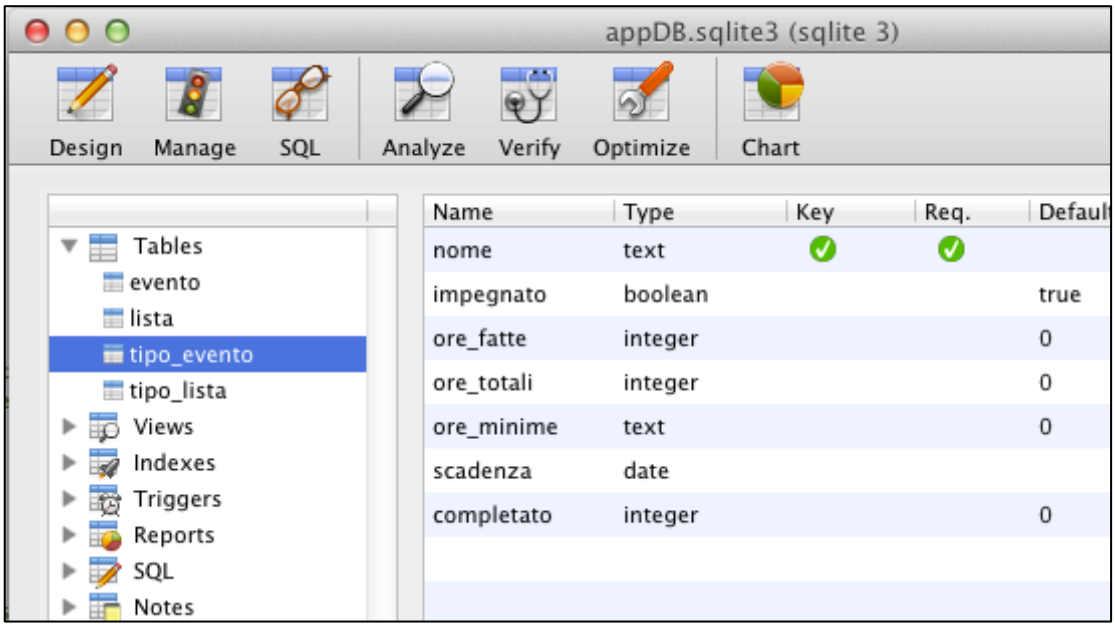



**Figura 16:** Tabella di *Tipo Evento* in **SQLiteManager**.

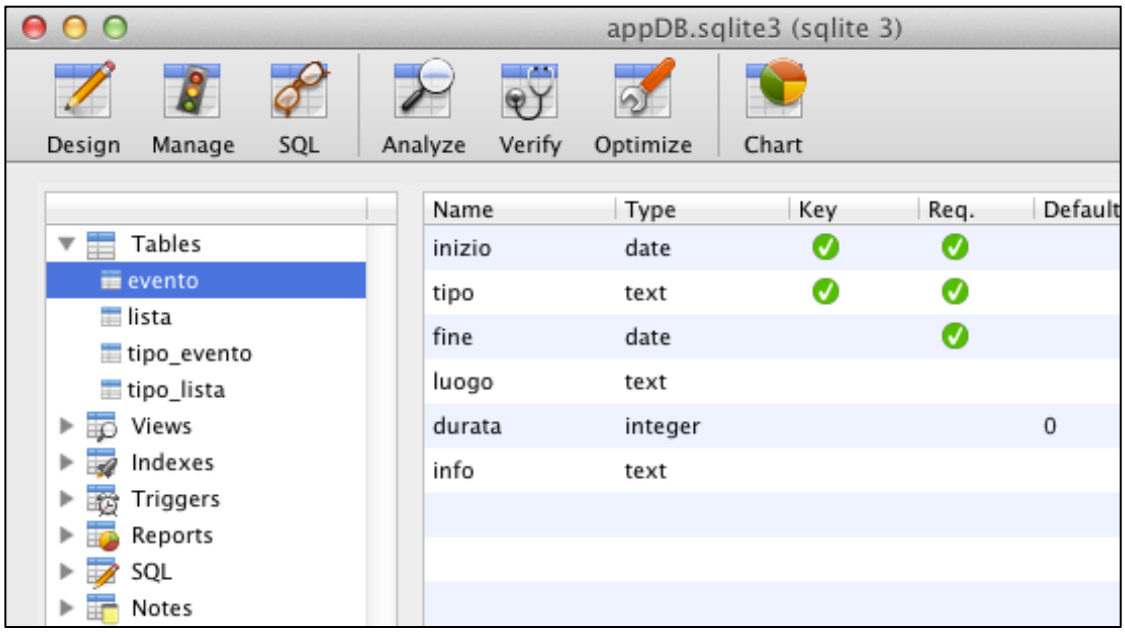



**Figura 17:** Tabella di *Evento* in **SQLiteManager**.

### 6.1.1 – I Trigger

Si sono poi realizzati i trigger necessari a mantenere la coerenza con le regole imposte dall'app e per la derivazione dei dati.

#### *Durata Evento*

Trigger implementato per calcolare il dato derivato *durata_evento* partendo dall'orario di inizio e quello di fine di un evento:

```
CREATE TRIGGER durata_evento
AFTER INSERT ON evento
begin update or ignore evento set durata = ( strftime('%s',new.fine) - strftime('%s',new.inizio) )/60
where tipo = new.tipo and inizio = new.inizio;end
```

#### *Dato Derivato Ore_Totali*

Questi due trigger servono per il calcolo del dato derivato *Ore_Totali* per ogni tipo di evento. Quando un evento viene aggiunto, modificato o cancellato, la sua durata viene sommata (o sottratta) al valore di *Ore_Totali* del *Tipo Evento* a cui appartiene.

**Inserimento o modifica di *evento*:**

```
CREATE TRIGGER tipo_evento_ore_tot_aggiungi
AFTER UPDATE ON evento
begin update or ignore tipo_evento set ore_fatte = ore_fatte + new.durata - old.durata
where nome = new.tipo;end
```

**Cancellazione di *evento*:**

```
CREATE TRIGGER tipo_evento_ore_tot_rimuovi
AFTER DELETE ON evento
begin update or ignore tipo_evento set ore_fatte = ore_fatte - old.durata
where nome = old.tipo;end
```

***Completamento di un Tipo Evento***

Trigger per il calcolo dello stato di completamento di un *Tipo Evento*; così da poter riconoscere le attività terminate. Con questo:

```
CREATE TRIGGER completamento_totale
AFTER UPDATE ON "tipo_evento"
begin update or ignore tipo_evento set completato = 1
where nome = new.nome and ore_totali <= ore_fatte;end
```

### 6.1.2 – SQL Query Per la Verifica di Disponibilità Temporale

All'interno dell'applicazione sono diverse le funzioni che effettuano l'accesso al database tramite l'API Titanium Ti.Database. Le query eseguite variano da funzione a funzione; una in particolare, però, sta alla base dell'intero funzionamento della gestione del database: verificare la disponibilità di un blocco temporale dati in ingresso valori di tipo *date* di inizio e di fine.

Purtroppo SQLite non permette la creazione di *stored procedure*, la query è stata allora inserita in una specifica funzione all'interno di una variabile JavaScript. I quattro casi possibili considerati nella query sono rappresentati in figura 18.

**Query per ricerca della disponibilità temporale:**

```
/*
start: inizio del periodo del quale si vuole verificare la disponibilità
end: fine del periodo del quale si vuole verificare la disponibilità
*/
select * from evento e]
where (e.inizio like ?)                    -- ? = start [date]
and (
        ( e.inizio <= ? and e.fine > ? )   -- ? = start [date-time];  ? = start [date-time]
        or ( e.inizio < ? and e.fine >= ? ) -- ? = end [date-time];  ? = end [date-time]
        or ( e.inizio >= ? and e.inizio < ?) -- ? = start [date-time];  ? = end [date-time]
        or ( e.fine > ? and e.fine <= ? )  -- ? = start [date-time];  ? = end [date-time]
)
```

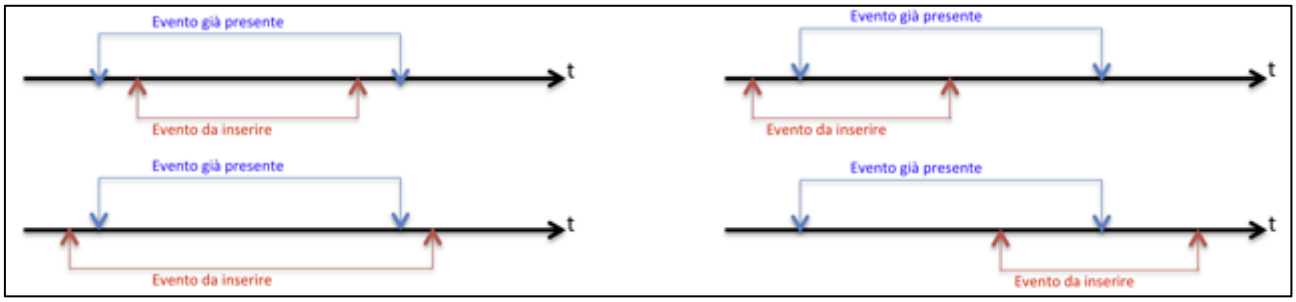



**Figura 18:** In blu è rappresentato un evento già presente nel database; in rosso i posizionamenti non ammessi, che creerebbero sovrapposizione.

## 6.2 – Il Progetto in Titanium

L'app CalPlanner è stata sviluppata con l'ausilio del framework Titanium ed il suo ambiente di sviluppo integrato (IDE) Titanium Studio. Utilizzando questo strumento i progetti per dispositivi mobili devono avere una struttura predefinita, generata automaticamente dalla piattaforma. Lo sviluppo dell'applicazione procede con l'inserimento di file JavaScript all'interno della cartella *Resource*, con riferimento al file principale *app.js* (il primo ad essere caricato all'avvio dell'app).

### 6.2.1 – Struttura del Progetto

In figura 19 è mostrata la struttura del progetto in Titanium dell'app sviluppata.

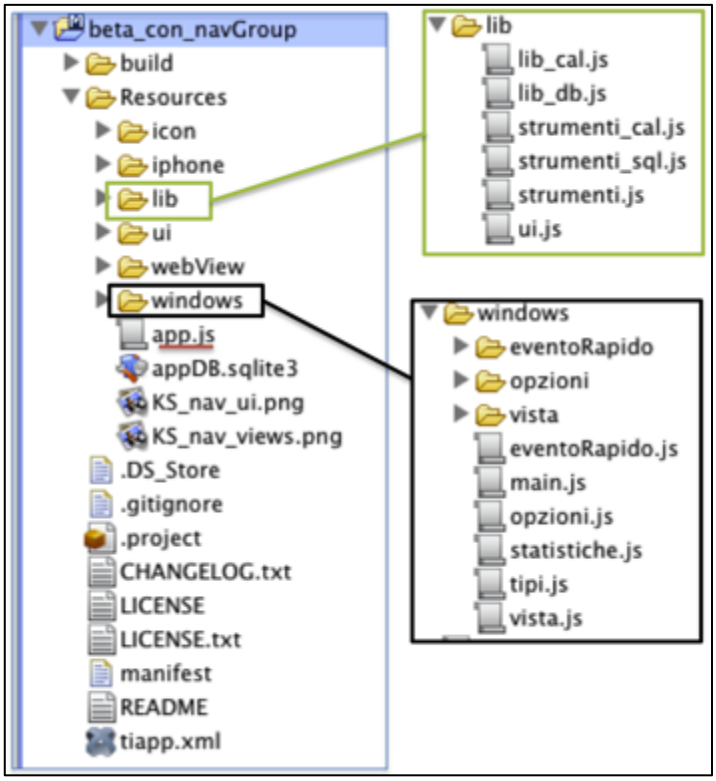



**Figura 19:** Struttura del progetto dell'app in Titanium Studio. Nel dettaglio la cartella *lib* e *windows*.

Il file *tiapp.xml* è un file xml contenente direttive per la compilazione dell'app; è modificabile, ed è qui che va specificato il *path* del modulo *Titanium-Calendar* affinché venga incluso. Per il resto tutti i file necessari alla realizzazione dell'app sono contenuti nella cartella *Resources*.

Il progetto è strutturato in modo che all'avvio, dal file *app.js* venga caricata una *window* principale dalla quale è possibile richiamare tutte le funzionalità dell'app, contenute in *window* secondarie. Il codice di queste *window* è in file separati, contenuti nella cartella *windows*, con l'unica eccezione per i file necessari a generare la "vista-totale" che sono nella cartella *webView*. Tutti gli strumenti (funzioni JavaScript) che le *windows* impiegano nel loro funzionamento sono contenute nelle librerie JavaScript, che si trovano nella cartella *lib*. Tutte le componenti grafiche, i moduli aggiuntivi (JavaScript, non nativi), sviluppati per l'interfaccia utente e utilizzati anch'essi dalle *window* sono nella cartella *ui*. In *Resource* si trova anche il database e tutte le immagini o icone utilizzate per l'interfaccia grafica.

Infine, dopo la compilazione con Titanium del progetto, nella castella *build* verrà generato il progetto XCode dell'app. Questo progetto può essere compilato ed installato sull'emulatore od un dispositivo iOS.

***Il File app.js***

Nel file *app.jsi* si trovano le dichiarazioni per inizializzare tutti gli elementi necessari per il funzionamento dell'app. Come prima cosa viene dichiarata una variabile (*AppGio*) che svolge la funzione di *namespace* per le variabili, impiegate nell'applicazione, condivise, visibili da tutti i file JavaScript del progetto:

```
//dominio per variabili dell'app
var AppGio = {
      // un "sottospazio" è, ad esempio, ui
      ui : {
            bgImage : 'icon/wood_l.png',
            bgImageView: 'icon/wood_l2.png',
            bgColor : '#BBEEFF',
            dashboardHeight : 280,
            dashboardWidth : 306,
            // winOpened diventa un sottospazio a sua volta
            //invocabile con: AppGio.ui.winOpened
            winOpened:{},//window aperta

            //funzioni PRINCIPALI. Sono solo dichiarati i nomi,
            //le funzioni sono implementate altrove
            createEventWindow : function() {},
            createStatsWindow : function() {},
            createViewWindow : function() {},
            createOptionWindow : function() {},
            createTipiWindow : function(){},
      }
```

Poi, è inizializzato il database:

```
var db_install = Ti.Database.install('appDB.sqlite3', 'appDB');
db_install.close();
```

Infine sono inclusi tutti gli altri file JavaScript necessari; tra i quali, *main.js* che contiene il codice di inizializzazione della finestra principale dell' User Interface.

### 6.2.2 – Implementazione delle Librerie

Le librerie JavaScript sono state divise per ambito di utilizzo: in *lib_cal.js* si trovano le funzioni e classi di oggetti JavaScript per la gestione del calendario nativo, in *lib_db.js* quelle per la gestione del database e per la sincronizzazione database-iCal.

***Libreria lib_cal.js***

All'interno di questo file ci sono le istruzioni per inizializzare il modulo *Titanium-Calendar* e le funzioni per interfacciarsi con esso. In questa libreria tutte le date sono oggetti JavaScript; con la dicitura "Date" si intende quindi un oggetto di questo tipo.

**Crea Evento**

Questa funzione permette di creare un oggetto *evento* compatibile con il modulo calendario. I suoi parametri in ingresso sono:

- Titolo [String]
- Data di inizio [Date]
- Data di fine [Date]
- Luogo [String]

Restituisce in uscita un oggetto di *Tipo Evento* con i seguenti metodi necessari a modificare i parametri senza dover creare un oggetto nuovo e un metodo *salva()* per memorizzare l'evento sul calendario nativo sfruttando l'interfaccia messa a disposizione dal modulo calendario.

**Trova Evento**

Per quanto riguarda il recupero (sia per la visualizzazione che per la cancellazione) il procedimento è più complesso che per la memorizzazione. Il modulo calendario mette a disposizione solamente un'interfaccia per il recupero degli eventi in un determinato arco temporale, e restituisce un oggetto di tipo NSDictionary, nel quale viene perso l'ordine cronologico degli eventi (sono ordinati in base al loro nome e non alla data). Per ricostruirlo è necessario processare tale oggetto nel seguente modo:

```
//oggetto risultati ricerca che parte da e.start fino a e.end
var tmp = calendar.findEvents({ start : e.start, end : e.end });
//restituisco come array di eventi_cal
var ret = [];
// array risultati temporanei da 'tmp' ---> da ordinare
var eventi = [];
//titolo che si sta cercando = e.title
if(e.title) {
	for(i in tmp) {
		if(tmp[i].title === e.title) {
			eventi.push(i);
		}
	}
eventi.sort();
}
```

Nella libreria sono implementati anche metodi per la ricerca di eventi basata sulla **similarità dei titoli**, ricercando all'interno dei titoli parole specifiche (anche parti di parole). Questa funzionalità è stata sfruttata per la sincronizzazione iCal-databse. Gli oggetti restituiti sono di *Tipo Evento*. Per ottenere questo risultato viene sfruttata la funzione *match* dell'oggetto *String* di JavaScript:

```
for(var j = 0; j < e.parole.length; j++) {
	//parole_di_titolo sono le parole che vengono ricercate
	parte_di_titolo = e.parole[j];
	parte_di_titolo = parte_di_titolo.toLowerCase();
	match = titolo_intero.match(parte_di_titolo);

	if(match != null) {
		ok = true;
	}
	else {
		ok = false;
		break;
	}
};
```

**Cancella Evento**

Questa funzione è stata implementata da quella di ricerca, poiché per la cancellazione di un evento, come prima cosa, è necessario il recuperare gli oggetti *evento* corrispondenti. Successivamente, viene cancellato l'evento corrispondente sul database e infine quello su iCal mediante l'interfaccia fornita dal modulo calendario.

***Libreria lib_db.js***

In questo file sono definite le classi *Tipo Evento* ed *Evento* corrispondenti alle tabelle implementate nel database. I loro metodi e le altre funzioni qui definite stanno alla base di tutte le funzionalità sviluppate nell'app.

Tutte le operazioni che comportano la modifica del database si basano su query SQL (sintassi SQLite3), eseguite tramite l'API Titanium.database tramite il comando "*execute*". La libreria è impostata anche in modo da richiedere l'utilizzo di soli oggetti JavaScript Date per la rappresentazione delle date. Le trasformazioni per la compatibilità con il database, per le funzioni che leggono o scrivono su di esso, è gestita automaticamente con funzioni apposite. In questo modo in tutto il progetto si utilizzano solo oggetti Date, e le conversioni del caso sono lasciate alle funzioni della libreria.

**Tipo Evento**

E' il corrispettivo JavaScript dell'entità *Tipo Evento* del database, i suoi attributi sono:

- Nome [String]
- Ore_totali [integer]
- Ore_minime [integer]
- Ore_fatte [integer]
- Scadenza [Date]
- Completato [Boolean]

All'interno della libreria sono implementati metodi per creare un nuovo tipo, aggiornarlo o eliminarlo. Altre funzioni utili sono:

- ***getTipo()***: dato in ingresso una stringa rappresentate il nome, restituisce l'oggetto *Tipo Evento* corrispondente, se esiste. Si basa sull'esecuzione della query:

```
db = AppCal.db.apri_db();
var res = db.execute('select * from tipo_evento where nome = ?', tipo_evento);
```

- ***getEvents()***: dato in ingresso un nome di *Tipo Evento* restituisce un array contenente tutti gli eventi associati ad esso. Specificando la data di inizio e di fine è possibile limitare la ricerca all'interno di quell'arco temporale. La query SQL per la ricerca nel database è:

```
ris = db.execute('select
tipo,datetime(inizio,\'localtime\'),datetime(fine,\'localtime\'),luogo,durata,info
'+
'from evento where date(inizio,\'localtime\') >= ? and date(inizio,\'localtime\')
<= ? and tipo = ? order by inizio',
inizio, fine, tipo);
```

- ***getTuttiTipi()***: restituisce un array di stringhe contenente tutti i nomi dei tipi di evento memorizzati nel database.

**Evento**

Per evitare ambiguità, ci si può riferire a questa classe anche con il termine: *evento_db*, per distinguerla da quella impiegata in *lib_cal*.

Questo oggetto rappresenta l'entità *Evento* e i suoi attributi sono:

- Inizio [Date]
- Fine [Date]
- Tipo [String]
- Luogo [String]
- Durate [Integer]
- Info [String]

I suoi metodi permettono di scrivere, modificare ed eliminare l'evento che rappresenta nel database.

**Gestisci Import**

La funzione per la gestione degli import, *gestisciImport()*, ha come parametri di input:

- **Array di n titoli** (*String[n]*) che identificano oggetti evento memorizzati in iCal (questi sono recuperati tramite le funzioni di *lib_cal*) che si vogliono importare.

  Questi titoli non sono identificatori di oggetti evento di iCal, possono esistere più eventi con lo stesso titolo nello stesso periodo di tempo ricercato. La funzione è in disambiguare tali eventi creando differenti *eventi_db* corrispondenti. Questo e la ricerca basata solo su parti di titolo permettono di implementare funzioni per ricerche dinamiche, basate sul contesto e sulla similarità semantica dei titoli degli eventi (sviluppi futuri: riconoscimento sinonimi con l'ausilio di un dizionario per il confronto dei termini).
- **Data di inizio** (*Date*), data dalla quale deve partire la ricerca.
- **Data di fine** (*Date*), data di fine ricerca.
- **Titolo di Tipo di Evento** (*String*), un nome di un tipo di evento valido nel quale si vogliono far confluire gli eventi importati.

Sono creati *eventi_db* per ogni evento di iCal, associati al tipo di evento specificato; successivamente vengono memorizzati.

Per mantenere coerenza tra iCal e database, prima di memorizzare gli eventi associandoli ad un tipo di evento, quest'ultimo viene svuotato. In questo modo viene mantenuta sempre una copia che rispecchi gli eventi presenti in iCal. Questo perché l'app rimane un'applicazione di supporto a iCal, se ad esempio l'utente cancella un evento memorizzato su iCal che era stato precedentemente importato, viene considerata quella modifica più importante. Gli eventi importati possono appartenere anche a calendari differenti, è l'utente a determinare quali far confluire in un determinato tipo. Ad esempio in uno stesso *Tipo Evento* possono confluire eventi provenienti dal calendario *work* di iCal e da quello di Google Calendar.

**Verifica Disponibilità**

La funzione di ricerca della disponibilità di uno slot temporale nel database, ha come parametri di ingresso: ***inizio*** [Date]; ***fine*** [Date]. La ricerca si basa sulla query SQL già descritta nel paragrafo 6.1.2 e restituisce in uscita un valore booleano per indicare se lo slot temporale è libero (*true*) o occupato (*false*).

**Trova Disponibilità**

Questa funzione serve per ricercare insiemi di slot temporali da allocare per la pianificazione di attività, mantenendo la coerenza con gli eventi memorizzati nel database.

In input richiede:

- Data di inizio e di fine del *Tipo Evento* (nuovo) che si vuole pianificare
- Ore minime e/o ore totali del *Tipo Evento*.
- Ore minime e/o ore massime per settimana.

  Variabile booleana per indicare la priorità tra ore minime ed ore massime per settimana, che si traduce in:
  - Priorità ore minime = Si cerca di allocare il minimo di ore ogni per settimana.
  - Priorità ore massime = Si cerca di allocare il massimo di ore per settimana.
- Array di 7 variabili booleane rappresentanti i giorni della settimana all'interno dei quali è possibile allocare ore.
- Ora di inizio e ora di fine della fascia oraria ammissibile.
- Ore minime e ore massime per ciascun incontro.

  Variabile booleana per indicare la priorità tra ore minime ed ore massime per ogni singolo evento, che si traduce in:
  - Priorità ore minime = Si cerca di allocare il minimo di ore per ogni incontro.
  - Priorità ore massime = Si cerca di allocare il massimo di ore per ogni incontro.

Il risultato in uscita è un array di oggetti di tipo *disponibilità*, che hanno come attributi: *ore_di_inizio* e *ora_di_fine*. Questo array rappresenta l'insieme degli slot temporali allocabili per un dato tipo di evento.

L'esecuzione dell'algoritmo avviene secondo le priorità indicate dai parametri in input ed esegue un ciclo di diminuzione delle priorità finché non è ritrovato un array compatibile con i vincoli, ad esempio, in base alle condizioni imposte inpartenza:

- **Priorità: ore minime settimanali**

  Se al termine dell'algoritmo non sono state ritrovate sufficienti ore si ripete l'esecuzione cerca di allocare il numero massimo di ore settimanali. Se quest'ultimo caso fallisse, allora non è possibile trovare soluzione con i vincoli iniziali dati.

- **Priorità alle ore minime per incontro**
  Se al termine dell'algoritmo non sono state ritrovate sufficienti ore si ripete l'esecuzione cercando di allocare il numero massimo di ore per incontro.

La ricerca all'interno di un determinato giorno in una determinata fascia oraria si basa sulla costruzione di una matrice di oggetti di tipo *disponibilità*. Questa matrice viene costruita a partire dagli eventi già presenti in memoria e che occupano parte del blocco.

In figura 20 si può vedere lo schema logico dell'algoritmo di ricerca delle disponibilità. Viene eseguito un ciclo che si conclude solo quando è stato ricercato l'intero periodo indicato o quando sono state trovate abbastanza ore da allocare. All'interno del ciclo per ogni giorno valido viene costruito un array delle disponibilità a partire dagli eventi memorizzati in quel giorno, nella fascia orari specificata. In ogni giorno viene allocato un numero di ore minimo finché non sono alloccate un numero minimo di ore settimanali. Se in una settimana non si raggiunge il numero prefissato di ore, o alla fine del periodo indicato non si è raggiunto un numero di ore totali sufficienti, la funzione si interrompe e restituisce un vettore vuoto. Se invece la funzione termina correttamente viene restituito un vettore di disponibilità valido per i vincoli indicati.

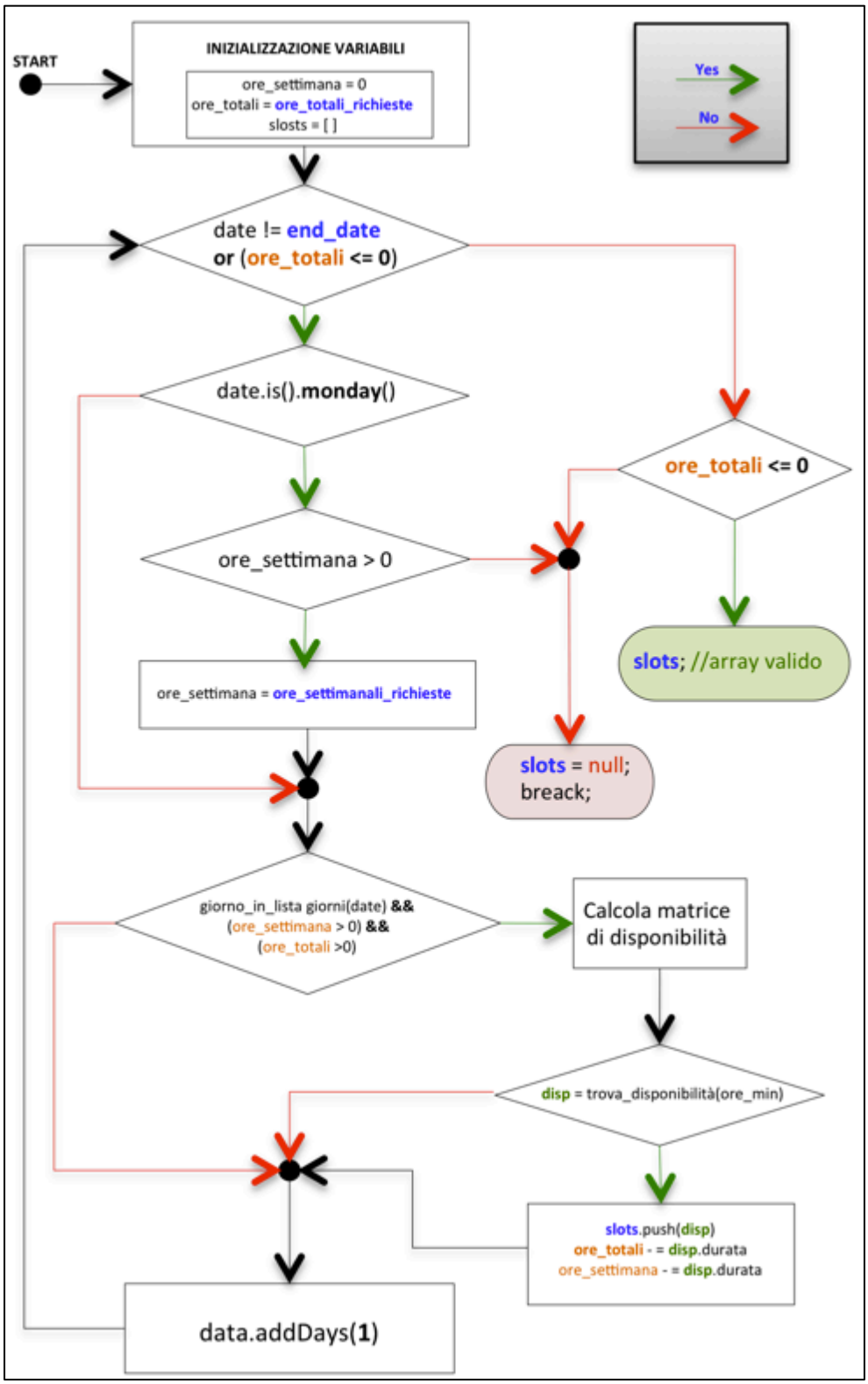



**Figura 20:** Diagramma dell'algoritmo di ricerca della disponibilità. La variabili sono: **slots** è l'array che conterrà al termine dell'esecuzione tutti gli oggetti *disponibilità* (**disp**) che giorno per giorno sono stati trovati; **ore_settimana** è il contatore delle ore della settimana corrente allocate, se alla fine della settimana (cioè il lunedì) queste ore non sono state raggiunte l'algoritmo si interrompe e slots risulterà vuoto; ore_totali è il contatore che sarà inizializzato con il numero di ore che si vuole allocare; **date** inizialmente è uguale alla data di inizio di ricerca, e viene modificata ogni ciclo (da essa dipende il controllo condizionale iniziale); **end_date** è costante ed uguale alla data di fine. Le funzioni che vengono utilizzate sono: **is().monday()** serve per conoscere se a quella data corrisponde il giorno della settimana "lunedì" (è una funzione di DateJS); **giorno_in_lista()** verifica che il giorno corrispondente alla data sia uno di quelli in cui bisogna cercare;**calcola_matrice_disponibilità()** costruisce un arrai di oggetti *disponibilità* corrispondente alla fascia oraria indicata, in base agli eventi presenti in memoria; **trova_disponibilità()** cerca una *disponibilità* con dimensione maggiore o uguale alla dimensione minima per incontro, se esiste l'aggiunge in coda all'array **slots.**

### 6.2.3 – Implementazione della Vista Totale

La vista totale che permette di visualizzare tipi di eventi in un'unica tabella riassuntiva è stata realizzata mediante una pagina HTML che viene caricata all'interno di una finestra mediante le APIs Titanium. I contenuti e il DOM (*Document Object Model*) della pagina vengono controllati con istruzioni JavaScript. Alcune di esse sono implementate sfruttando anche le librerie DateJS e jQuery.

La struttura di questa pagina è mostrata in figura 21:

**Figura 21:** Struttura della vista totale.

Le componenti di questa vista sono:

- *Header*: parte superiore per la visualizzazione del mese e dei giorni.
- *Vista Sinistra*: contiene i nomi dei tipi di evento visualizzati
- *Vista Centrale*: contiene tutte le caselle delle ore di un dato tipo memorizzate in determinato giorno.
- *Footer*: Vista in basso con il totale delle ore per giorno.

E' stato definito anche un foglio di stile (CSS) che insieme a funzioni JavaScript permette di avere un *look and feel* avanzato rispetto ad una pagina HTML semplice; le funzioni permettono di:

- Scorrere la vista verso destra (o sinistra, se si torna al punto iniziale) mantenendo in posizione fissa la *Vista Sinistra*. L'algoritmo che permette questo è il seguente e che si attiva ogni volta che viene effettuato uno scroll è il seguente:

```
var scroll_old_x = 0;
var scroll_old_y = 0;

$(window).scroll(function() {
	var bar = document.getElementById("left-bar");
	var old_top = $("#left-bar").css('top');
	var top_num = parseInt(old_top.split('px')[0]);
	var s_x = $(window).scrollTop() - scroll_old_x;
	scroll_old_x = $(window).scrollTop();
	var diff_x = top_num - s_x;
	var new_top = diff_x + "px";

	$('#left-bar').css('top', new_top);
});
```

- Scorrere in alto (o in basso) mantenendo in posizione fissa l'*header* ed il *footer*:

```
var scroll_old_x = 0;
var scroll_old_y = 0;

$(window).scroll(function() {
	var header = document.getElementById("header");
	var old_left = $("#header").css('left');
	var left_num = parseInt(old_left.split('px')[0]);
	var s = $(window).scrollLeft() - scroll_old_y;
	scroll_old_y = $(window).scrollLeft();
	var diff = left_num - s;
	var new_left = diff + "px";

	$('#header').css('left', new_left);
	$('#footer').css('left', new_left);
});
```

- Ingrandire la vista mantenendo le proporzioni delle celle con le gesture tipiche di iOS (pinch).
- Possibilità di cliccare ogni cella e gestire l'evento con le APIs Titanium, ad esempio modificando le ore alle quali la cella si riferisce passando il controllo a componenti nativi (*picker* o *textField*).

Le funzioni JavaScript che controllano la creazione dinamica della vista durante l'esecuzione dell'app sono:

- **visualizza_periodo()**: prende in ingresso due date, inizio e fine, e genera l'*header*.
- **visualizza_evento()**: prende in ingresso un nome (nome di tipo) ed un array di interi. Genera una riga inserendo il nome nella *vista sinistra* e una serie di celle alla stessa altezza nella *vista centrale*.

  L'arrey viene calcolato a sua volta con una funzione **calcola_ore()** che dato un nome di tipo e un periodo di tempo calcola quante ore per giorno sono memorizzate per quel dato tipo (se nessuna inserice 0).
- **visualizza_totale()**: Prende in ingresso un array di interi. Questo array è costruito run time mentre vengono eseguite le funzioni **calcola_ore()** mantenendo un array parallelo.
- **controllo_totale():** In base alle regole dell'app quando una casella di totale non soddisfa i vincoli viene modificata di colore e segnalata all'utente visivamente.

## 6.3 – User Interface

L'ultima fase della realizzazione del progetto è l'implementazione dell'interfaccia utente: la finestra principale dell'app, il menù, e tutte le altre finestre secondarie per l'accesso alle funzionalità.

Ad ogni finestra è associato un file JavaScript. All'interno di questo file sono dichiarati tramite le APIs Titanium.UI tutte le componenti che vanno a costituire la vista e i loro parametri di configurazione (*bottoni, picker, label, table-view, text-field etc.*). Tramite chiamate a funzioni delle librerie (*lib_cal* e *lib_db*) e la gestione dei risultati da esse prodotte si effettua il controllo degli eventi (*event-handling*) con l'implementazione della logica di applicazione vera e propria.

Per ottenere maggior modularità del codice ogni finestra è creata con l'invocazione a una funzione. Inoltre questa scelta ha un vantaggio dal punto di vista prestazionale: ogni funzione è eseguita come una *closure* (la visibilità delle variabili è locale e al termine dell'esecuzione viene liberata automaticamente la memoria allocata non più necessaria). Inoltre con l'utilizzo di funzioni per la creazione di finestre, in punti diversi dell'app è possibile richiamare la creazione della stessa finestra con una semplice invocazione.

### 6.3.1 – Moduli delle Componenti Grafiche Personalizzate

Titanium.UI mette a disposizione le interfacce JavaScript per il controllo di innumerevoli componenti UI native di iOS, sono però componenti base. Modificando i loro parametri e utilizzandone insieme, si costruiscono oggetti personalizzati (*custom*). Accade spesso che questi componenti *custom* possano essere riutilizzati in più momenti all'interno dell'app; è, allora, conveniente creare moduli per la loro inizializzazione automatica. Inoltre, se i componenti sono richiamati solo nel momento in cui vengono utilizzati, utilizzando questa tecnica di programmazione di Titanium si ha anche un vantaggio prestazionale poiché la memoria viene allocata solo quando indispensabile e rilasciata non appena l'oggetto non è più utilizzato.

Di seguito, in figura 22, sono mostrati due esempi di modulo personalizzato. Questi *picker* hanno metodi per la comparsa/scomparsa automatica, impostazione automatica all'ora/data corrente, memorizzazione dei valori modificati alla pressione del bottone "done", sfondo personalizzato e altezza modificabile (i componenti nativi hanno dimensioni fisse).

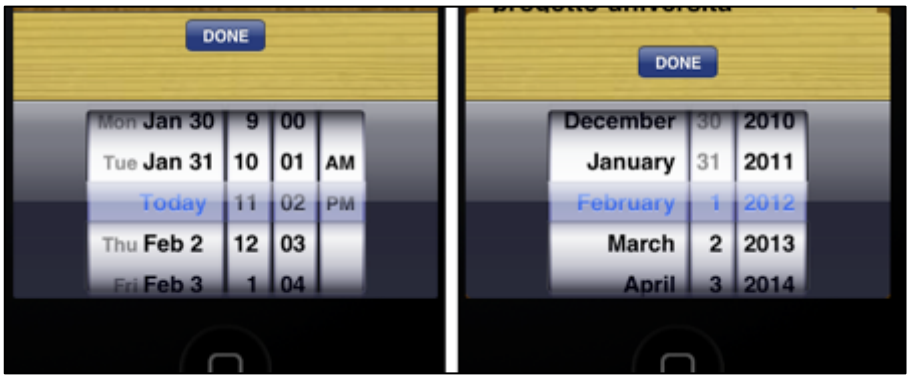



**Figura 22:** Componenti picker personalizzati. A sinstra picker_data_ora. A destra picker_data_anno.

Nel file JavaScript nel quale si vogliono impiegare, i moduli sono richiamati con la funzione:

```
var customPickerData = require('ui/customPicker').pickerData;
```

```
////////////////////////////////////////////////////////////////////////////////
// Picker Data a scomparsa
////////////////////////////////////////////////////////////////////////////////
var pickerData = new customPickerData();
```

Così, sarà possibile invocare i costruttori degli oggetti esportabili tramite il modulo:

In questo modo, è stato inizializzato un'oggetto *custom*, un picker a scomparsa per la selezione di data e ora, al quale è possibile associare, ad esempio, un *eventListener* con il quale si può controllare un oggetto *Date*.

### 6.3.2 – Le Finestre delle Funzioni

Le principali finestre implementate nell'app sono:

- *Main*
- *Evento Rapido*
- *Vista* [1]
- *Statistiche*
- *Tipi Evento*
- *Import*

#### *Main*

*Main*, in figura 23, è la prima finestra a essere aperta all'avvio. A essa è associato il *Navigation-Group* (gestore della gerarchia delle finestre aperte). All'apertura carica dal file *icone.js* i dati necessari per creare le icone delle funzioni che appaiono nella sua *deshboard* centrale. Ad ogni icona è associata una delle funzioni per la creazione di finestre. Quando una di queste icone viene cliccata la funzione crea una finestra che viene aperta e aggiunta al *Navigation-Group*. La finestra main è anche la finestra "base": quando viene cliccata l'icona "*home*" in qualsiasi punto dell'app, tutte le operazioni in esecuzione vengono annullate, le finestre vengono chiuse e si viene riportati in questa finestre.

**Figura 23:** Da sinistra: finestra *Main*; finestra *Evento Rapido*; finestra *Nuovo Tipo* per inserimento di un nuovo tipo di evento.

[1] Già descritta nel capitolo 6.2.3

Di seguito è mostrato il codice completo JavaScript per la finestra *Main*.

```
//Modal è necessaria per gestire navGroup
var modal = Ti.UI.createWindow();

//La finestra principale:
var mainWin = Ti.UI.createWindow({
      navBarHidden : true,
      title : 'Home',
      backgroundImage : AppGio.ui.bgImage,
});

//L'oggetto navigation-group
navGroup = Ti.UI.iPhone.createNavigationGroup({
      window : mainWin,
});
modal.add(navGroup);
modal.orientationModes = [Ti.UI.LANDSCAPE_LEFT,Ti.UI.LANDSCAPE_RIGHT,Ti.UI.PORTRAIT];

//Apro la finestra modal.
modal.open();

//View dove verranno visualizzate le icone
var viewIcon = Ti.UI.createView({
      height : AppGio.ui.dashboardHeight,
      width : AppGio.ui.dashboardWidth,
      backgroundImage : AppGio.ui.bgImageView,
      bottom : 20,
      borderRadius : 2,
      layout : 'horizontal',
});
/
/Aggiungo le icone alla vista
for( i = 0; i < 5; i++) {
      viewIcon.add(createIcon(AppGio.ui.icons.list[i]));
}
mainWin.add(viewIcon);
```

Il codice in Titanium per la dichiarazione di una finestra è racchiuso in un solo comando: *Titanium.UI.createWindow()* ed alcuni parametri, a seconda del layout che si vuole ottenere (proprietà della finestra). Per aprire la finestra è sufficiente invocare il comando *open()*. Alla finestra, poi, possono essere aggiunti tutti gli oggetti del caso (label, bottoni, etc.) con il comando *add()* e si è già inizializzata una finestra visualizzabile.

Nel caso della finestra *Main*, viene impiegato anche un oggetto *navigationGroup* ed una finestra ausiliaria (*modal*), che non verrà però visualizzata, serve solo come "contenitore" di tutte le finestre. Mentre tramite la funzione "*createIcon()*" vengono costruite le icone, che sono gli elementi cliccabile che permettono l'accesso alle altre finestre.

```
AppGio.ui.icons = {
	height : 30,
	width : 30,
list : [{image:'icon/todo.png',func:AppGio.ui.createEventWindow,nome:'evento rapido'},
	{image:'icon/stats.png',func:AppGio.ui.createStatsWindow,nome:'statistiche'},
	{image:'icon/view.png',func:AppGio.ui.createViewWindow,nome:'view'},
	{image:'icon/tipo_evento.png',func:AppGio.ui.createTipiWindow,nome:'tipi di evento'},
	{image:'icon/option.png',func:AppGio.ui.createOptionWindow,nome:'opzioni'}]
};
//funzione per creare ogni icona della vista principale
var createIcon = function(icon) {
	//la win che si crea chiamando la funzione associata all'icona premuta
	var iconWin = undefined;
	var view = Ti.UI.createView({height : 85,width : 153,});
	var icona = Ti.UI.createView({
		backgroundImage : icon.image,
		top : 20,
		left : 60,
		height : AppGio.ui.icons.height,
		width : AppGio.ui.icons.width
	});
	var label = Ti.UI.createLabel({
		text : icon.nome,
		top : 50,
		textAlign : 'center',
	})
	view.add(icona);
	view.add(label);
	view.addEventListener('click', function() {
		Ti.API.info('click su ' + icon.nome);
		iconWin = icon.func();
		iconWin.navBarHidden = false;
		navGroup.open(iconWin, {
			animated : true
		});
		AppGio.ui.winOpened = iconWin;
	})
	return view;
};
```

Ne risulta che per ogni invocazione di *createIcon(icon)* venga creata una vista (una *view*) con l'icona scelta per ogni data finestra. Tutti i parametri di ogni icona sono memorizzati nell'array *list*; tra questi c'è anche il nome della funzione che viene invocata quando un'icona (una vista contenente l'icona) viene cliccata (generando quindi l'evento gestito dall'*eventListener* della *view*). Quando una vista viene cliccata, il risultato è allora l'invocazione di una delle "funzioni principali". Ogni vista principale restituisce come output un oggetto *window*, che viene aperta ed associata al *navigationGroup*.

Il codice necessario per la creazione della finestra principale e per la gestione di tutte le altre finestre è concluso. Le funzioni per la creazione della finestra dovranno avere una struttura di questo tipo:

```
AppGio.ui.createEventWindow = function() {
	Ti.API.info('function AppGio.ui.createEventWindow');
	var win = Ti.UI.createWindow({
		title : 'evento rapido',
		//
		backgroundImage : AppGio.ui.bgImageView,
	});
	//
	//Corpo della window, con altri elementi UI
	//e eventLister
	//
	return win;
};
```

### *Evento Rapido*

La finestra mostrata in figura 23 che permette di inserire un nuovo tipo di evento. I tipi di evento vengono mantenuti in una *table view* scorribile che si aggiorna ogni volta che viene modificata la tabella *Tipo Evento* nel database. Da questa finestra si può anche accedere alla finestra che permette l'inserimento di un nuovo tipo di evnto.
In figura 24 è mostrato l'accesso alla finestra inserimento evento (evento rapido) selezionando un tipo di evento dalla tabella della finestra.

**Figura 24:** A sinistra la finestra *Evento Rapido*; a destra la finestra per l'inserimento di un evento del tipo selezionato. E' possibile modificare le ore dell'evento e la data (impostata sull'ora corrente).

### *Statistiche*

La finestra è mostrata in figura 25. Gli eventi sono raggruppati per tipo; e ogni gruppo ha un header con un colore che varia a seconda che i vincoli siano rispettati o meno (rosso: scaduto; giallo: in scadenza; verde: completato; grigio: nessun vincolo). I tipi di eventi della tabella sono indicizzati per nome ed è possibile effettuare una ricerca con l'apposita barra. Un algoritmo calcola anche la percentuale di avanzamento di ogni *Tipo Evento* e permette la

visualizzazione di questo dato come barra di avanzamento all'interno dell'header del gruppo corrispondente.

È anche possibile gestire la *tableRow* (oggetto tabella, contenente i riferimenti agli eventi) in modalità "*editable*". In questa modalità è possibile eliminare direttamente da qui gli eventi, e verificare la variazione delle statistiche. I cambiamenti vengono propagati anche al database, ma non al calendario nativo; questa è stata una scelta progettuale.

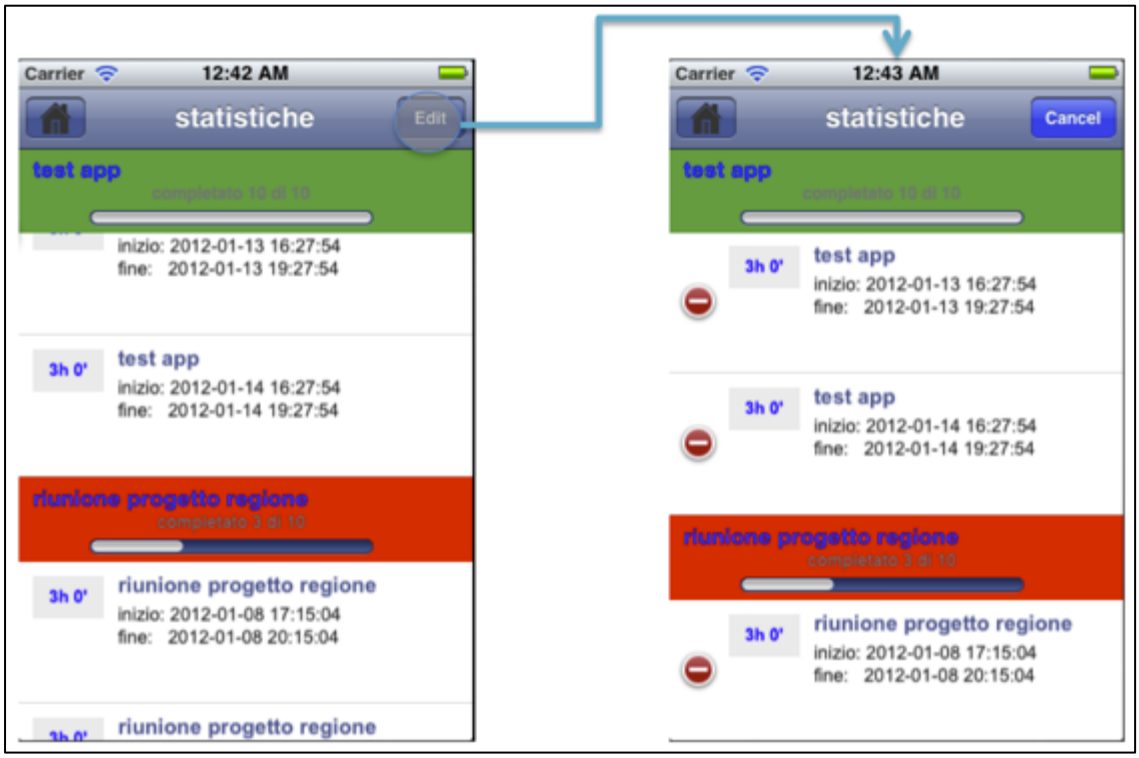



**Figura 25:** Finestra *Statistiche*. A destra in modalità *modifica*, con la quale si possono modificare/cancellare gli venti del database.

***Tipi di Evento***

Con la finestra *Tipo di Evento*, figura 26, è possibile modificare i vincoli di un tipo di evento.

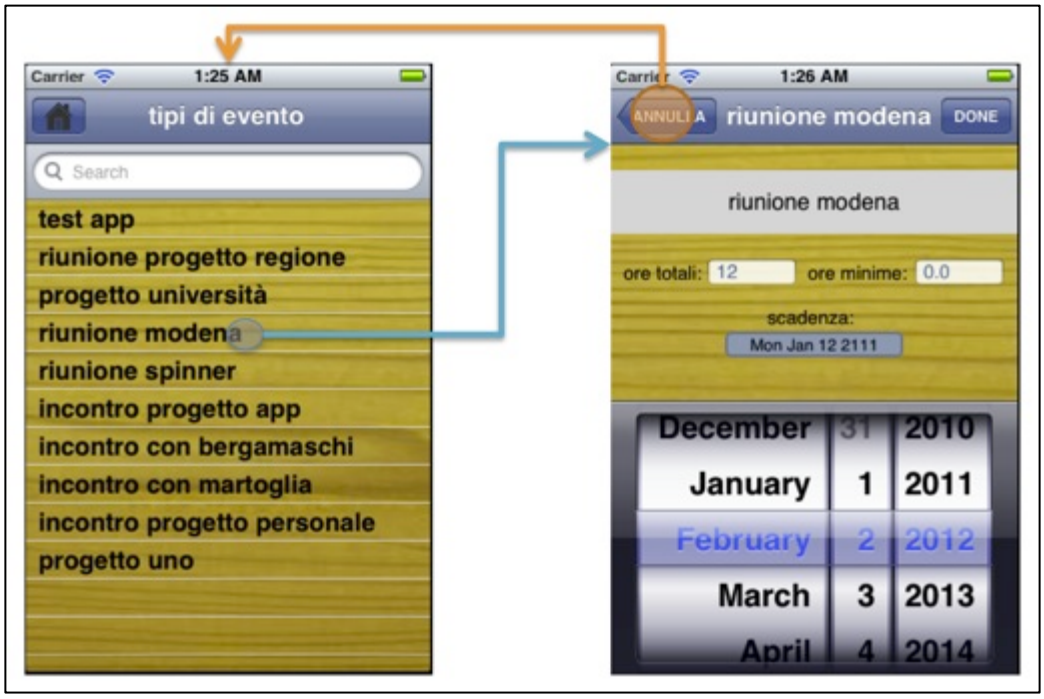



**Figura 26:** Finestra *Tipo di Evento* (sinistra) e *Modifica Tipo di Evento* (destra).

***Import***

La gestione degli import avviene da questa finestra (figura 27). Nel *text-field* devono essere inserite le parole da ricercare nei calendari di iCal. Dinamicamente vengono ricercati tutti gli eventi che contengono quelle parole, se più eventi hanno titolo uguale viene mostrato solo una volta. Viene così generato un elenco di titoli di eventi presenti in iCal. Quelli selezionati vengono gestiti dall'app con l'apposita funzione di libreria (*gestisciImport()* ).

**Figura 27:** Sequenza delle azioni per importare eventi da iCal nel database dell'app. Da sinistra: nella prima finestra vengono inserite le parole da ricercare e sono spuntati i titoli degli eventi da ricercare in iCal (titoli di iCal, non di *Tipo Evento*); nella seconda finestra è selezionato il tipo di evento nel quale vengono fatti confluire questi eventi; nell'ultima vengono mostrati gli effetti sulle statistiche (prima dell'operazione descritta "progetto università" era vuoto).

Di seguito è mostrato l'algoritmo che permette la ricerca dinamica dei titoli da calendario. Da notare come i titoli che sono già stati ritrovati non vengano ripetuti nella *tableRow*; in questo modo vengono riconosciute "classi di evento" direttamente nel calendario nativo. Questo facilita molto sia l'importazione di eventi simili (appartenenti alla stessa classe) da parte della funzione *gestisciImport*, sia rende più facile l'organizzazione degli eventi all'utente.

```
textField.addEventListener('change', function() {
	data = [];
	if(textField.velue == null) {
		tableView.fireEvent('aggiorna');
	}
	titoli = [];tmp = [];parole = [];

	tmp = textField.getValue().split(' ');
	for( i = 0; i < tmp.length; i++) {
		if(tmp[i] != "") {
			parole.push(tmp[i]);
		}
	}
	if(parole.length > 0) {
		keys = AppCal.cal.trova_evento({
			start : ss,
			end : ee,
			title : 'quelsiasi tanto non lo usa',
			parole : parole,
			in_title : true
		});
		for(key in keys) {
			trovato = false;
			for( i = 0; i < titoli.length; i++) {
				if(JSON.parse(keys[key]).title == titoli[i]) {
					trovato = true;
				}
			}
			if(!trovato) {
				titoli.push(JSON.parse(keys[key]).title);
			}
		};
		Ti.API.info('titoli: ' + titoli.length);
		for(t in titoli) {
			checks++;
			data.push({
				title : titoli[t],
				hasCheck : true,
			});
		}
		tableView.fireEvent('aggiorna');
	}
});
textField.addEventListener('return', function() {
	parole = textField.getValue().split(' ');
});
```

## Conclusioni e Sviluppi Futuri

Durante il periodo di tesi è stata progettata e sviluppata un'applicazione per dispositivi mobili basati sul sistema operativo iOS per la gestione e pianificazione di attività: CalPlanner. Sono state seguite tutte le fasi che hanno portato alla realizzazione di un app funzionante, dal progetto (definizione delle funzionalità, requisiti, scelta delle tecnologie e degli strumenti, etc.) all'implementazione (database, librerie e moduli JavaScript, Interfaccia Utente, etc.).

CalPlanner non ha l'intento di sostituirsi a quelli che l'utente già utilizza per la memorizzazione dei propri eventi (es: iCal, Google Calendar, etc.), ma di integrarsi con loro. Lo scopo dell'app è di affiancare l'utente nella pianificazione di attività. Gli eventi sono importati da calendari già esistenti, ma possono essere creati anche con una funzionalità dell'app, e classificati secondo le esigenze dell'utente che definisce classi di evento. Ogni classe di evento mantiene aggiornati degli attributi (es: ore totali, ore minime) e dei vincoli (es: scadenza) che vengono utilizzati per costruire viste schematiche e riassuntive di tutti gli eventi inseriti suddivisi per classi. L'app vuole anche offrire strumenti per la pianificazione di attività future: imponendo vincoli (scadenze, ore per incontro, ore totali, etc.) viene ritrovata (se esiste) una serie di disponibilità temporali coerenti con gli eventi già presenti sul calendario.

L'intera applicazione è stata sviluppata in Titanium, un framework che permette di scrivere applicazioni native per dispositivi mobili con linguaggi tipici del web dinamico (JavaScript, HTML e CSS), consentendo il deploy di tali app sia su piattaforma iOS che Android. In questo progetto è stata implementata solo l'applicazione per iOS, ma utilizzando gli stessi strumenti e le stesse librerie è possibile adattare l'app per Android.

Titanium si è rivelato uno strumento molto potente e versatile per applicazioni di tipo *utility*. Il suo punto di forza è indubbiamente la riduzione del lavoro e dei tempi di sviluppo rispetto agli strumenti classici (iOS SDK ed Android SDK), grazie all'impiego di un linguaggio semplice e dinamico (JavaScript) ed APIs di alto livello.

Ha però ancora limiti, ad esempio per l'interazione dell'app con il calendario nativo si è dovuto impiegare un modulo aggiuntivo open-source, sviluppato in linguaggio nativo per iOS. Tuttavia, nel complesso, risulta completo: ciò che non fornisce come APIs di alto livello è comunque integrabile con moduli nativi.

**Sviluppi Futuri**

L'app fornisce funzionalità che le permettono di essere utilizzata per la pianificazione e gestione di eventi. Per come è stata concepita e strutturata il suo naturale sviluppo sarebbe quello di includerne di nuove, soprattutto orientate ad un utilizzo multiutente (*social working*) ed ad un contesto aziendale (pianificazione e gestone di attività di vendite)

***Funzionalità per Social Working***

All'interno del database dell'app confluiscono dati relativi ad eventi, anche appartenenti a calendari differenti (es: lavoro, casa, GoogleCalendar etc.), quindi è possibile ricostruire una struttura dati che rappresenti l'organizzazione del tempo da parte dell'utente, con particolare riferimento alla creazione di liste di slot temporali "liberi", utilizzabili (una struttura di oggetti "*disponibilità*"). Una funzionalità aggiuntiva potrebbe permettere all'utente di impostare *profili* preferenziali di fasce orarie, come ad esempio:

- Orario lavoro : Dal Lunedì al Venerdì; dalle 8:30 alle 12:30 e dalle 14:00 alle 19:00
- Orario riunioni in sede: Lunedì e Giovedì; dalle 14:00 alle 19:00
- Qualsiasi altro profilo che si adegui alle esigenze del settore di impiego.

Con l'impiego della struttura dati delle *disponibilità* e *profili* scelti volta per volta dall'utente (memorizzabili o costruibili dinamicamente) si può estendere la funzionalità **proiezione** già presente nell'app. Questo avrebbe un duplice obiettivo:

I. migliorare l'usabilità dell'applicazione stand alone con il maggior numero di vincoli che rappresentano le esigenze dell'utente e che possono variare da ricerca a ricerca.
II. offrire lo strumento base per l'implementazione di funzionalità per *"social workin"*: utilizzo dell'applicazione in un contesto multiutente.

Con "*social working"* si intende l'utilizzo di applicazioni per la pianificazione e gestione di attività in contesto multiutente. Lo scenario di utilizzo è ad esempio quello di un gruppo di utenti vuole pianificare una serie di attività nel tempo con determinati vincoli (vincoli dell'attività); ogni utente ha le proprie esigenze (vincoli di utente); l'applicazione (utilizzata da ogni utente) deve interagire con le altre in modo da ritrovare un piano di allocazione temporale che soddisfi tutti i vincoli (per ogni utente e per l'attività).

Per fare questo è necessario dal punto di vista concettuale anche l'introduzione di meccanismi per la rappresentazione della priorità degli eventi e degli utenti:

- Per ogni utente ci deve essere una scala di priorità per gli eventi. Ad esempio se un evento già esistente per un utente può essere spostato senza problemi e ciò determinerebbe una soluzione migliore per tutti gli utenti, deve essere data la possibilità di farlo.
- Per l'attività che si deve pianificare deve esistere una scala di priorità per gli utenti, per la quale il soddisfacimento di vincoli per alcuni utenti deve essere più importate di quello di altri.

Dal punto di vista implementativo sono invece necessari meccanismi di comunicazione per la condivisione di dati. Questo richiede l'implementazione di moduli per lo scambio di dati tramite email, sms o server intermedio. Titanium mette a disposizione APIs per l'invio e ricezione di sms, mail e gestione di socket (per la comunicazione con il server eventualmente).

***Inserimento in un Contesto Aziendale***

L'app è basata sulla definizione di classi di tipi di eventi caratterizzati da vincoli e sulle quali vengono mantenute statistiche. Questo modello potrebbe essere adottato come strumento di supporto per agenti di vendita. Spesso, infatti, gli agenti di vendita di un'azienda operano secondo schemi basati su **indicatori di comportamento**, come: numero di visite a clienti esistenti, numero di visite a clienti nuovi, numero di visite a clienti superiori (o inferiori) ad una certa dimensione, etc.. Aggiungendo la possibilità di mantenimento di statistiche di questo l'app consentirebbe all'agente di conoscere il livello di avanzamento della sua attività in relazione a vincoli prefissati dalla struttura di incentivi operate dall'impresa.

Mantenendo **indicatori di risultato** (ad es.: raggiungimento di un budget di vendita predeterminato) l'app può guidare l'agente nella pianificazione di attività basata sulle passate esperienze dell'impresa in quell'attività. Se ad esempio, basandosi sui dati raccolti, si è notato in passato una relazione causale positiva tra un certo tipo di interventi (es: numero di visite, numero di ore visite, etc.) ed i risultati ottenuti (valori e volumi di vendita di quelle attività), allora l'app potrebbe supportare l'agente nella pianificazione di un piano di intervento basato sul modello derivante dalla correlazione **indicatori di comportamento–indicatori di risultato**. Introducendo inoltre la possibilità di georeferenziamento dei clienti, organizzandoli

per classi coerenti con gli obiettivi, sarebbe possibile migliorare ulteriormente la pianificazione degli interventi considerando anche la prossimità spaziale dei clienti. Titanium offre APIs di alto livello per l'integrazione di funzionalità di geolocalizzazione e anche per l'interazione con GoogleMaps.

Anche in questo contesto si può immaginare un utilizzo multiutente. L'impresa potrebbe utilizzare l'app come supporto per la programmazione di attività dei singoli venditori, sia come monitoraggio in tempo reale a disposizione dell'impresa sull'operato degli agenti/venditori al fine di attivare o aggiungere interventi in tempi rapidi basato sugli indicatori di tutti gli agenti.

***Altre Piattaforme***

Infine l'applicazione è stata sviluppata in Appcelerator Titanium sia per la sua semplicità, sia perché permette di sviluppare con un unico strumento per più piattaforme mobili contemporaneamente. Per ora l'applicazione è ottimizzata per iPhone ed iPod Touch, ma può essere facilmente adattata ad iPad, modificando i parametri di configurazione dell'interfaccia utente. Per la portabilità in Android il passaggio non è immediato, infatti bisogna riadattare alcune funzioni e molte componenti dell'interfaccia grafica al diverso sistema operativo. Questo perché Titanium mette a disposizione le APIs di alto livello che svolgono la funzione di ponte con quelle di basso livello delle piattaforme mobili, che però non sono uguali tra di loro. Ciò significa che per tutte le componenti di iOS sviluppate per le quali non esiste il corrispettivo in Android, bisogna riprogettare il modulo o la funzione; sempre in Titanium, però, conservando tutto il codice di controllo JavaScript che rimane uguale. Anche le funzioni che implementano la logica dell'applicazione rimangono uguali, come non viene modificato il database e le funzioni per la sua gestione. I cambiamenti da apportare riguarderebbero soprattutto la parte grafica, perché è in quella che le APIs si differenziano maggiormente; mentre per quanto riguarda la comunicazione con il calendario nativo, per Android sono già presenti APIs Titanium, non ci sarebbe la necessità di ricostruire un apposito modulo nativo.

## Appendice A: Utilizzo dell'ereditarietà in JavaScript

Le 2 funzioni alla base del metodo sono *eredita* e *estendi*. L'attributo "*sup*" permette di richiamare i metodi della classe padre.

```
Object.prototype.eredita = function(padre) {
	padre.apply(this, Array.prototype.slice.call(arguments, 1));
}

Function.prototype.estendi = function(padre) {
	var costruttore = this.constructor;

	var sup = new padre();

	this.prototype = sup;
	this.prototype.constructor = costruttore;
	this.prototype.sup = sup;
};
```

Un esempio di utilizzo in JavaScript è il seguente:

Classe *persona*:

```
var persona = function(nome) {
	var altezza_padre = 1.70;
	this.nome = nome;
	this.cognome = "Rossi";

	this.saluta = function() {
		Ti.API.info("Ciao! sono " + this.nome + "  " + this.cognome + " - Persona");
	};
	this.altezza = function() {
		Ti.API.info("Altezza: " + altezza_padre);
	};
};
```

Classe *padre*, eredita da *persona*:

```
var padre = function(nome) {
	var altezza_padre = 1.80;
	//EREDITA
	this.eredita(persona, nome);
	this.cognome = "Simonini";
	this.altezza = function() {
		Ti.API.info("Altezza padre: " + altezza_padre + " - funzione_padre");
	};
	this.saluta = function() {
		Ti.API.info("Ciao! sono " + this.nome + "  " + this.cognome + " -
funzione_padre");
	}
};
//ESTENDI
```

Classe *figlio*, eredita da *padre*:

```
padre.estendi(persona);
//
var figlio = function(nome, altezza) {
	this.altezza = altezza;
	//EREDITA
	this.eredita(padre, nome);
	this.altezza = function() {
		Ti.API.info("Altezza: " + altezza+ " - funzione_figlio");
	};
	this.info = function(){
		Ti.API.info('Sono un oggetto Figlio');
	}
};
//ESTENDI
figlio.estendi(padre);
```

Esempio di utilizzo di oggetti creati con le classi precedenti:

```
var oggetto_persona = new persona("persona");
oggetto_persona.saluta();// Ciao! sono persona  Rossi - Persona

var oggetto_padre = new padre("Stef");
oggetto_padre.saluta();// Ciao! sono Stef  Simonini - funzione_padre
oggetto_padre.altezza();// Altezza padre: 1.80 - funzione_padre
//oggetto_padre.info();// !Questa funzione non esiste!

var oggetto_figlio = new figlio("John", 1.83);
oggetto_figlio.saluta();// Ciao! sono John  Simonini - funzione_padre
oggetto_figlio.altezza();// Altezza: 1.83 - funzione_figlio
oggetto_figlio.sup.altezza();// Altezza padre: 1.80 - funzione_padre
oggetto_figlio.info();// Sono un oggetto Figlio
```

# Bibliografia

[ 1 ] www.gartner.com/technology/home.jsp

[ 2 ] en.wikipedia.org/wiki/Mobile_operating_system

[3] techcrunch.com/2009/06/08/40-million-iphones-and-ipod-touches-and-50000-apps/

[4] events.apple.com.edgesuite.net/11piuhbvdlbkvoih10/event/index.html]

[5] www.fonehome.co.uk/2010/07/16/android-market-hits-1-billion-downloads-100000-apps/

[6] www.wired.com/gadgetlab/2011/12/10-billion-apps-detailed/

[7] www.iphoneitalia.com/apple-chiudera-il-2011-a-3-miliardi-di-dollari-di-fatturato-proveniente-solo-da-appstore-android-a-425-milioni-news-237896.html/ihs-screen-digest-research-201105-app-store-market

[8] www.ihs.com/

[ 9 ] wiki.appcelerator.org/display/guides/iOS+Platform+Overview

[ 10 ] www.coolpage.com/developer/javascript/Correct OOP for Javascript.html

[ 11 ] developer.appcelerator.com

[12]developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Principles/Principles.html#//apple_ref/doc/uid/TP40006556-CH5-SW1

[13] www.sqlite.org/lang_datefunc.html

[14] www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

[15] github.com/stelford/Titanium-Calendar

[16] wiki.appcelerator.org/display/guides/iOS+Module+Development+Guide

[17]developer.apple.com/library/IOs/#documentation/Cocoa/Reference/Foundation/Classes/NSDictionary_Class/Reference/Reference.htm

[18] wiki.appcelerator.org/display/guides/Mobile+Best+Practices

[19] docstore.mik.ua/orelly/webprog/jscript/ch11_03.htm

[20] www.sqlite.org/lang.html

[21] www.sqlabs.com/sqlitemanager_features.php

[ 22 ] wiki.appcelerator.org

[ 23 ] code.google.com/p/datejs/wiki/APIDocumentation

[24] docs.jquery.com/Main_Page