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 dell'Applicazione Web Share Mobile per Dispositivi Mobili Multipiattaforma

Relatore:

Chiar.mo Prof. Sonia Bergamaschi

Correlatore:

Prof. Riccardo Martoglia

Correlatore Esterno:

Ing. Andrea Prandini

Elaborato di Laurea:

Marco Stanzani

Anno Accademico 2010/2011

INDICE

Introduzione **4**

Parte I: Strumenti Usati **6**

Capitolo 1. Stato dell'arte dei dispositivi mobili e dei loro software **7**

1.1. Dispositivi Mobili **7**

1.1.1. Smartphone 7

1.1.2. Tablet Computer 8

1.1.3. Sistema Operativo per dispositivi mobili 10

1.2. Software Development Kit **11**

1.3. Android **12**

1.3.1. Sviluppo 12

1.3.2. Le Applicazioni 13

1.3.3. La Dalvik Virtual Machine 13

1.3.4. Creazione, compilazione, emulazione 14

1.3.5. Confronto con iOS di Apple 15

1.4. IOS Apple **15**

1.4.1. Caratteristiche 16

1.4.2. Sviluppo 17

1.5. SQLite **18**

1.5.1. Vantaggi e Svantaggi di SQLite 18

1.6. JavaScript Object Notation **20**

1.7. JQuery Mobile **21**

1.7.1. Piattaforme supportate 21

1.7.2. Anatomia di una pagina 22

1.7.3. Collegamento a una pagina esterna 23

1.7.4. Collegamento a una pagina interna 24

1.7.5. Aggiornamento automatico dei titoli delle pagine 25

1.7.6. Creare un collegamento alla pagina chiamante. 25

1.7.7. Conclusioni 25

Capitolo 2. Appcelerator Titanium **27**

2.1. Sviluppo di un'applicazione **28**

2.2. Struttura di Titanium **28**
2.3. Concetti sul design di Titanium **29**
2.3.1. Windows 30
2.3.2. Views 31
2.3.3. Widgets 32
2.4. Struttura di un progetto **33**
2.5. Configurazione dell'applicazione **34**

Parte II: Descrizione dell'Applicazione Web Share Mobile **35**

Capitolo 3. Progettazione dell'applicazione Web Share Mobile **36**
3.1. Web Share Web Application **36**
3.1.1. Gestione degli utenti 37
3.1.2. Schema E-R 38
3.2. Professional User Manager **39**
3.2.1. Schema E-R 40
3.3. Inquadramento dell'applicativo WS Mobile **40**
3.4. Requisiti Funzionali **41**
3.4.1. Lato Server 41
3.4.2. Lato Client 43
3.5. Requisiti minimi richiesti da Android e Apple **48**
3.5.1. Caratteristiche richieste da Android 48
3.5.2. Caratteristiche richieste da Apple 49
3.6. Diagramma Use Case generale **50**

Parte III: Realizzazione del progetto **51**

Capitolo 4. Implementazione dei moduli lato Server **52**
4.1. Autenticazione dell'Utente **52**
4.2. Creazione e invio dell'oggetto JSON **54**
4.2.1. Acquisizione delle informazioni 54
4.3. Creazione del oggetto JSON **55**
4.4. Invio del file **56**

Capitolo 5. Implementazione dell'App Web Share Mobile **58**
5.1. Struttura del database e la sua connessione al progetto **58**

5.1.1. Descrizione delle tabelle del database 58
5.1.2. Connessione al progetto 60
5.2. Impostazioni dell'Applicativo 62
5.2.1. Sviluppo della pagina delle Impostazioni per iOS 62
5.2.2. Sviluppo della pagina delle Impostazioni per Android 65
5.2.3. Gestione dei dati con Titanium 67
5.3. Caratteristiche generali di Web Share Mobile 69
5.3.1. Struttura dell'interfaccia grafica 69
5.3.2. Implementazione della GUI su dispositivi iOS 71
5.3.3. Implementazione della GUI su dispositivi Android 73
5.3.4. Riempimento delle tabelle 75
5.4. Gestione delle informazioni dei documenti 77
5.4.1. Acquisizione dell'oggetto JSON 77
5.4.2. Aggiornamento delle informazioni nel DB 78
5.4.3. Cancellazione delle informazioni e dei relativi file in eccesso 79
5.5. Gestione dei File 81
5.5.1. Download del File 81
5.5.2. Visualizzazione del documento 83
5.6. Invio di un File tramite Email 85

Conclusioni 87

Sviluppi futuri 88

Glossario 89

Bibliografia 90

Ringraziamenti 91

Introduzione

Nel corso dei ultimi anni la grande diffusione di smartphone e tablet ha portato una parte dell'informatica a focalizzarsi sullo sviluppo di applicazioni per questi dispositivi. I programmatori quindi si sono dovuti scontrare su diverse problematiche come: l'enorme differenza di prestazioni fra questi dispositivi e i personal computer; diversi sistemi operativi mobili che permettono di sviluppare nuove App solamente mediante delle proprie SDK; eterogeneità hardware fra i vari dispositivi anche nel caso in cui sfruttano un sistema operativo identico (risoluzione del display differente, presenza o meno di un monitor touch screen ecc..).

Tutte queste problematiche hanno portato le software house, che vogliono intraprendere lo sviluppo di applicativi per smartphone e tablet, a fare diverse scelte ovvero: sviluppare per una sola tipologia di piattaforma, oppure creare diversi team di sviluppo ognuno dei quali specializzato sulla programmazione su un tipo di sistema operativo mobile, oppure usare delle tecnologie innovative che permettono di sviluppare un App per più piattaforme.

All'interno di questo elaborato verrà mostrato lo sviluppo, per dispositivi Apple e Android, di un'applicazione chiamata Web Share Mobile usando il Framework Appcelerator Titanium. La scelta è ricaduta su questo Framework perché è uno dei primi che permette di creare applicazioni native per i vari dispositivi che supporta, partendo da un codice comune scritto in Java Script.

Viene ora esposta la struttura secondo la quale è organizzato il seguente elaborato di laurea.

Nel primo capitolo verrà effettuata una panoramica generale sui dispositivi mobili e sui sistemi operativi Android e iOS; inoltre ci saranno anche alcuni accenni su SQL Mobile e JQuery Mobile, tecnologia usata per lo sviluppo di una seconda App.
Nel secondo capitolo viene descritto nella sua interezza il Framework Appcelerator Titanium.
Nel quarto capitolo verrà descritto in generale l'applicativo in modo da chiarire i suoi requisiti funzionali mediante anche l'uso di diagrammi.
Il quinto e il sesto capitolo descrivono minuziosamente l'implementazione del progetto mostrando anche parti di codice. In particolare nel quinto capitolo si descrive l'applicazione lato Server, nell'ultimo capitolo invece l'applicazione lato Client contenuta all'interno dello smartphone o del tablet.

Parte I: Strumenti Usati

Capitolo 1.

Stato dell'arte dei dispositivi mobili e dei loro software

1.1. Dispositivi Mobili

Un **dispositivo mobile** (anche conosciuto come un **dispositivo portatile**, **palmare**) è semplicemente un dispositivo di piccole dimensioni che implementa funzionalità uguali o simili a quelle dei personal computer.

A causa delle loro dimensioni ridotte la comunicazione tra utente e dispositivo avviene tramite monitor touchscreen e/o mini tastiere; inoltre possiedono caratteristiche Hardware e prestazionali inferiori rispetto i personal computer, presupposto significativo da considerare durante la fase di sviluppo di un software.

1.1.1. Smartphone

Uno **Smartphone** è un dispositivo portatile che abbina funzionalità di telefono cellulare a quelle di gestione dei dati personali, può derivare dall'evoluzione di un palmare a cui si aggiungono funzioni di telefono o viceversa, di un telefono mobile a cui si aggiungono funzioni di palmare [1].

La caratteristica più interessante degli **Smartphone** è la possibilità di installarvi ulteriori applicazioni (**App**), che incrementano le funzionalità del dispositivo. Queste App possono essere sviluppate dal produttore del telefono, dallo stesso utilizzatore, o da terze parti.

Attualmente gli **Smartphone** vengono venduti con diversi accessori hardware come fotocamera, GPS, componenti che permettono di connettere il dispositivo con il Web

[1] **Wikipedia**, Smartphone. Disponibile su http://it.wikipedia.org/wiki/Smartphone.

(connessioni GSM, GPRS, EDGE, UMTS, HSDPA e HSUPA), piuttosto che con tecnologie come il Bluetooh e il Wi-Fi che permettono la comunicazione con altri apparecchi. Inoltre vengono anche preinstallati programmi per l'accesso a internet, la gestione delle e-mail, la pianificazione delle attività e per la compatibilità con i più comuni formati di file come **PDF** e quelli della suite Microsoft Office.

1.1.2. Tablet Computer

Un "**Tablet computer**", o semplicemente "**Tablet**", è un computer di dimensioni compatte che come principale sistema di input utilizza uno schermo touchscreen controllato da una penna o tramite le dita.

Il nome deriva dalla forma del dispositivo che ricorda una tavoletta utilizzata per la scrittura. Questi dispositivi non sono dotati di tastiera sebbene questa spesso possa essere simulata sullo schermo [2].

Nel 2010 Apple ha presentato l'iPad un Tablet computer che ha rilanciato il mercato di questi dispositivi difatti le vendite nel 2010 oscillano intorno ai 19,5 milioni.

Una significativa differenza tra un Tablet e un personal computer è la capacità di un utente di sviluppare e installare applicazioni senza la necessità di far approvare le suddette da un ente esterno. La nuova classe di Tablet computer come l'iPad presentano un approccio definito "giardino murato". Il dispositivo tramite restrizioni tecnologiche impedisce all'utente l'installazione di applicazioni arbitrarie, le applicazioni per poter essere installate sul dispositivo devono essere approvate dal produttore, questo rende più difficile l'installazione di applicazioni malevoli ma nel contempo limita la libertà dell'utente. I rappresentati del movimento open source criticano questo approccio ritenendolo troppo limitante per l'utente e ritengono che dispositivi che adottano questo approccio non possano essere equiparati a dei personal computer. Tuttavia alcuni sistemi operativi per Tablet computer non sono vincolati dal giardino murato e di conseguenza più simili ai personal computer da questo punto di vista.

[2] **Wikipedia**, Tablet Computer. Disponibile su http://it.wikipedia.org/wiki/Tablet_computer.

Vantaggi e Svantaggi dei Tablet computer

Gli svantaggi e i vantaggi di un Tablet computer sono altamente soggettivi, quelli che per alcuni sono dei vantaggi per altri potrebbero essere degli svantaggi, il seguente elenco mostra le voci più comuni

Vantaggi

- Può essere utilizzato anche in ambiti scomodi per un dispositivo dotato di tastiera e mouse, per esempio in piedi, a letto.
- Leggerezza, i modelli più leggeri possono essere utilizzati con poco sforzo anche come e-reader.
- Lo schermo sensibile al tocco permette lo sviluppo di programmi come quelli musicali o grafici con un livello di interazione e di immediatezza di gestione non raggiungibile con un mouse.
- Un dispositivo Tablet permette di realizzare disegni o schemi come su un foglio di carta, cosa non possibile con la stessa naturalezza utilizzando un mouse.
- Alcuni utenti trovano più intuitivo premere sull'immagine che rappresenta un oggetto piuttosto che guidare un dispositivo di puntamento come il mouse per selezionare un oggetto.

Svantaggi

- Costo, uno schermo sensibile al tocco costa mediamente di più di una tastiera e un mouse e difatti a parità di prestazioni i Tablet costano più dei personal computer.
- La digitazione di un testo su uno schermo è mediamente più lenta della digitazione su una tastiera fisica, sebbene alcuni programmi possano ridurre il divario.
- Il dispositivo non è pensato per lunghe digitazioni dato che non prevede uno spazio dove appoggiare i polsi durante il suo uso.
- La potenza di calcolo di questo tipo di dispositivi è generalmente limitata.

1.1.3. Sistema Operativo per dispositivi mobili

Un **sistema operativo per dispositivi mobili**, in inglese "**mobile OS**", è un sistema operativo che controlla un dispositivo mobile (Smartphone o Tablet) con lo stesso principio con cui **Mac OS**, **Linux** o **Windows** controllano un desktop computer oppure un laptop. Tuttavia affronta problematiche legate alla natura del dispositivo mobile, più critiche rispetto ad un desktop o a un laptop; tra le tante: la limitatezza delle risorse(memoria, CPU), l'assenza di alimentazione esterna, differenti protocolli di trasferimento dati per l'accesso a internet (WiFi, GPRS, HSDPA,...), nuovi metodi d'immissione(touchscreen, minitastiere) e ridotte dimensioni del display [3].

Comparazione con i principali Sistemi operativi per dispositivi mobili

Nella Tabella 1.1 vengono elencati i principali sistemi operativi per dispositivi mobili con le rispettive informazioni.

Feature	iOS	Android	Windows Mobile	Windows Phone 7	BlackBerry OS	Symbian
Produttore	Apple	Open Handset Alliance(Google)	Microsoft	Microsoft	RIM	Symbian Foundation
Versione Corrente	5	2.3.4	6.5.5	7.0.7008.0	6.0.0	9.5
Famiglia di Sistemi Operativi	Mac OS X/Unix-like	Linux	Windows CE 5.2	Windows CE 7	Mobile OS	Mobile OS
Architettura CPU Supportata	ARM	ARM, MIPS, Power Architecture, x86	ARM	ARM	ARM	ARM, x86
Linguaggio di programmazione	C, C++, Objective C	C, C++, Java	C++	C++	Java	C++
Application Store	App Store	Android Marketplace	Windows Marketplace for Mobile	Windows Phone Marketplace	App World	Symbian Horizon
Sincronizzatori e gestione dispositivo	iTunes	APK	Windows Mobile Device Center/ActiveSync	Software Zune	Blackberry Desktop Manager	Nokia Ovi Suite

Tabella 1.1: Principali Sistemi operativi mobili

Nei prossimi capitoli verranno descritti i due sistemi operativi mobili più diffusi, Android e iOS, i quali sono installati sia su Smartphone che su Tablet.

[3] **Wikipedia**, Mobile Operating System. Disponibile su http://en.wikipedia.org/wiki/Mobile_OS.

1.2. Software Development Kit

Software Development Kit (più brevemente **SDK**) è un termine che in italiano si può tradurre come "pacchetto di sviluppo per applicazioni", e sta a indicare un insieme di strumenti per lo sviluppo e la documentazione di software [4].

Gli SDK possono variare dimensioni e tecnologie utilizzate considerevolmente, ma tutti possiedono alcuni strumenti fondamentali:

- *un compilatore*, per tradurre il codice sorgente in un eseguibile;
- *librerie standard* dotate di interfacce pubbliche dette API - Application Programming Interface;
- *documentazione sul linguaggio di programmazione* per il quale l'SDK è stato sviluppato e sugli strumenti a disposizione nell'SDK stesso;
- *informazioni sulle licenze da utilizzare* per distribuire programmi creati con l'SDK.

Questo corredo di base può essere esteso con strumenti di vario tipo:

- *compilatori* per diversi linguaggi di programmazione;
- *programmi per la gestione delle versioni* o per interfacciarsi a essi:
- *editor di sorgenti*: programmi che permettono di scrivere codice più velocemente o con più accuratezza, molto spesso hanno basilari funzioni di Syntax Highlighting (evidenziazione della sintassi) o di validazione;
- *IDE*: editor di sorgenti con funzioni molto avanzate. Di seguito una breve lista delle funzionalità che è possibile trovare in un IDE:
    - completamento automatico del codice sorgente, creazione automatica di pezzi di codice e formattatori automatici;
    - possibilità di "disegnare" l'interfaccia grafica di un'applicazione in modo visuale;
    - strumenti per l'analisi del codice: risoluzione delle dipendenze, analisi statistica, ricerche;
    - strumenti per creare e mantenere la documentazione relativa al codice sorgente;
    - strumenti per il Refactoring

Molti SDK sono disponibili gratuitamente e possono essere prelevati direttamente dal sito del produttore: in questo modo si cerca di invogliare i programmatori ad utilizzare

[4] **Wikipedia**, SDK. Disponibile su http://it.wikipedia.org/wiki/Software_development_kit.

un determinato linguaggio o sistema. Vi è anche un utilizzo orientato al mercato: alcuni programmi vengono venduti assieme al loro SDK permettendo ai acquirenti di sviluppare ulteriori parti del programma acquistato.
I fornitori di SDK per specifici sistemi inoltre possono offrire delle versioni più specifiche: sia Microsoft che la Apple Computer ad esempio prevedono SDK per lo sviluppo per computer piuttosto che per dispositivi mobili.

1.3. Android

Android è un sistema operativo open source inizialmente sviluppato da Android Inc. e successivamente acquistato da Google nel 2005.
La piattaforma è basata sul **kernel Linux**, usa il database **SQLite**, la libreria dedicata **SGL** per la grafica bidimensionale e supporta lo standard **OpenGL ES 2.0** per la grafica tridimensionale. Le applicazioni vengono eseguite tramite la **Dalvik Virtual Machine** che è a tutti gli effetti una Java virtual machine adattata ai dispositivi mobili. Questa macchina virtuale è ottimizzata per sfruttare la poca memoria presente nei dispositivi mobili, consente inoltre di utilizzare diverse istanze della macchina virtuale contemporaneamente e nasconde al sistema operativo sottostante la gestione della memoria e dei thread [5].
Android è fornito di una serie di applicazioni preinstallate: un browser (basato su WebKit), una rubrica e un calendario.

1.3.1. Sviluppo

Lo sviluppo di un'applicazione per il sistema operativo Android viene effettuata mediante il software development kit (**SDK**) che include: gli strumenti di sviluppo, le librerie, un emulatore del dispositivo, la documentazione (in inglese), alcuni progetti di esempio, tutorial e altro. È installabile su qualsiasi computer x86 compatibile che usi come sistema operativo Windows, Mac OS X o Linux. L'IDE ufficialmente supportato per lo sviluppo di applicazioni per Android è **Eclipse**, fornito di un adeguato plug-in.
Per gli utenti meno esperti, che non conoscono l'arte della programmazione, viene fornito lo strumento **App Inventor**, un ambiente di sviluppo che permette di creare semplici App, per uso esclusivamente personale, tramite una interfaccia grafica.

[5] **Wikipedia**, Android. Disponibile su http://it.wikipedia.org/wiki/Android.

App Inventor fa parte di Google Labs, “luogo” virtuale ove gli ingegneri di Google sviluppano le nuove applicazioni sperimentali.
Le Apps sviluppate con App Inventor non possono essere caricate sull'Android Market.

1.3.2. Le Applicazioni

Le applicazioni sono i programmi che vengono eseguiti dall'utente tramite l'interfaccia grafica del terminale Android.
Le applicazioni di Android sono sviluppate all'interno di un framework, ossia di una struttura dati specifica. La struttura del framework è molto chiara se si utilizza l'ambiente di sviluppo (SDK) con Eclipse; tuttavia il mancato utilizzo di Eclipse non impedisce di scrivere applicazioni Android funzionanti.
Le applicazioni Android sono caratterizzate da una dualità: **parti dinamiche scritte in Java** e **parti statiche scritte in XML**. Le parti statiche possiedono caratteristiche che non cambiano durante l'esecuzione dell'applicazione, come per esempio il colore dello sfondo. Mentre delle parti dinamiche fanno parte gli aspetti programmatici come per esempio la gestione degli eventi. Questo dualismo è però solo apparente. Durante l'esecuzione, infatti, la Dalvik Virtual Machine (DVM) esegue sempre un solo programma.

1.3.3. La Dalvik Virtual Machine

Tramite l'SDK (o meglio: tramite gli strumenti utilizzati mediante l'SDK) trasformiamo la nostra applicazione Android in un codice intermedio chiamato bytecode; questo è esattamente quello che accade abitualmente in Java, ossia:

```
Codice Java: compilazione : bytecode : VM -> esecuzione reale del
programma
```

Il bytecode Java viene eseguito dal programma chiamato macchina virtuale (**Virtual Machine, VM**), invece negli ambienti Android viene usata una variante chiamata Dalvik Virtual Machine (DVM) [6]. Ogni terminale Android ha la sua DVM che ha il compito di eseguire il bytecode di un’App. Si ottiene quindi la seguente catena di esecuzione:

[6] **Wikipedia**, Dalvik Virtual Machine. Disponibile su http://it.wikipedia.org/wiki/Dalvik_virtual_machine

```
Applicazione Android : compilazione : bytecode : DVM -> esecuzione
reale dell'applicazione Android
```

Quindi, essendo la DVM uguale per tutti i dispositivi Android, ogni applicazione può essere eseguita su ogni terminale, indipendentemente dal costruttore e dall'implementazione. La conseguenza di questo aspetto è che molti costruttori di dispositivi mobili hanno scelto Android come sistema operativo; in questo modo possono fornire ai propri utenti un ambiente condiviso e migliaia di App, installabili su quasi ogni dispositivo.

1.3.4. Creazione, compilazione, emulazione

Tramite l'SDK possiamo passare dallo sviluppo della nostra applicazione alla sua effettiva esecuzione su un emulatore, piuttosto che un dispositivo concreto. Per descrivere l'applicazione al dispositivo prescelto si utilizza il file *Manifest.xml* Possiamo quindi affermare che un'applicazione è descritta completamente da una tripletta:

- Codice Java: Si afferma che il linguaggio Java per applicazioni Android sia il Java standard (esistono delle contestazioni a questa affermazione). Un dato di fatto: la virtual machine è diversa (Dalvik). Dal punto di vista del programmatore l'approccio cambia: si sta programmando un terminale mobile che risponde necessariamente ad eventi (touch schermo, azioni da tastiera). Nella tipica applicazione Android non ci sono metodi "main": tutto viene pilotato dall'utente; potremmo dire che l'utente stesso è il programma principale.
- Risorse XML: Parte dichiarativa contenente diverse informazioni: informazioni di layout e supporto multilingue.
- Andorid Manifest XML: File che descrive l’applicazione al dispositivo.

Il codice Java viene compilato insieme all'xml per generare un file con estensione *.apk*: esso contiene il bytecode per la DVM. Successivamente può essere installato sul emulatore o su un dispositivo fisico.

1.3.5. Confronto con iOS di Apple

Una differenza tra iOS ed Android è di natura strategica; tale differenza può essere riassunta in questa frase:

Android è un master, iOS è uno slave.

Questa distinzione non è di merito; si tratta di termini tecnici che servono per definire la posizione dei terminali rispetto agli altri dispositivi elettronici in grado di comunicare con loro. Con questa espressione si intende parlare di un terminale Android che può vivere senza essere tributario di alcun altro terminale basato su quel sistema operativo, ciò costituisce un'unità indipendente. I dispositivi basati su iOS sono progettati per essere dipendenti dal cosiddetto "hub", un secondo dispositivo cui sono attribuiti maggiori permessi operativi.

1.4. IOS Apple

IOS è il sistema operativo mobile di **Apple**. Originariamente sviluppato per l'**iPhone**, è stato successivamente esteso per gli altri dispositivi mobili della famiglia Apple come **iPod touch** e **iPad**. L'interfaccia utente di IOS è basato sul concetto di “manipolazione diretta”, implementato mediante il multi-touch e diversi elementi di controllo di come i cursori, interruttori e pulsanti [7].

La risposta agli input dell'utente è immediata e fornisce un'interfaccia fluida.

Tutti i dispositivi elencati precedentemente sono muniti di accelerometro, il quale viene utilizzato da alcune applicazioni per acquisire le gestualità dell’utilizzatore, come per esempio lo scuotere o il ruotare del dispositivo.

IOS come Mac OS X è una derivazione di UNIX e usa un microkernel XNU Mach basato su Darwin OS ed è costituito ha quattro livelli di astrazione:

- il Core OS layer,
- il Core Services layer,
- il Media layer,
- Cocoa Touch layer.

[7] **Wikipedia**, iOS Apple. Disponibile su http://en.wikipedia.org/wiki/IOS_Apple.

1.4.1. Caratteristiche

❖ Home screen

La schermata iniziale consente di visualizzare le icone delle applicazioni e un dock sul fondo dello schermo, nel quale gli utenti possono riporre le applicazioni maggiormente utilizzate. La schermata iniziale viene visualizzata all'avvio del dispositivo o premendo il pulsante "Home" (un pulsante fisico del dispositivo). Nella parte superiore dello schermo è presente una barra di stato usata per visualizzare i dati come: il tempo; il livello della batteria e potenza del segnale. Il resto dello schermo è dedicato all'applicazione corrente.

Dall'iOS versione 3.0, una funzione di ricerca Spotlight è disponibile nella schermata iniziale, permettendo agli utenti di cercare applicazioni, e-mail, contatti e file simili.

❖ Cartelle

Con iOS 4 è stato introdotto un semplice sistema di archiviazione delle App. Per creare una cartella basta trascinare una applicazione sopra un'altra, una volta fatto questo la cartella viene creata e a questo punto possono essere trascinate al suo interno altre App (al massimo 12 su iPhone e 20 su iPad). Un titolo per la cartella viene dato automaticamente in base al tipo di applicazioni che contiene, ma può essere modificato dall'utente.

❖ Notifiche

Grazie l'aggiornamento a iOS 5, la funzionalità di notifiche è stata rinnovata. Infatti se prima era solo possibile vedere le notifiche sulle icone delle applicazioni o delle cartelle, ora sono raccolte all'interno di una finestra, visualizzabile trascinando in basso la parte superiore dello schermo (simile al sistema operativo Android).

❖ Applicazioni già installate

IOS viene rilasciato con al suo interno diverse App, le quali permettono all'utente di usare tutte le caratteristiche del proprio smartphone o tablet.

Esempio di App: telefono, il web browser Safari, il gestore delle e-mail e dei messaggi, il calendario, l'agenda, il servizio meteo, le mappe ecc..

❖ Multitasking

Prima dell'iOS 4 il multitasking non era presente, infatti esisteva solo la possibilità di selezionare un'applicazione alla volta, questo perchè Apple temeva che l'esecuzione di più applicazioni contemporaneamente avrebbe portato a un consumo eccessivo della batteria.

A partire da iOS 4 il multitasking è supportato attraverso sette API:

- Audio di sottofondo
- VOIP
- Background location
- Notifiche Push
- Notifiche locali
- Task finishing
- Fast app switching

❖ Cambio di applicazione

Premendo due volte il pulsante di Home viene visualizzata, nella parte inferiore dello schermo, la lista delle applicazioni attive in quel momento. Cliccando su un icona la app selezionata viene aperta, mentre se si tiene premuta viene data la possibilità all'utente di chiuderla.

1.4.2. Sviluppo

Lo sviluppo di una App per il sistema operativo iOS viene agevolata mediante l'utilizzo di una SDK dell'Apple. Essa contiene il simulatore di iPhone e iPad che permette di testare la App durante lo sviluppo, l'editor **Cocoa**, il framework di sviluppo e diversi manuali per aiutare sia i programmatori alle prime armi che quelli esperti. Come linguaggio di programmazione viene usato Object-C.

Come nota negativa questa SDK può essere solamente installata su macchine **Mac,** impedendo ai programmatori sprovvisti di questo supporto, di sviluppare App per dispositivi mobili Apple.

1.5. SQLite

SQLite è una libreria software open source, scritta in linguaggio C che implementa un **DBMS SQL** di tipo **ACID,** incorporabile all'interno di applicazioni[8].

SQLite permette di creare una base di dati (comprese tabelle, query, form, report) contenuta in un unico file, come nel caso dei moduli Access di Microsoft Office e Base di OpenOffice.org.

SQLite è un processo standalone utilizzabile di per sé, ma può essere incorporato all'interno di un altro programma. È utilizzabile con il linguaggio C/C++, ed esistono legami anche per altri linguaggi. È inoltre stato integrato nella versione 5 di PHP, consentendo a questo popolare linguaggio di disporre di un altro RDBMS indipendentemente dalla presenza di MySQL. Viene utilizzato in Mozilla Firefox per memorizzare i bookmark, la cronologia di navigazione ed altre informazioni, infine anche su dispositivi mobili che utilizzano come sistema operativo Android e iOS.

Una peculiarità di SQLite è la gestione flessibile dei tipi di dati, ogni campo può contenere qualsiasi tipo di dato.

1.5.1. Vantaggi e Svantaggi di SQLite

Vantaggi:

- è compatta (meno di 500KB per l'intera libreria alla versione 3.6.14);
- è molto veloce;
- il codice sorgente è liberamente disponibile, chiaro e ben commentato;
- supporta buona parte dello standard SQL92;
- è semplice da utilizzare;
- Soddisfa le proprietà ACID
- è multipiattaforma;
- include un programma di utilità a riga di comando per accedere al database anche manualmente (come su MySQL, Postgresql e tanti altri DB) o tramite scripting;
- supporta database che possono essere anche molto grandi; attualmente il limite è 2TB (241 byte);

[8] **Wikipedia**, SQLite. Disponibile su http://it.wikipedia.org/wiki/SQLite.

- un database consiste di un unico file, il cui formato interno è indipendente dalla piattaforma;
- non ha dipendenze esterne;
- normalmente non richiede alcun lavoro di amministrazione, ma supporta il comando "SQL VACUUM" nel caso si renda necessario compattare esplicitamente i dati del database.

Svantaggi:

- non offre le stored procedure.
- non prevede la gestione dei permessi d'accesso, demandata al software con cui si interagisce con il database e/o al meccanismo dei permessi del file system;
- non ha una vera gestione della concorrenza: le applicazioni che lo utilizzano, se necessario, devono implementarla;
- per garantire la coerenza del file del database sono usati i lock del file system, e quindi vi possono essere problemi qualora quest'ultimo non li implementi correttamente, ad esempio con file system di rete (come NFS);
- non offre alcuna cache per le query;
- non ha protocolli di rete;
- non supporta alcuni importanti costrutti SQL quali RIGHT JOIN e FULL OUTER JOIN;
- non supporta le sottoquery variabili;
- non supporta la scrittura diretta nelle viste;
- non consente di modificare, cancellare o rinominare le colonne di una tabella: il comando ALTER TABLE è infatti limitato alla modifica del nome della tabella e all'aggiunta di colonne in coda alla stessa;
- non supporta trigger di tipo "FOR EACH STATEMENT" (solo "FOR EACH ROW", eventualmente combinato con "INSTEAD OF");

1.6. JavaScript Object Notation

Acronimo **JSON, JavaScript Object Notation,** è un formato adatto per lo scambio dei dati in applicazioni client-server. Viene usato in **AJAX** come alternativa a **XML/XSLT**. La semplicità di JSON ne ha decretato un rapido utilizzo specialmente nella programmazione in AJAX. Il suo uso tramite JavaScript, è particolarmente semplice, infatti l'interprete è in grado di eseguirne il parsing tramite una semplice chiamata alla funzione eval(). Questo fatto lo ha reso velocemente molto popolare, anche grazie alla diffusione della programmazione in JavaScript nel mondo del Web [9].

I tipi di dati supportati da questo formato sono:

- booleani (true e false);
- interi, reali, virgola mobile;
- stringhe racchiuse da doppi apici (");
- array (sequenze ordinate di valori, separati da virgole e racchiusi in parentesi quadre [ ]);
- array associativi (sequenze coppie chiave-valore separate da virgole racchiuse in parentesi graffe);
- null.

Di seguito viene riportato un esempio di JSON:

```
{
      "type": "menu",
      "value": "file",
      "items": [
            {"value": "new", "action": "CreateNewDOc"},
            {"value": "open", "action": "OpenDoc"},
            {"value": "close", "action": "CloseDOc"}
      ]
}
```

Concludendo il JSON permette di inviare un qualsiasi oggetto Java rappresentato in stringa.

[9] **Wikipedia**, JSON. Disponibile su http://it.wikipedia.org/wiki/Json.

1.7. JQuery Mobile

JQuery Mobile è una strategia mobile che permette, mediante linguaggio JavaScript, di sviluppare un'interfaccia utente funzionante sulla maggior parte dei web browser per smartphone e tablet.

Per rendere questo possibile tutte le pagine di JQuery Mobile sono scritte utilizzando una semantica HTML che garantisce la compatibilità con praticamente qualsiasi dispositivo abilitato al web. Nei dispositivi che interpretano file CSS e JavaScript, questo framework permette di trasformare la semantica di una pagina in una ricca esperienza interattiva paragonabile all'utilizzo di un'applicazione nativa.

La gestione della navigazione e della transizione animata di una pagina viene implementata con AJAX [10].

1.7.1. Piattaforme supportate

Di seguito sono elencati i dispositivi che supportano JQuery Mobile suddivisi in quattro livelli di compatibilità:

Livello A – Ottima esperienza interattiva, sono garantite anche le transizioni animate di pagina.

- Apple iOS 3.2-5.0 beta
- Android 2.1-2.3
- Windows Phone 7
- Blackberry 6.0
- Blackberry Playbook
- Palm WebOS (1.4-2.0)
- Firebox Mobile (Beta)
- Opera Mobile 11.0
- Kindle 3
- Chrome Desktop 11-13
- Firefox Desktop 3.6-4.0
- Internet Explorer 7-9
- Opera Desktop 10-11

[10] **JQuery**, JQuery Mobile. Disponibile su http://jquerymobile.com.

Livello B – Ottima usabilità ma non c'è il supporto delle funzioni AJAX di navigazione delle pagine-

- Blackberry 5.0
- Opera Mini (5.0-6.0)
- Windows Phone 6.5

Livello C – Non garantiscono una buona esperienza interattiva

- Blackberry4.x
- Opera Mini (5.0-6.0)
- All older smartphone platforms and featurephones - Any device that doesn't support media queries will receive the basic, C grade experience

In questo momento non supportati

- Nokia S60: Prossimo obiettivo nella Beta 2
- Meego
- Samsung Bada
- Palm WebOS 3.0

1.7.2. Anatomia di una pagina

La struttura della "pagina" di JQuery Mobile è ottimizzata per supportare pagine singole, o link ad altre pagine contenute nello stesso documento HTML.

Un sito web creato con jQuery Mobile deve iniziare con un tag *'doctype'* HTML5 per sfruttare appieno tutte le funzionalità del framework. I vecchi dispositivi con dei browser che non supportano HTML5, ignorano il tag *'doctype'* e i vari attributi personalizzati [11].

All'interno del "*head*" della pagina devono essere inseriti tutti i riferimenti a JQuery, JQuery Mobile e al tema CSS per dispositivi mobili.

All'interno del tag "*body*" ogni "pagina" sul dispositivo mobile è contenuta all'interno di un elemento (di solito un *div*) con attributo i *data-role = "page"*.

[11] **JQuery**, JQuery Mobile Anatomy of a page.
Disponibile su http://jquerymobile.com/demos/1.0rc1/docs/pages/page-anatomy.html

Nel contenitore della "pagina" qualsiasi markup HTML può essere utilizzato, tuttavia le pagine di JQuery Mobile sono suddivise in tre **div**, ognuno dei quali implementa rispettivamente: l'intestazione, il contenuto e il piè di pagina. Anche in questo caso sono definiti dall'attributo *data-role = "header" o "content" o "footer"*.

Di seguito è riportato un esempio di pagina completa:

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
<link rel="stylesheet"
href="http://code.jquery.com/mobile/1.0b1/jquery.mobile-1.0b1.min.css" />
<script type="text/javascript" src="http://code.jquery.com/jquery-
1.6.1.min.js"></script>
<script type="text/javascript"
src="http://code.jquery.com/mobile/1.0b1/jquery.mobile-
1.0b1.min.js"></script> </head>
<body>
<div data-role="page">
<div data-role="header">
<h1>Page Title</h1>
</div><!-- /header -->
<div data-role="content">
<p>Page content goes here.</p>
</div><!-- /content -->
<div data-role="footer">
<h4>Page Footer</h4>
</div><!-- /footer -->
</div><!-- /page -->
</body>
</html>
```

1.7.3. Collegamento a una pagina esterna

JQuery Mobile automatizza il processo di costruzione di siti e applicazioni che sfruttano AJAX.

Di default, quando si fa clic su un link che punta a una pagina esterna (es. prodotti.html), il framework analizza il link per generare una richiesta AJAX; mentre si attende l'apertura della pagina è visualizzata un'immagine di caricamento.

Se questa richiesta ha successo, il contenuto della pagina viene aggiunto al **DOM** (**Document Object Model**) e tutti i widget mobili sono inizializzati automaticamente, dopo di che la pagina viene visualizzata mediante una transizione animata.

Se invece la richiesta AJAX fallisce, il framework visualizza un breve messaggio d'errore che scompare dopo poco tempo; così facendo viene spezzato il flusso di navigazione [12].

1.7.4. Collegamento a una pagina interna

Un singolo documento HTML può contenere più pagine le quali vengono caricate insieme. Ogni pagina è contenuta in un distinto div di tipo "page" e ha bisogno di essere identificata univocamente nel file tramite un ID, il quale permette il collegamento fra le varie pagine interne. Per aprire una pagina interna basta inserire nell'attributo *href* del tag *<a>* il suo identificativo preceduto dal simbolo # (*href = "#ID della pagina da richiamare"*)[13].

Viene riportato un esempio di 2 "pagine" navigabili contenute all'interno di un unico file.

```
<body>
<!-- Start of first page -->
<div data-role="page" id="foo">
        <div data-role="header">
                <h1>Foo</h1>
        </div><!-- /header -->
        <div data-role="content">
                <p>I'm first in the source order so I'm shown as the
page.</p>
                <p>View internal page called <a href="#bar">bar</a></p>
        </div><!-- /content -->
        <div data-role="footer">
                <h4>Page Footer</h4>
        </div><!-- /footer -->
</div><!-- /page -->

<!-- Start of second page -->
<div data-role="page" id="bar">
        <div data-role="header">
                <h1>Bar</h1>
        </div><!-- /header -->
        <div data-role="content">
                <p>I'm first in the source order so I'm shown
                        as the page.</p>
                <p><a href="#foo">Back to foo</a></p>
        </div><!-- /content -->
        <div data-role="footer">
                <h4>Page Footer</h4>
        </div><!-- /footer -->
</div><!-- /page -->
</body>
```

[12] **JQuery**, Linking pages. Disponibile su http://jquerymobile.com/demos/1.0rc1/docs/pages/page-links.html.

[13] **JQuery**, Linking pages. Disponibile su http://jquerymobile.com/demos/1.0rc1/docs/pages/page-links.html.

1.7.5. Aggiornamento automatico dei titoli delle pagine

Quando la prima pagina di un sito, basato su JQuery Mobile , viene caricata il framework, mediante AJAX, acquisisce l'intero contenuto della pagina richiesta. Così facendo tutte le pagine sono inserite all'interno dello stesso **DOM**; quest'approccio è necessario per garantire la navigazione ma porta ad avere un unico titolo per tutte le pagine.

Per rimediare a questo, il framework analizza automaticamente il titolo della pagina e cambia l'attributo *title* del documento principale. Se sono presenti più pagine all'interno di un unico file, è aggiunto a ogni singolo contenitore di pagina l'attributo *data-title* [14].

1.7.6. Creare un collegamento alla pagina chiamante.

JQuery Mobile permette allo sviluppatore di creare un pulsante per riaprire la pagina chiamante utilizzando solamente l'attributo *data-rel="back"* all'interno di un tag *<a>* (**anchor**). Eventuali click su questo tag si limitano ad andare indietro di una voce nella cronologia di navigazione del sito web; così facendo qualsiasi url inserito nell'attributo *href* viene ignorato.

Se invece non si vuole andare indietro nella cronologia ma generare un vero e proprio passaggio indietro basta aggiungere all'**anchor** l'attributo *data-direction="reverse".*

1.7.7. Conclusioni

Vantaggi

- JQuery mobile permette di creare siti web e web application utilizzabili con la maggior parte dei dispositivi mobili;
- Si riesce a mettere a disposizione di tutti gli utenti delle applicazioni senza passare dai canali di distribuzione canonici, come per esempio App Store o Android Market.

[14] **JQuery**, Titles. Disponibile su http://jquerymobile.com/demos/1.0rc1/docs/pages/page-titles.html

Svantaggi

- JQuery mobile, in questo momento, è una versione beta e quindi sono presenti diversi bug che rendono difficile lo sviluppo di applicazioni complesse;
- Non sono delle App native ma solamente dei siti o web application con un tema mobile.
- Per accedere all'App si deve usare il web browser del dispositivo.

Per ulteriori informazioni sul framework e sui suoi componenti consultare il sito http://www.jquerymobile.com/

Capitolo 2.

Appcelerator Titanium

Appcelerator Titanium è una piattaforma per lo sviluppo di applicazioni per dispositivi mobili e desktop tramite tecnologie web. Appcelerator Titanium, pubblicato nel Dicembre 2008, è stato sviluppato da Appcelerator Inc. Il supporto per lo sviluppo su piattaforme iPhone e Android è stato aggiunto nel Giugno 2009, mentre per iPad nell'Aprile 2010. Da poco (Giugno 2011) è stato anche inserito il supporto per BlackBerry [15].

Il punto di forza di questo framework, che ne ha determinato il successo, è la possibilità di creare applicazioni native, sia nel ambito mobile che in quello desktop, utilizzando linguaggi sfruttati per la creazione di web application. È comunque necessario avere conoscenze specialistiche per sviluppare applicazioni software native per ogni tipo di piattaforma di calcolo.

Oltre a scaricare il framework e l'SDK di Appcelerator Titanium, per lo sviluppo di applicazioni per dispositivi mobili, è necessario scaricare l'SDK per iOS e di Android; quest'ultima può essere installata su qualsiasi sistema operativo desktop, mentre per la prima c'è il vincolo di utilizzare una macchina con installato Mac OS X.

Nell'aprile 2010 è stata ampliata la linea di prodotti dando la possibilità ai programmatori di sviluppare applicazioni native anche per tablet.

Nel giugno 2011, Appcelerator ha rilasciato il nuovo SDK, la versione 1.7, e un completo IDE per lo sviluppo.

Concludendo Appcelerator Inc. offre anche servizi cloud-based per il packaging, il testing e la distribuzione di applicazioni software sviluppate su questa piattaforma

[15] **Wikipedia**, Appcelerator Titanium, http://en.wikipedia.org/wiki/Appcelerator_Titanium.

2.1. Sviluppo di un'applicazione

Un'applicazione sviluppata con il framework Titanium, che sia per ambiente desktop che mobile, è suddivisa, come si vede dalla Figura 2.1 in 4 parti [16]:

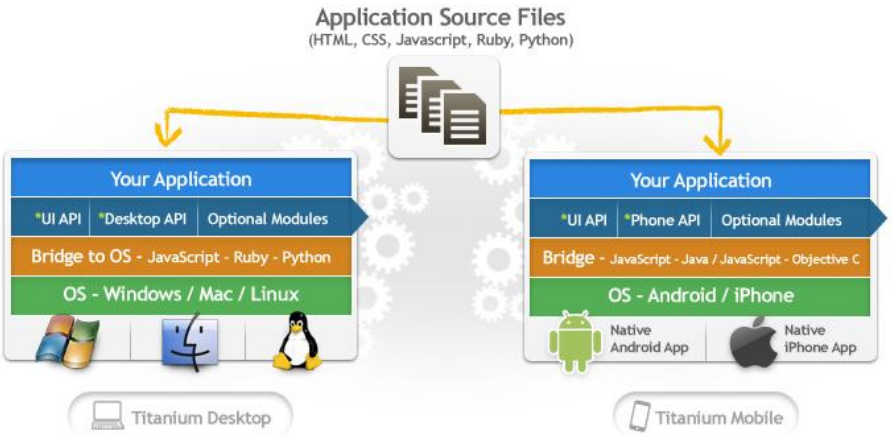


Figura 2.1: Struttura logica di una applicazione

I. Codice **html/css/javascript** che implementa il "core" dell'applicazione, la logica e l'interfaccia utente (UI);

II. API che permettono l'utilizzo di componenti e/o funzionalità del dispositivo, per esempio la gestione della fotocamera o degli eventi rilevati dall'accelerometro;

III. Un ponte che permette di convertire il codice scritto in codice nativo (Es. conversione da JavaScript a Java o Objective C);

IV. La parte che permette di creare i pacchetti dell'applicazione sviluppata.

Da questo punto verrà analizzato solamente Appcelerator Titanium Mobile.

2.2. Struttura di Titanium

Anche se ogni piattaforma, che supporta Titanium, è implementata in modo diverso, è concettualmente simile nel design. Titanium funziona traducendo il codice JavaScript in un codice nativo e successivamente invoca la piattaforma sottostante per costruire il pacchetto finale.

[16] **Appcelerator**, Getting started with Titanium Studio. Disponibile su http://wiki.appcelerator.org/display/tis/Getting+Started+with+Titanium+Studio.

Titanium implementa la sua funzionalità mediante tre elementi distinti [17]:

- **Pre-compiler:** il suo ruolo è quello di prendere il codice JavaScript dell'applicazione e ottimizzarlo (ridurre gli spazi, ridurre le dimensioni dei simboli, ecc) e quindi creare una gerarchia di dipendenza di tutte le API di Titanium utilizzate dall'applicazione,
- **Front-end compiler:** Il ruolo del front-end compiler è di generare il codice specifico per la piattaforma nativa,
- **Platform compiler & packager:** Ogni piattaforma nativa ha un set di strumenti associati (ad esempio, Xcode per IOS) che vengono utilizzati per compilare l'applicazione finale. Dopo la compilazione, l'app è confezionata per l'esecuzione sia sul simulatore che sul dispositivo per test. Successivamente questo pacchetto potrà essere distribuito anche mediante i canali messi a disposizione dal produttore del sistema operativo mobile (App Strore per iOS e Android Market per Android).

2.3. Concetti sul design di Titanium

Il design Titanium è strutturato con criteri in teoria familiari agli sviluppatori JavaScript, inoltre Titanium utilizza il tipico pattern architetturale **MVC** (**Model View Controller**) per la creazione dell'interfaccia grafica dell'applicazione.

L'interfaccia utente di Titanium è nativa. Anche se si utilizza JavaScript per l'applicazione, a differenza di una web application, l'interfaccia utente è costruita con componenti nativi. Quando Titanium compila l'app, utilizza le componenti grafiche native e non quelle del browser web. Ciò significa che l'applicazione si comporta come una vera applicazione nativa.

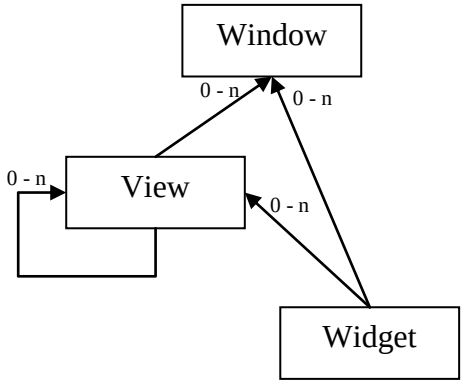
In Titanium, si utilizza un set standard di API già presente nel framework, tale da permettere di sviluppare più facilmente l'applicazione. Queste API, inoltre, forniscono il supporto per utilizzare i componenti e le peculiarità del dispositivo; per esempio il GPS, l'accelerometro, la wifi ecc..

[17] **Appceleretor**, The Titanium Architecture. Disponibile su http://wiki.appcelerator.org/display/guides/The+Titanium+Architecture

Di seguito sono elencati i componenti principali per la progettazione grafica mediante Titanium:

- Window
- View – una vista per disegnare dei contenuti sullo schermo
- Widget – sono delle viste speciali che implementano azioni specifiche come i bottoni.

Queste tre componenti, sono gerarchicamente disposte nel seguente modo:



Come si vede in Figura 2.2, all'interno di una finestra possono essere presenti sia viste che widget. La vista può contenere solo altre viste o widget. Mentre i widget sono dei componenti che non possono avere al loro interno altri elementi grafici.

Figura 2.2: Gerarchia dei componenti grafici

2.3.1. Windows

Windows hanno un ciclo di vita e ospitano al loro interno una o più viste. Una finestra può essere: aperta; chiusa; a schermo intero; schermo modale; schermo parziale (nel senso che occupa meno del 100% dello schermo intero) e inoltre può essere di per sé autosufficiente oppure, il suo contenuto controllato da un altro file JavaScript[18].

2.3.1.1. *Esempio di creazione di una semplice finestra*

```
var win = Ti.UI.createWindow ();
var view = Ti.UI.createView ({backgroundColor: "rosso" });
win.add (view);
win.open ();
```

In questo esempio viene creata un finestra (win) alla quale viene aggiunta una vista che ha uno sfondo rosso. Come risultato si ottiene una finestra completamente rossa.

[18] **Appceleretor**, Windows. Disponibile su http://wiki.appcelerator.org/display/guides/The+Titanium+Architecture#TheTitaniumArchitecture-Windows.

2.3.1.2. *URL basato su Windows*

A volte è più facile strutturare l'applicazione portando la logica di una finestra all'interno di un differente file JavaScript. L'URL per reperire il file JavaScript deve essere locale e quindi all'interno del progetto stesso.

Ristrutturando il codice precedente si ottiene:

File `app.js`

```
var win = Ti.UI.createWindow ({url: "view.js"});
win.open ();
```

Nuvo file `view.js`

```
var win = Ti.UI.currentWindow;
var view = Ti.UI.createView ({backgroundColor: "rosso" });
win.add (view);
```

Si noti che la variabile win è definita dalla variabile `Ti.UI.currentWindow`. Titanium definisce un insieme di variabili speciali che mutano automaticamente a seconda del contesto JavaScript.

`Ti.UI.currentWindow` definisce la finestra in quel momento aperta, in questo modo si riesce a fare riferimento alla finestra aperta precedentemente all'interno del file `app.js`. Quando si esegue questo esempio, si ottiene lo stesso risultato del primo esempio.

2.3.2. Views

Le viste sono dei componenti gerarchici dell'interfaccia utente, al loro interno possono essere inseriti altri componenti come altre viste o delle immagini. Grazie a questa loro caratteristica l'interfaccia grafica viene costruita grazie l'uso di viste specializzate (Widget), le quali sono contenute all'interno della libreria di Titanium chiamata `Titanium.UI` [19].

Di seguito viene presentato un esempio di vista contenete un semplice immagine PNG:

```
var win = Ti.UI.createWindow ();
var view = Ti.UI.createView ({BackgroundImage: "myimage.png" });
win.add (vista);
win.open ();
```

Un altro metodo per visualizzare un'immagine è quello di usare una vista specializzata che, grazie ad alcune funzionalità aggiuntive, garantisce una migliore gestione e

[19] **Appceleretor**, Views. Disponibile su http://wiki.appcelerator.org/display/guides/The+Titanium+Architecture#TheTitaniumArchitecture-Views.

visualizzazione dell'immagine. Questo tipo di vista specializzata viene implementata dall'**API ImageView** che garantisce, all'utente finale, di ridimensionare la foto visualizzata.

```
var win = Ti.UI.createWindow ();
var view = Ti.UI.createImageView ({
  immagine: "myimage.png" ,
  larghezza: 24,
  Altezza: 24
});
win.add (vista);
win.open ();
```

Appcelerator Titanium permette ai suoi sviluppatori di rendere interattive le viste, questo viene fatto aggiungendo un **EventListener** che rimane in ascolto di un dato evento (per esempio il clik) e quando viene captato viene eseguita una data funzione.

```
view.addEventListener('click',function()
{
  view.animate({width:96,height:96,duration:1000});
});
```

In questo caso, facendo riferimento all'esempio precedente, quando la vista viene cliccata le dimensioni dell'immagine vengono quadruplicate in un secondo.

2.3.3. Widgets

I **widget** sono semplicemente delle viste speciali che forniscono funzionalità particolari, alcuni esempi di widget possono essere i pulsanti o campi di testo. Tuttavia, a differenza delle viste classiche, i widget non possono contenere altri elementi grafici.
In questo esempio, viene aggiunto un pulsante che gestirà l'animazione dell'immagine, contenuta in una ImageView [20].

[20] **Appceleretor**, Widgets. Disponibile su http://wiki.appcelerator.org/display/guides/The+Titanium+Architecture#TheTitaniumArchitecture-Widgets

```
win = Ti.UI.createWindow();
var view = Ti.UI.createImageView({
  image:"myimage.png",
  width:24,
  height:24
});
var button = Ti.UI.createButton({
  title:"Animate",
  width:80,
  height:40,
  bottom:10
});
button.addEventListener("click",function(){
  view.animate({top:0,duration:500});
});
win.add(view);
win.add(button);
win.open();
```

Dopo l'evento click del pulsante, l'immagine si sposterà alla parte superiore dello schermo, la posizione finale che viene definita impostando la proprietà `top` uguale a `0`.

2.4. Struttura di un progetto

Quando un progetto viene creato, vengono generati diversi file [21]:

- **LICENSE** – questo file descrive la licenza di .
- **LICENSE.txt** – Licenza dell'applicazione per l'utente finale. Attualmente non è usato per I progetti mobile ma sarà usato in futuro.
- **README** – file che descrive il progetto.
- **tiapp.xml** – questo è il principale descrittore dell'applicazione, definisce I dettagli dell' app ed è usato dal packager.
- **build** – in questa directory sono contenuti I file e le risorse di ogni specifico telefono (iOS e Android). Viene usata dal packager e dallo sviluppatore per creare la build del progetto, compilare e creare la propria distribuzione nativa.
- **Resources** – cartella molto importante che contiene tutti I file dell'applicazione, come I file javascript,html, le imagine, ecc..
- **i18n** – cartella opzionale che contiene le risorse per mettere in lingua l'app.
- **app.js file** – contenuto nella cartella Resources, è il punto di partenza dell'software.

[21] **Appcelerator**, The Application project structure. Disponibile su http://wiki.appcelerator.org/display/guides/The+Application+Project+Structure

2.5. Configurazione dell'applicazione

Nella maggior parte delle applicazioni le configurazioni possono essere settate mediante Titanium Developer o Titanium Studio. Tuttavia possono esserci alcuni casi dove è necessario specificare delle proprietà modificando direttamente il file tiapp.xml [22].

iOS-Configurazioni specifiche

Ci sono alcune configurazioni specifiche solo per iPhone che ne permettono il controllo di funzioni o capacità disponibili solo su iPhone

- persistent-wifi – questo tag `<persistent-wifi>` dichiara che il telefono necessita di una connessione persistente a una rete wifi
- prerendered-icon – `<prerendered-icon>` tag che specifica all'iPhone di non aggiungere la lucentezza all'icona dell'APP. Di default il valore è false
- statusbar-style – Setta il colore della barra di stato.

File Info.plist Personalizzato

Per poter personalizzare il file Info.plist, il quale definisce le caratteristiche generiche dell'app per iOS, basta copiarlo dalla cartella `build/iphone` a quella di `root` del progetto. Così facendo tutti i valori modificati all'interno di questo file saranno utilizzati dal compilatore di Titanium.

File AndroidManifest.xml Personalizzato

Per apportare delle modifiche al file AndroidManifest.xml, che definisce le caratteristiche generiche dell'app per Android, bisogna seguire un procedimento del tutto simile a quello descritto in precedenza. L'unica differenza sta nella posizione iniziale del file, ovvero `build/android`.

[22] **Appcelerator**, Application Configuration. Disponibile su http://wiki.appcelerator.org/display/guides/The+Application+Project+Structure#TheApplicationProjectStructure-ApplicationConfiguration

Parte II: Descrizione dell'Applicazione Web Share Mobile

Capitolo 3.

Progettazione dell'applicazione Web Share Mobile

3.1. Web Share Web Application

Il modulo in oggetto denominato **Web Share**, sviluppato da Quix S.R.L., permette la **distribuzione** e la **catalogazione** di documenti eterogenei in formato elettronico assolvendo in modo semplice e rapido il problema dello **smistamento dei report** o di qualunque informazione generata durante i processi aziendali.

Alcuni esempi di documenti che possono essere gestiti mediante Web Share sono: fatture; documenti di trasporto; bolle di accompagnamento; report; comunicazioni aziendali; manuali; verbali di riunione; ecc.

L'applicazione è utilizzabile in **modalità web** mediante un normale browser e un collegamento alla rete aziendale o internet. L'utente dopo avere assolto alla procedura di **autenticazione** mediante utente e password è in grado di visualizzare una **cartella "virtuale"** contenente solo ed esclusivamente i documenti di propria competenza.

I documenti possono essere inseriti anche in modalità **batch** e quindi essere generati da qualunque procedura in grado di creare un documento elettronico (PDF, Word, excel, files di testo ecc.) e posizionarlo in una cartella condivisa accessibile via rete da Web Share.

3.1.1. Gestione degli utenti

La gestione degli **utenti registrati** che potranno utilizzare l'applicativo WebShare avviene mediante apposite pagine visibili solo dall'amministratore del sistema.

Tramite questa gestione sarà possibile **creare nuovi utenti** (inserendo i dati anagrafici, l'username e la password) e assegnarne i diritti, vale a dire quali operazioni può fare sul sistema, per esempio: **sola consultazione** come nel caso dei clienti, **creazione di documenti** per gli utenti interni, **amministrazione**, ecc.

Nel seguente diagramma, in Figura 3.1, vengono elencati le principali categorie di utenti che interagiscono con l'applicazione e la loro posizione gerarchica.

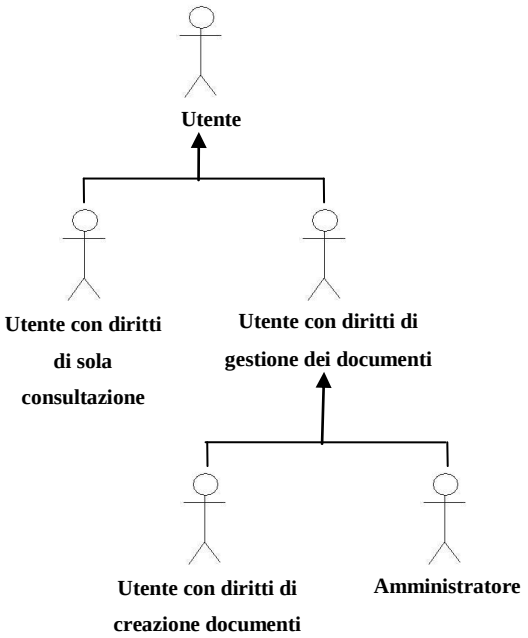


Figura 3.1: Diagramma della classificazione degli utenti

Il sistema prevede anche la possibilità di fornire i diritti a tutti gli utenti di un **gruppo** allo scopo di semplificare il lavoro di pubblicazione.

Tutto questo si può realizzare mediante un'altra applicazione chiamata PUMa, la quale viene trattata nel prossimo capitolo.

3.1.2. Schema E-R

Le varie informazioni di Web Share sui documenti sono contenute in quattro tabelle del data base:

1. *ws_document*, contiene la maggior parte delle informazioni dei documenti.
2. *ws_archive*, al suo interno è presente la lista degli archivi; ogni documento deve fare riferimento a un solo archivio.
3. *ws_target*, contiene gli utenti o i ruoli o i gruppi che possono visualizzare un documento.
4. *ws_status*, contiene lo stato del documento (letto o non letto) per ogni utente che lo può visualizzare e scaricare.

Le relazioni che legano le varie tabelle sono rappresentate dal seguente schema in Figura 3.2:

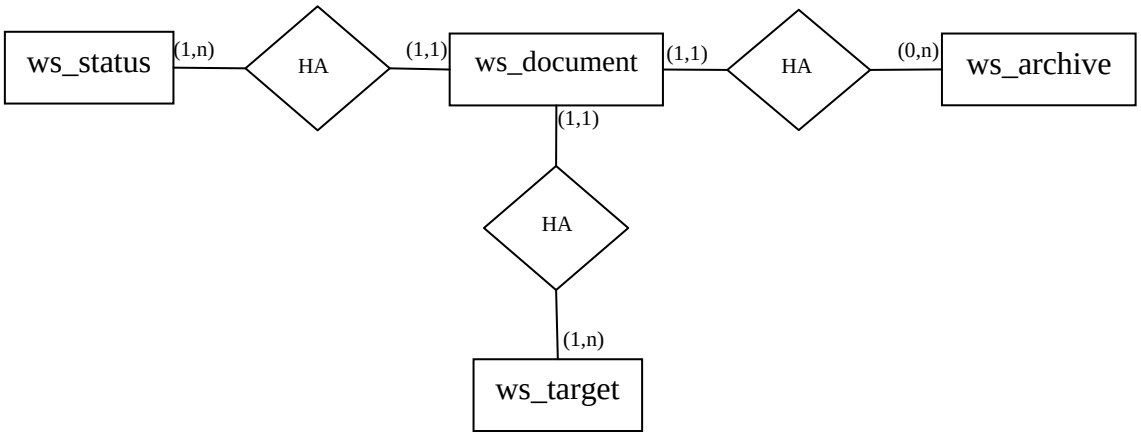


Figura 3.2: Schema E-R di Web Share

3.2. Professional User Manager

PUMa, acronimo di Professional User Manager, è una soluzione sviluppata dalla Quix S.R.L. che ha il compito di:

- assolvere il problema della autenticazione
- gestire le autorizzazioni dell'utente
- garantire la sicurezza degli accessi
- gestire le password e relative scadenze
- facilitare l'amministrazione dei profili

PUMa è una applicazione Java J2EE che implementa tutte le features richieste da un sistema di gestione utenti tra cui:

- gestione utenti e gruppi
- gestione ruoli
- single sign on
- integrazione verso LDAP esterni
- gestione scadenza password
- amministrazione Web-Based
- estendibilità
- regole di accesso
- semplicità di integrazione con altre applicazioni

Delegando a PUMa la gestione dell'autenticazione e delle autorizzazioni si ha il duplice beneficio di eliminare uno sviluppo ripetitivo e di avere un unico punto di accesso per la gestione dei profili.

PUMa rispetta i paradigmi della specifica J2EE e quindi risulta trasparente alle applicazioni Java che delegano al container la gestione della sicurezza.

Dal punto di vista architetturale PUMa è diviso sostanzialmente in tre moduli:

- **plugin di autenticazione** per l'application server
- **applicazione Web** per la gestione utenti
- **libreria di distribuzione** per le singole web application (puma.jar)

3.2.1. Schema E-R

Il data base di PUMa è costituito da tre tabelle, la prima contiene tutte le informazioni degli utenti, la seconda tutti i ruoli, e l'ultima tutti i vari gruppi.

Come si può vedere in Figura 3.3 ogni utente può avere uno o più ruoli e/o fare parte di uno o più gruppi, anche ogni gruppo può avere associato uno o più ruoli.

Dato che la relazione utente - ruolo e gruppo - ruolo sono molto simili quindi si utilizza solo una tabella, chiamata *t_actrole,* per creare la relazione. In particolare questa tabella contiene l'identificativo del ruolo, l'identificativo dell'"Attore" (utente o gruppo) e il campo *Tipo* che definisce se l'attore è un utente o un gruppo.

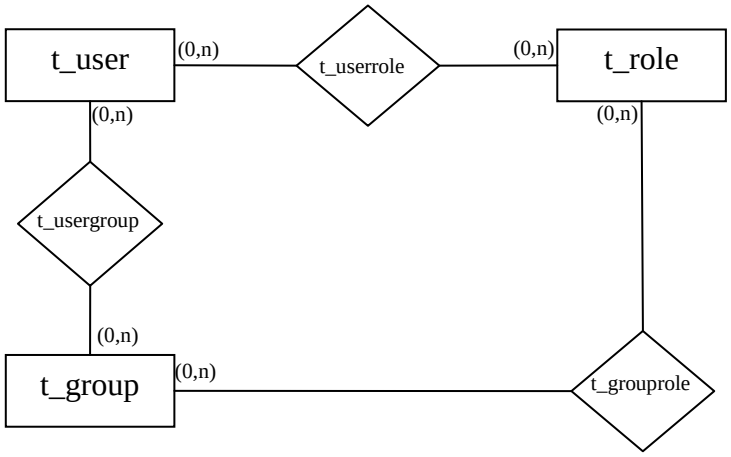


Figura 3.3: Schema E-R di PUMa

3.3. Inquadramento dell'applicativo WS Mobile

La realizzazione del progetto ha come obbiettivo quello di creare un applicativo per tutti i dispositivi mobili che supportano come sistema operativo iOS o Android.

L'App **Web Share Mobile** (o **WS Mobile**) deve garantire a un utente, che si autentica correttamente al server di Web Share, di poter ottenere tutti gli aggiornamenti dei documenti e le loro informazioni, solamente se questi possono essere visualizzati dall'utente.

Tutte le informazioni e i documenti devono essere memorizzati sul dispositivo in modo da assicurare la visualizzazione dei documenti scaricati anche in modalità off-line.

La comunicazione tra processo Client e processo Server avviene nel seguente modo:

1. Il Client WS Mobile si autentica al server e richiede tutte le informazioni, aggiornate o nuove, dei documenti che l'utente loggato può visualizzare.
2. Se l'autenticazione ha successo il Server risponde con un oggetto JSON contenete tutte le informazioni.
3. In un secondo momento, quando l'utente lo desidera, può richiedere al Server anche il documento completo.
4. Il Server quando ottiene la richiesta di download del file, ricontrolla se l'utente è autenticato e se ha il diritto di scaricare il documento richiesto; se esistono queste due condizioni viene inviato al client il documento.

Come detto in precedenza, tutto deve essere memorizzato in locale, in particolare: il documento all'interno del File System e le loro informazioni nel Data Base interno gestito da SQLite.

Il software sarà realizzato mediante l'editor Eclipse e utilizzando il framework Appcelerator Titasnium.

3.4. Requisiti Funzionali

All'interno di questo sotto capitolo vengono descritti tutti i vari requisiti che devono essere implementati all'interno dell'applicativo. Sono inoltre specificate le caratteristiche minime che il Software deve soddisfare per essere accettato e pubblicato dall'Apple e da Android.

3.4.1. Lato Server

All'interno di questo capitolo vengono descritti i requisiti che occorre aggiungere al Server, come l'inserimento di un nuovo filtro di autenticazione e la generazione del JSON da inviare al Client.

3.4.1.1. *RS#1*

Titolo	Autenticazione
Input	Username e Password
Processo	Si deve aggiungere un filtro che gestisca l'autenticazione, in modo da evitare quella usata dalla Web Application. Questo filtro acquisisce

	username e password da due parametri contenuti nella richiesta che vengono controllati da PUMa tramite dei metodi. Se i dati passati sono corretti vengono inviati al client le informazioni richieste oppure, in caso contrario, viene visualizzato un messaggio di errore.
Output	Permesso di scaricare i documenti o le loro informazioni oppure visualizzazione di un messaggio di errore.

3.4.1.2. *RS#2*

Titolo	Creazione e invio di un oggetto JSON
Input	Numero di documenti massimo che l'utente vuole memorizzare sul dispositivo.
Processo	Questo processo si attiva quando l'utente richiede di aggiornare le informazioni dei documenti memorizzate all'interno del dispositivo. Il server acquisisce da questa richiesta il numero massimo di documenti che il Client può memorizzare e ricava le informazioni dei documenti più nuovi e/o aggiornati di recente fino al raggiungimento della soglia sopra citata. Dopo di che tutte le informazioni vengono inserite in un oggetto JSON e inviate al dispositivo richiedente.
Output	Creazione e invio di un oggetto JSON contenente le informazioni dei documenti più recenti.

3.4.1.3. *RS#3*

Titolo	Invio di un documento
Input	Identificativo del documento
Processo	Il processo controlla, una volta acquisito l'identificativo del documento richiesto dal Client WS Mobile, se questo file è ancora presente all'interno del Server e se il richiedente possiede i diritti necessari per la visualizzazione. Nel caso in cui tutti i controlli risultino positivi il file viene inviato altrimenti viene inoltrato un messaggio di errore.
Output	Invio del documento richiesto dall'utente oppure messaggio di errore

3.4.2. Lato Client

Di seguito sono descritte le principali caratteristiche dell'App WS Mobile come: il download dei documenti e delle loro informazioni; la cancellazione dei documenti più vecchi e l'invio di un documento tramite e-mail.

3.4.2.1. *RC#1*

Titolo	Creazione della pagina delle impostazioni
Input	Nome utente, password, url del server, numero massimo di documenti memorizzabili
Processo	Generare una pagina delle impostazioni (seguendo gli standard di *Apple* e di *Android*) che permetta di memorizzare: nome utente; password; url del server e il numero massimo di documenti possibili da memorizzare. Questo numero serve anche per indicare al server quante informazioni dei documenti si desidera scaricare.
Output	Salvataggio delle impostazioni nella tabella *User* del data base.

3.4.2.2. *RC#2*

Titolo	Richiesta di una password di sicurezza.
Input	Password.
Processo	Il processo ha l'obiettivo di chiedere all'utente la password inserita nella pagina delle impostazioni, in modo da impedire a utenti malintenzionati di accedere a documenti riservati. In particolare, viene impedito l'accesso all'App finché non viene inserita la password esatta. Questo controllo viene effettuato ogni volta che si accede all'App e può essere disattivato attivando l'opzione "*memorizza password*" all'interno delle impostazioni.
Output	Accesso all'App oppure ulteriore richiesta della password

3.4.2.3. *RC#3*

Titolo	Cambio utente e/o cambio url del Server
Input	Nome utente e url del Server
Processo	Questo processo ha l'obiettivo di cancellare tutti i documenti e le relative informazioni quando il nome utente o l'url del Server viene cambiato. Per verificare il cambiamento di questi due parametri viene effettuato un controllo all'avvio dell'App o durante la riattivazione dal "*Background*", questo ultimo viene effettuato solo per iOS 4 e successivi e Android.
Output	Cancellazione di tutte le informazioni e tutti i documenti scaricati.

3.4.2.4. *RC#4*

Titolo	Creazione pagina delle *Categorie*
Input	Informazioni contenute nella tabella *Documenti* del data base
Processo	Per creare questa pagina si ricavano, dal data base del dispositivo, tutte le categorie (mai ripetute) dei documenti visualizzate all'interno della prima pagina. Viene aggiunta anche un'ulteriore voce (elemento "*Tutti*") che permette di visualizzare la lista di tutti i documenti.
Output	Visualizzazione della lista di tutte le categorie che hanno almeno un documento.

3.4.2.5. *RC#5*

Titolo	Aggiornamento dei documenti
Input	Nome utente, password e numero massimo di documenti memorizzabili.
Processo	Il processo che permette l'aggiornamento delle informazioni dei documenti più recenti, viene richiamato mediante un pulsante contenuto nella pagina delle *Categorie*. Questi dati sono inviati dal Server utilizzando il formato JSON e una volta che il client ha elaborato la risposta i dati vengono memorizzati all'interno della tabella *Documenti* del Data Base. Successivamente viene aggiornata anche la pagina principale dell'App.
Output	Memorizzazione delle informazioni nella tabella *Documenti* e aggiornamento dell'elenco delle categorie

3.4.2.6. *RC#6*

Titolo	Modifica dei documenti e/o delle loro informazioni
Input	Risposta JSON del Server e la tabella *Documenti*
Processo	Se durante la fase di aggiornamento si rilevano delle modifiche alle informazioni di un documento, esse vengono modificate e il documento viene segnato come "ancora da leggere". Inoltre se si evidenzia una modifica anche nel file, se scaricato, viene cancellato dal dispositivo; anche in questo caso il documento viene segnato come "da leggere" e "da scaricare".
Output	Modifica delle informazioni della data base e eventuale cancellazione del file.

3.4.2.7. *RC#7*

Titolo	Cancellazione dei documenti e delle loro informazioni in eccesso
Input	Tabella *Documenti* e numero massimo di documenti memorizzabili.
Processo	Dopo l'update si controlla se è stato superato il limite dei documenti memorizzabili sul dispositivo. Se ciò si verifica, sono cancellate le informazioni e, se esistono, i relativi file fino a rientrare all'interno del valore massimo definito dall'utente.
Output	Cancellazione dei documenti più vecchi.

3.4.2.8. *RC#8*

Titolo	Creazione pagina dei *Documenti*
Input	Identificativo della categoria selezionata nella pagina delle *Categorie*
Processo	Quando viene selezionata una categoria, contenuta nell'elenco della prima pagina dell'App, viene aperta una nuova finestra la quale contiene una lista di documenti, non necessariamente scaricati in precedenza, che appartengono alla categoria. Ogni elemento dell'elenco è costituito da una descrizione composta da: un'icona per identificare il tipo di file; il nome del documento e la data di creazione o della sua ultima modifica sul Server.
Output	Visualizzazione della lista dei documenti.

3.4.2.9. *RC#9*

Titolo	Download di tutti i file
Input	Nome utente, password e identificativo del documento
Processo	All'interno della pagina dei Documenti viene implementato un pulsante che ha il compito, se cliccato dall'utente, di scaricare i documenti della categoria selezionata oppure tutti i file se selezionata l'opzione "*Tutti*" dalla prima finestra.
Output	Download di tutti i file o solo di quelli appartenenti a una categoria.

3.4.2.10. *RC#10*

Titolo	Mostrare i documenti ancora da leggere e/o ancora da scaricare.
Input	Dati contenuti nella tabella *Documenti*
Processo	Questa funzione ha lo scopo di mostrare graficamente se le informazioni sono ancora da leggere e se il documento deve essere ancora scaricato. In particolare nella prima pagina, vengono evidenziate le righe e aggiunto il numero di documenti ancora da leggere, mentre nella pagina dei *Documenti* appare un pallino se il file è ancora da scaricare e se le informazioni di un documento non sono ancora state visionate l'intera riga risulta evidenziata.
Output	Maggiore vi visibilità dei documenti nuovi.

3.4.2.11. *RC#11*

Titolo	Creazione pagina Informazioni_*Documento*
Input	Identificativo del documento selezionato nella pagina dei *Documenti*
Processo	Quando è selezionato un documento, contenuto nell'elenco della pagina dei *Documenti*, viene aperta una nuova finestra la quale contiene al suo interno tutte le sue informazioni.
Output	Visualizzazione delle informazioni di un documento

3.4.2.12. *RC#12*

Titolo	Download di un file
Input	Nome utente, password e identificativo del documento
Processo	Il seguente processo invia al Server le informazioni sopracitate. Una volta verificate, se correte, il file viene inviato al Client. Al termine del download il documento viene visualizzato sul dispositivo.
Output	Messaggio di errore o visualizzazione del file

3.4.2.13. *RC#13*

Titolo	Visualizzazione di un file
Input	url locale del documento scaricato e tipo di file (questo parametro serve solo per android).
Processo	La visualizzazione di un file è differente tra dispositivi iOS e Android; nel primo caso basta aprire il file, specificando il suo url locale, all'interno di un componente di Titanium chiamato *WebView.* Nel secondo caso invece un file si può aprire utilizzando solamente un'applicazione esterna a WS Mobile. Nel prossimo capitolo verrà spiegata l'esatta implementazione.
Output	Visualizzazione del file

3.4.2.14. *RC#14*

Titolo	Invio di un documento scaricato tramite e-mail.
Input	Url locale del file.
Processo	Il seguente modulo permette di inviare un documento, come allegato, tramite e-mail. Questo processo può essere attivato solamente se il file è presente all'interno del *File System* del dispositivo.
Output	E-mail inviata

3.4.2.15. *RC#15*

Titolo	Pagina di presentazione dell'App
Input	Versione
Processo	Creare una pagina che contenga la descrizione dell'app.
Output	Pagina di presentazione e descrizione.

3.4.2.16. *RC#16*

Titolo	Compatibilità dell'App su dispositivi Apple e Android
Input	Tipo di dispositivo
Processo	Fare in modo che la grafica fra android e ipone sia il più possibile simile e implementare tutti i requisiti sopra citati per entrambe le piattaforme.
Output	Applicazione compatibile con i due sistemi operativi mobili

3.5. Requisiti minimi richiesti da Android e Apple

Apple e Android pretendono che un'App rispetti determinati standard per poter essere messa su *Apple Store* o *Android Market* e successivamente scaricata dagli utenti interessati.

3.5.1. Caratteristiche richieste da Android

- L'applicazione deve essere firmata utilizzando una chiave crittografica privata con una validità che superi il 22 ottobre 2033.
- Devono essere inseriti all'interno del file *manifest.xml* il codice della versione e il suo nome. Queste informazioni sono necessarie per Android al fine gestire automaticamente gli aggiornamenti dell'App installate sui dispositivi.
- Deve essere definita un'icona.[23]

Se si utilizza come framework Appcelerator Titanium gli ultimi due requisiti vengono gestiti automaticamente.

[23] **Android**, Publish on Android Market. Disponibile su http://developer.android.com/guide/publishing/publishing.html

3.5.2. Caratteristiche richieste da Apple

- Descrivere l'applicazione evidenziando le sue funzionalità e/o il suo design unico;
- Creare screenshots che rappresentino nel miglior modo possibile la App; Essi devono essere:
    - Privi di status bar;
    - Con un contenuto leggibile;
    - Creati dal dispositivo (non dal simulatore);
    - Devono essere anche localizzati (differenti per ogni lingua supportata dall'App);
- Includere nell'App gli errori di rete;
- Specificare i dispositivi che supportano l'App (iPhone, iPod Touch e iPad)
- Se la App ha necessità di dati per il funzionamento, durante le prove per la convalida, fornire questi dati su iTunesconect
- Usare delle parole chiave per identificare l'App nel migliore dei modi così facendo si faciliterà la ricerca del software all'interno dell'*App Store*. Non sono ammessi nomi protetti da copyright o offensivi;
- Creare icone e splash screen dell'applicazione per ogni tipo di dispositivo e per ogni risoluzione di schermo (normale o HD);
- Definire la tipologia di App (Book, Business, Education, Entertaiment, Finance, Games, News, Photografy, ecc);
- Definire l'età minima che deve avere l'utilizzatore per scaricarla.[24]

Per ulteriori informazioni consultare il sito: https://developer.apple.com/.

[24] Apple, iTunesConnect_DeveloperGuide. Disponibile su https://itunesconnect.apple.com/docs/iTunesConnect_DeveloperGuide.pdf

3.6. Diagramma Use Case generale

Il diagramma in Figura 3.4 descrive quali azioni possono essere compiute da un qualsiasi utente che usa il programma, con l'unico vincolo di inserire le informazioni corrette per l'autenticazione all'interno della pagina delle impostazioni.

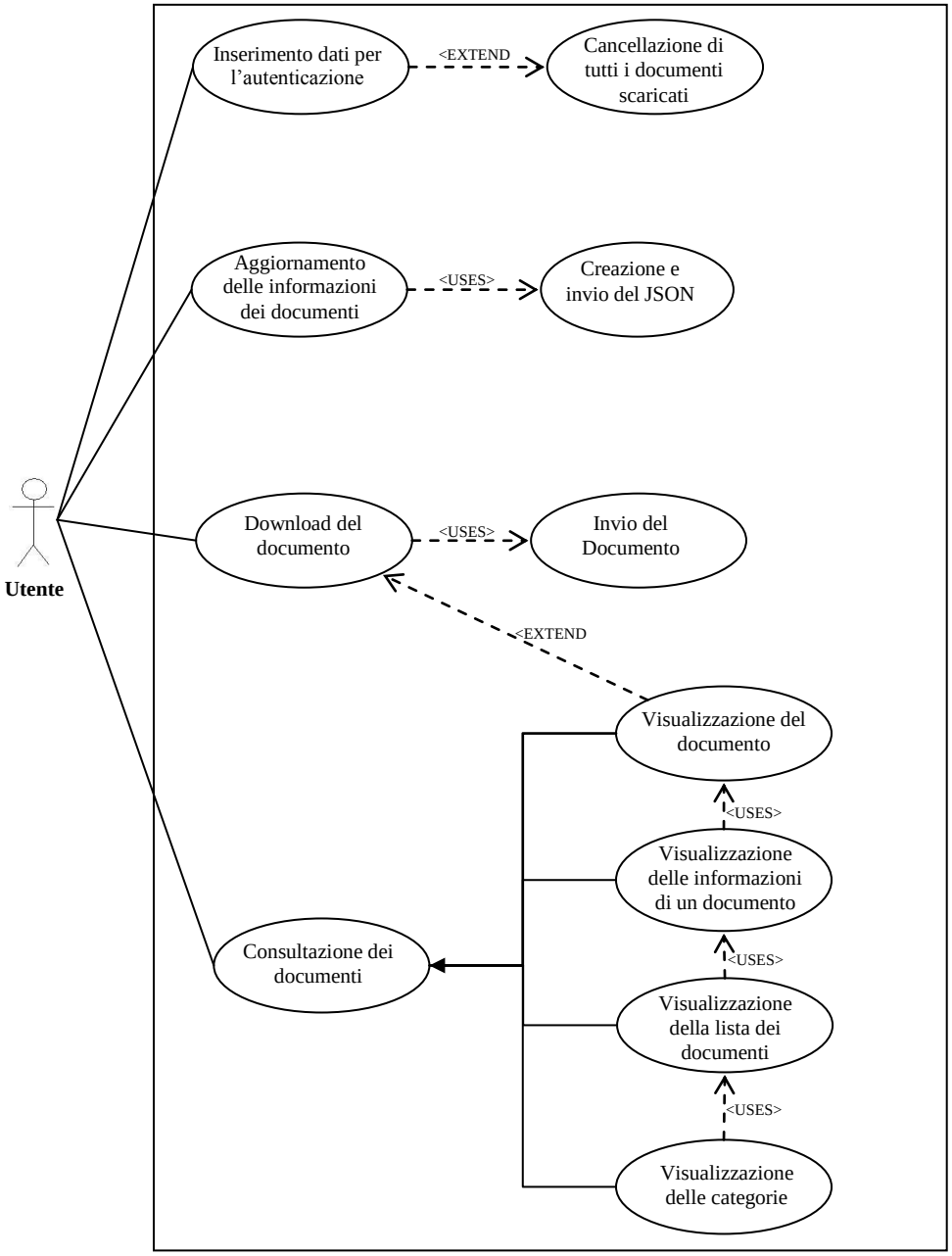


Figura 3.4: Diagramma Use Case dell'applicativo

Parte III: Realizzazione del progetto

La terza e ultima parte di questo elaborato descrive in maniera chiara e precisa la realizzazione di ogni requisito funzionale descritto in precedenza, evidenziando anche le parti di codice più rilevanti.

Il progetto sarà descritto mediante due macro argomenti: la creazione dei moduli lato Server da integrare alla Web Application *Web Share* già esistente e l'implementazione dell'App per dispositivi mobili.

Capitolo 4.

Implementazione dei moduli lato Server

All'interno di questa sezione viene descritto lo sviluppo dei tre moduli che implementano il Server dell'App, in particolare:

- ✓ Il modulo per l'autenticazione dell'utente, implementato mediante una *Basic Authentication*;
- ✓ Il modulo per la creazione e l'invio del file JSON contenente le informazioni dei documenti più recenti;
- ✓ Il modulo per l'invio del file richiesto dal Client.

Questi componenti sono integrati all'interno della Web Application *Web Share*, già esistente e sviluppata mediante il Framework **Struts I**.

4.1. Autenticazione dell'Utente

Il modulo implementa un filtro che acquisendo dalla richiesta *http* nome utente e password, verifica mediante dei metodi di **PUMA,** se l'utente ha diritto di accedere o meno al servizio. In caso di successo dell'autenticazione l'utente viene memorizzato in sessione con tutte le sue informazioni come il ***Gruppo*** e il ***Ruolo*** di appartenenza, questi dati servono per mostrare al richiedente solo i documenti ad esso associati, se invece l'utente non ha i diritti per accedere viene inviavo al Client l'*Errore HTTP 401*, ovvero "*autorizzazione negata*".

Di seguito viene riportato il codice del filtro:

```java
public void doFilter(ServletRequest _request, ServletResponse_response,
               FilterChain filter) throws IOException, ServletException {

        final HttpServletRequest request = (HttpServletRequest) _request;
        final HttpServletResponse response = (HttpServletResponse) _response;
         HttpSession session = request.getSession();

         // try to authenticate them
         String un = null, pw = null;
         // No, see if there are parameters
         if (un == null) {
             un = request.getParameter("user");
             pw = request.getParameter("password");
         }

         if (isAuth(un, pw)) {
             // just authenticated
               User user=null;
               try {
                       user = PumaManagerFactory.getPumaManager().getUser(
                               request.getParameter("user"));
               } catch (PumaFinderException e) {
                       // TODO Auto-generated catch block
                       e.printStackTrace();
               }
               request.getSession().setAttribute("webshare.user", user);
              filter.doFilter(request, response);
              return;
         }

         // Unauthorized - do not call doFilter
         final String BASIC_REALM = "Basic realm=\"Login Users\"";
         response.setHeader("WWW-Authenticate", BASIC_REALM);
         response.setStatus(401);// Unauthorized
     }
```

Questo filtro viene eseguito ogni volta che l'utente richiede di scaricare un file o le informazioni dei documenti più recenti. Per fare questo è stata aggiunta la definizione di un filtro chiamato *AuthenticationFilter* nel file *web.xml* della Web App, dopo di che vengono mappate le condizioni di esecuzione, quando all'interno dell'url sono presenti le seguenti stringhe:

- ***.json** → presente quando l'utente richiede le informazioni dei documenti
- **/d/** → presente quando l'utente vuole scaricare un file

```java
<filter>
     <filter-name>AuthenticationFilter</filter-name>
     <filter-class>it.quix.webshare.web.AuthenticationFilter</filter-class>
</filter>

<filter-mapping>
     <filter-name>AuthenticationFilter</filter-name>
     <url-pattern>*.json</url-pattern>
     <url-pattern>/d/*</url-pattern>
</filter-mapping>
```

4.2. Creazione e invio dell'oggetto JSON

Il seguente metodo ha il compito, se l'autenticazione ha successo, di ottenere tutte le informazioni sui documenti più recenti e inviarli all'utente mediante un oggetto JSON.

Il processo per funzionare correttamente necessita di avere all'interno della richiesta il numero massimo di documenti che il Client può memorizzare, con questa informazione il Server inserisce all'interno del JSON le informazioni di tanti documenti quanti il valore di tale soglia. Inoltre vengono usate anche le informazioni dell'utente **PUMA** memorizzate in Sessione durante l'autenticazione.

Tutto questo avviene quando all'interno del path dell'url è presente il valore ***/documentmobile*** che avvia una **action** che esegue il metodo sopracitato.

4.2.1. Acquisizione delle informazioni

L'acquisizione delle informazioni avviene mediante una ricerca *SQL* all'interno del *DB* di *Web Share*. Questa ricerca si serve di filtri che hanno il compito di selezionare esclusivamente le informazioni utili per il richiedente. In particolare vengono restituite le informazioni di un numero di documenti compatibile con la soglia massima disponibile sul dispositivo mobile e visualizzabili solo da quell'utente; dopo di che tutte queste informazioni vengono memorizzate all'interno di una *Collection* di *Java.*

Successivamente vengono letti sempre da DB e memorizzati all'interno di una *Lista* anche tutte le descrizioni degli archivi. Di seguito viene riportato il codice:

```
private void creaJson(HttpServletRequest request, HttpServletResponse
response)throws IOException, ServletException{
        //LETTURA DB

        DocumentFilter filter = new DocumentFilter();
        String[] targets = ActionHelper.getTargets(request);
        filter.setTargets(targets);
        ……
        filter.setRowNumber(
                Integer.parseInt(request.getParameter("numDocument"))
        );
        ……
        Collection docList = DocumentManager.getInstance().search(filter);
        List archiveList = DocumentManager.getInstance().getArchives(filter);


        //CREAZIONE JSON

        ……
```

Come si può notare dal codice per ottenere i documenti visualizzabili solo dall'utente richiedente viene usata una funzione già esistente, `getTargets(request)`, che: dato l'utente **PUMA** (contenuto nella sessione della ***request***) controlla se:

- Ha un ruolo di amministratore, quindi può visualizzare tutti i documenti presenti nel Server;
- Non ha un ruolo di amministratore, la funzione ritorna a tutti i ***Gruppi*** ai quali appartiene l'utilizzatore.

Queste informazioni vengono passate a un Array di stringhe che viene utilizzato come valore da mettere nel filtro di ricerca. Successivamente viene mostrato il codice del metodo **`getTargets(request)`:**

```
public static String[] getTargets(HttpServletRequest request) {
        User user =
        (User)request.getSession().getAttribute("webshare.user");
        String[] targets = null;
        if (request.isUserInRole(Props.ADMINISTRATION_ROLE)) {
        // Se l'utente è amministratore inserisco un '*' come secondo target
                targets = new String[2];
                targets[0] = user.getDn();
                targets[1] = "*";
        } else {
        // Se l'utente non è amministratore devo definire il filtro
                Collection groups = user.getGroups();
                targets = new String[groups.size() + 1];
                targets[0] = user.getDn();
                int i = 1;
                for (Iterator iter = groups.iterator(); iter.hasNext(); i++) {
                        Group group = (Group) iter.next();
                        targets[i] = group.getDn();
                }
        }
        return targets;
}
```

4.3. Creazione del oggetto JSON

Per la creazione dell'oggetto JSON da inviare al Client viene usata la libreria **Flexjson**, la quale permette di generarlo partendo da una **Collezione Java**.

In questo specifico caso una volta ottenuta la Collection delle informazioni e la Lista degli archivi questo insieme viene inserito all'interno di una ***HashMap,*** successivamente trasformata in un oggetto JSON e inviata al richiedente.

Tramite questa libreria è possibile escludere elementi delle Collezioni superflui, tramite il metodo ***exclude*** e trasformare in un formato standard tutte le date contenute utilizzando il metodo ***trasform***.

```
//LETTURA DB
……
//CREAZIONE JSON
//Il trasform mette la data nel formato yyyy/MM/dd hh:mm:ss invece che in
//millisecondi
JSONSerializer serializer = new JSONSerializer().transform(
               new BasicDateTransformer(), Date.class).transform(
               new DateTransformer("yyyy/MM/dd hh:mm:ss"),
               "DocumentList.referencedate","DocumentList.startdate");

//Creazione mappa, si iseriscono le liste ottenute dalle query
Map m=new HashMap();
m.put("DocumentList", docList);
m.put("ArchiveList", archiveList);

/*creazione del JSON, vengono esclusi gli elementi superflui e si usa il metodo
prettyPrint(true) per permettere una lettura piu'semplice del JSON da parte del
amministratore o del programmatore*/
String rsJSON = serializer.prettyPrint(true).exclude("DocumentList.attribute1",
       "DocumentList.attribute2","DocumentList.class",
       "…").deepSerialize(m);

//attachment invio il file nella rensponse il file
response.setContentType("application/json");
response.setHeader("Content-Disposition",
       "inline;filename=inputParameterJson");
response.getOutputStream().write(rsJSON.getBytes())
}
```

4.4. Invio del file

L'invio di un file richiesto da un utente viene implementato mediante un metodo il quale, dopo aver effettuato i vari controlli sull'esistenza del file e sulla possibilità di visualizzazione del documento da parte dell'richiedente, lo invia e setta all'interno del DB della Web Application un flag a *1*, per indicare che il documento è stato letto. Anche in questo caso la richiesta prima viene controllata dal filtro di autenticazione

Di seguito si riporta il codice del modulo:

```
public void performTask(HttpServletRequest request, HttpServletResponse
        response) {
        //l'user esiste perche' ha superato il filtro iniziale
        User user = (User)request.getSession().getAttribute("webshare.user");
        String[] targets = ActionHelper.getTargets(request);
        /*---Dall'url si ricava che documento è stato richiesto---*/
        ……
        /*-------------------------------------------------------*/
        if ((documentResult != null) && (documentResult.isRead())) {
        // L'utente è abilitato a leggere il documento
                String documentName =
                        DocumentManager.getInstance().getRepository(
                        documentResult.getId()) +
                        System.getProperty("file.separator") +
                        documentResult.getId() + ".dat";
                File documentFile = new File(documentName);
                if (documentFile.isFile()) {
                        sendFile(response, documentResult.getFile(),
                                documentFile, user.getDn());
                }
                try {
                //controllo se esiste gia uno status per quel document.
                        Status status =DAOFactory.getStatusDAO(Props.DS_NAME).
                        getStatus(documentResult.getId(), user.getDn());
                //se sì, controllo se lo status del document è NEW.
                        if (status.getStatus() == 0) {
                        //setto lo status READ.
                                status.setDocument(documentResult.getId());
                                status.setTarget(user.getDn());
                                status.setRecieved(Calendar.getInstance().
                                                        getTime());
                                status.setStatus(1);
                                try {
                                //Aggiorno il DB
                                ……
                                } catch (DAOFinderException ex) {
                                //nel caso di questa eccezione non ho trovato lo
                                //status del document corrispondente.
                                //Creo uno stato e lo setto uguale a 1 come sopra
                                        try {
                                                //Aggiorno il DB
                                                ……
                                        } catch (DAOCreateException e) {
                                                ……
                                        }
                ……
}
```

Capitolo 5.

Implementazione dell'App Web Share Mobile

All'interno di questa sezione viene descritta l'implementazione del programma Client Web Share Mobile esponendo i sei punti principali, ovvero: la struttura della base di dati; la realizzazione della pagina delle proprietà dell'App; la creazione dell'interfaccia grafica; la gestione delle informazioni dei documenti; la gestione dei file (download e visualizzazione) e infine l'implementazione dell'invio dei documenti come allegati tramite una email.

In caso di diversità di sviluppo fra dispositivi iOS e Android queste verranno analizzate per entrambe le piattaforme.

5.1. Struttura del database e la sua connessione al progetto

I dispositivi mobili Apple e Android permettono di creare una base di dati mediante l'utilizzo del DBMS SQLite, già presente all'interno di ogni smartphone o tablet grazie alle sue caratteristiche di leggerezza e di velocità di esecuzione.

In questo paragrafo vengono descritte le varie tabelle del database e al termine vengono esposte le istruzioni necessarie per far interagire l'applicativo con il DBMS.

5.1.1. Descrizione delle tabelle del database

Il DB dell'App è costituito da due tabelle, non collegate fra di loro, le quali hanno il compito di contenere rispettivamente: le informazioni su ogni documento, chiamata ***Documents***, e i dati dell'utente, chiamata ***User***.

Tabella “Documents”

La seguente Tabella 5.1: parametri della tabella 'Documents' contiene al suo interno tutti i parametri della tabella *Documents* con le relative descrizioni. Oltre ai vari campi che contengono le informazioni di un singolo documento sono stati aggiunti anche due flag e l’url del Server per effettuare il download del file. I due flag sopracitati hanno rispettivamente il compito di indicare se le informazioni sono nuove e se il file è stato scaricato correttamente.

Campo	Formato	Descrizione
ID	TEXT	Identificativo del documento
Title	TEXT	Titolo del documento
Description	TEXT	Descrizione del documento
Author	TEXT	Autore del documento
RefDate	TEXT	Data di creazione del documento
StartDate	TEXT	Data di inizio della validità del documento
Archive	TEXT	Identificativo dell’archivio di appartenenza del documento
ArchDesc	TEXT	Descrizione dell’archivio di appartenenza del documento
Category	TEXT	Categoria del documento
Extreference	TEXT	Riferimento esterno
FileSize	TEXT	Dimensione del file
FileName	TEXT	Nome del file
FileType	TEXT	Tipo di file
Downloaded	INTEGER	Flag che indica se il file è stato scaricato o meno. • Downloaded=1→file scaricato • Downloaded=0→file non scaricato
URL	TEXT	Campo che contiene l’url per scaricare il documento
New	INTEGER	Flag che indica se il file è nuovo oppure no • New=1→file nuovo

		• New=0→file non nuovo

Tabella 5.1: parametri della tabella 'Documents'

Tabella "user"

La Tabella del database "user" contiene una copia di tutte le informazioni necessarie all'utente per connettersi al Server. Lo scopo principale di questa tabella è quello di confrontare le informazioni contenute all'interno della pagina delle Impostazioni dell'App, cosi facendo a ogni avvio si verifica se sono state apportate delle modifiche a queste informazioni. Tutto questo poiché le impostazioni sono situate all'esterno dell'App e non si ha modo, mediante il Framework Appcelerator Titanium, di percepire le modifiche che un utente potrebbe effettuare.

Successivamente nella Tabella 5.2: parametri della tabella 'User' vengono elencati tutti i campi con le relative descrizioni della tabella "User"

Campo	Formato	Descrizione
UserName	TEXT	Nome utente
Password	TEXT	Password
URLServer	TEXT	Url del server per poter scaricare le informazioni sui documenti riguardanti l'utente che stà effettuando l'accesso
NumDoc	INTEGER	Numero di file e di descrizioni dei documenti massimo.
lastUpdate	TEXT	Data e ora dell'ultimo aggiornamento
savePassword	TEXT	Flag che abilita o disabilita la richiesta della password al avvio o alla riattivazione dell'App. • savePassword=0→richiesta password • savePassword=1→password non richiesta

Tabella 5.2: parametri della tabella 'User'

5.1.2. Connessione al progetto

Il DBMS, essendo già integrato all'interno del dispositivo, non necessita di una connessione all'interno del codice sorgente basta solamente aprire il database nel seguente modo:

```
var db = Titanium.Database.open('dbWebShare'); //Creazione DB
```

Questa istruzione apre il DB e associa il suo riferimento alla variabile `db`, se non viene rilevata la base di dati indicata ne viene creata una nuova automaticamente con il nome indicato.

La variabile `db`, dichiarata all'interno del file principale **app.js**, per essere utilizzata da tutte componenti dell'App deve essere passata all'interno della dichiarazione di una nuova finestra. In questo modo possono essere passate ad una nuova finestra anche altre variabili.

```
var win_main = Titanium.UI.createWindow({  //creazione finestra
    title:'WS Mobile',
    url:'main_windows/archiveList.js',
    backgroundColor:'Images/sfondo.png',
    db: db      //passaggio alla nuova finestra del riferimento al DB
});
```

Queste variabili passate a una finestra figlia vengono richiamate semplicemente usando l'istruzione `NomeFinestra.Nomevariabile`.

La gestione del database avviane tramite il metodo `execute` il quale permette di eseguire la quasi totalità dei comandi SQL. Di seguito vengono riportati alcuni esempi.

```
win.db.execute('CREATE TABLE IF NOT EXISTS…');
win.db.execute('UPDATE Documents SET……');

var dbrows = win.db.execute(SELECT * FORM Documents');
//Lettura dei record del DB
while (dbrows.isValidRow()){
       alert(dbrows.fieldByName('ID'))  //scrivo in un alert il campo id
       dbrows.next(); //Sposto il puntatore al record successivo
}
```

Quando si eseguono dei comandi `SELECT` all'interno di una tabella è necessario controllare che i dati letti siano veramente presenti all'interno della variabile, se questo non avviene si rischia di incorrere in gravi errori, per esempio l'errore Null Pointer Exception, che portano al crash dell'App.

```
//Controllo dei dati e lettura dei record
try{
       var dbrows = win.db.execute(SELECT * FORM Documents');
       //Leggo
}catch(e){
       //Errore di lettura del data base
}
```

5.2. Impostazioni dell'Applicativo

La App Web Share mobile, per il suo corretto funzionamento, necessita di cinque dati: il nome utente; la password; l'url del server; il numero di documenti massimo da memorizzare nel dispositivo e l'impostazione di memorizzazione della password. Tutte queste informazioni sono contenute all'interno della pagina delle Impostazioni, la quale è implementata seguendo gli standard di Apple e Android; in questo modo è possibile mantenere la struttura dell'App analoga ad altri applicativi già pubblicati in rete.
Questa pagina viene creata all'interno di un file che non fa parte della struttura del Framework Titanium ma appartiene alle applicazioni che sono sviluppate in maniera nativa per le singole piattaforme. Tuttavia Titanium, grazie a delle sue API, permette di gestire tutte le informazioni contenute nelle pagine, indistintamente dalla piattaforma che ospita la App.
Di seguito viene descritto le sviluppo di questa pagina per iOS e per Android.

5.2.1. Sviluppo della pagina delle Impostazioni per iOS

Un requisito fondamentale di un'App per essere accettata e distribuita dalla Apple mediante Apple Store è quello di inserire, se esiste, la pagina delle Impostazioni (o delle Preferenze) all'interno della sezione ***Impostazioni*** di iOS, per questo motivo si ricorre alla creazione di una pagina esterna all'software. Per implementarla viene generato un pacchetto proprietario chiamato *Setting.bundle* il quale contiene il file *Root.plist* che descrive, mediante Tag proprietari, la struttura della pagina. Questo pacchetto per essere integrato all'interno dell'App deve essere collocato nella radice del progetto nel seguente percorso: **modules\iphone**.

Una pagina è composta da diversi campi, come caselle di teso, pulsanti, etichette e altri, i quali sono definiti a livello di codice da diverse coppie di *Tag chiave/valore*.
Tutti gli elementi che compongono una pagina delle Preferenze sono obbligati ad avere quattro informazioni principali:

1. Il tipo di elemento, che ha come chiave la stringa ***Type*** e come valore le stringhe contenute in Tabella 5.3: Tipi di le quali generano un componente diverso ciascuna:

Codice	Descrizione
PSTextFieldSpecifier	Casella di testo
PSTitleValueSpecifier	Etichetta
PSToggleSwitchSpecifier	Pulsante buleano
PSSliderSpecifier	Slider
PSMultiValueSpecifier	Selettore contenente più valori
PSGroupSpecifier	Contenitore di diversi componenti

Tabella 5.3: Tipi di componenti per iOS

2. Il titolo dell'elemento, che ha come chiave la stringa ***Title*** e come valore il titolo dell'elemento che sarà visualizzato nelle vicinanze dell'componente all'interno della pagina;
3. Il valore di default, definito dalla chiave con la stringa ***DefaultValue*** e dal valore che deve assumere il componente di default;
4. La chiave identificativa dell'elemento, definita dalla chiave ***Key*** e da un valore di tipo stringa privo di spazi che identifica univocamente il componente [25].

Di seguito, in Figura 5.1: Pagina delle Impostazioni, viene mostrata la pagina delle Impostazioni di Web Share Mobile.

Figura 5.1: Pagina delle Impostazioni iOS

[25] **Apple**, Implementing Apllication Preferences. Disponibile su http://developer.apple.com/library/ios/#DOCUMENTATION/iPhone/Conceptual/iPhoneOSProgrammingGuide/Preferences/Preferences.html

Oltre ai dati, specificati in precedenza, per ogni tipo di componente si possono aggiungere altre informazioni, sempre definite con una coppia di Tag di tipo *Chiave/Valore*¸ nella Tabella 5.4: Informazioni aggiuntive per ogni tipo di elemento sono mostrati le chiavi e le rispettive descrizioni di ogni proprietà aggiuntiva dei elementi

Elemento	Chiave	Descrizione
PSTextFieldSpecifier	IsSecure	Posso dare come valore a questo campo solo ***True*** o ***False*** e permette di nascondere la stringa contenuta ll'interno della casella di testo
	KeyboardType	Al click della casella di testo viene visualizzata un tipo di tastiera che può essere: numerica, alfanumerica o per la scrittura di Url.
	AutocapitalizationType	Campo che se abilitato con il valore ***True*** abilita l'auto riempimento della casella di testo
	AutoCorrectionType	Campo che se abilitato con il valore ***True*** attiva l'auto correzione della stringa contenuta nella casella di testo.
PSToggleSwitchSpecifier	TrueValue	Viene imposto come valore di default che il pulsante è sul valore ***True***.
	FalseValue	Viene imposto come valore di default che il pulsante è sul valore ***false***.
PSSliderSpecifier	MinimumValue	Si definisce il valore minimo del mio componente *Slider*
	MaximumValue	Si definisce il valore massimo del mio componente *Slider*
	MinimumValueImage	Si definisce il valore minimo che può assumere un immagine
	MaximumValueImage	Si definisce il valore massimo che può assumere un immagine
PSMultiValueSpecifier	Values	Si definisce la lista dei valori che può assumere questo componente
	Titles	Si definisce la lista dei titoli che che può assumere questo componente

Tabella 5.4: Informazioni aggiuntive per ogni tipo di elemento

Per capire meglio ciò che è stato descritto, viene riportato inseguito un esempio di codice:

```
<dict>
        <key>Type</key>
                <string>PSTextFieldSpecifier</string>
        <key>Title</key>
                <string>UserName</string>
        <key>Key</key>
                <string>userName</string>
        <key>DefaultValue</key>
                <string>apple</string>
        <key>AutocapitalizationType</key>
                <string>None</string>
        <key>AutocorrectionType</key>
                <string>No</string>
        <key>IsSecure</key>
                <false/>
        <key>KeyboardType</key>
                <string>Alphabet</string>
</dict>
```

5.2.2. Sviluppo della pagina delle Impostazioni per Android

Lo sviluppo di una pagina delle Impostazioni esterna all'applicazione è creata mediante istruzioni native del dispositivo. Tutto questo potrebbe essere evitato a favore di una pagina creata ad - oc tramite Titanium. Tuttavia, per mantenere una similitudine logica fra l'App di Android e quella di iOS, si opta di seguire lo stesso percorso scelto per lo sviluppo su dispositivi Apple.

Android per definire una pagina delle Preferenze utilizza un file di tipo ***xml*** chiamato ***preferences.xml*** e contenuto all'interno della directory di *root* del progetto nel seguente percorso: ***platform\android\res\xml***.

La pagina delle impostazioni può essere creata utilizzando cinque elementi:

Tag	Descrizione
`<CheckBoxPreference…>`	Una check box che può assumere solamente i valori ***True*** e ***Fale***.
`<ListPreference…>`	Una lista di scelte, mostrata all'interno di una casella di dialogo e composta da un gruppo di **radio button** dove solo un oggetto può essere selezionato.
`<EditTextPreference…>`	Componente che quando selezionato visualizza una finestra di dialogo contenente una casella di testo.

`<RingtonePreference…>`	Con questo componente vengono visualizzati tutte le suonerie che l'utente può impostare.
`<PreferenceCategory…>`	Componente che raggruppa al suo interno diversi elementi della pagina delle Preferenze

Tabella 5.5: Tipi di componenti per Android

Nella Tabella 5.5: Tipi di componenti viene mostrato il **Tag** per definire all'interno del codice l'elemento scelto. All'interno di ognuno di questi **Tag** devono obbligatoriamente essere definite tre informazioni: chiave, titolo e descrizione, in più può essere aggiunto anche il valore di defaul. Di seguito viene mostrata la lista di queste quattro informazioni comuni per ogni elemento:

1. **Android:key** → stringa che definisce univocamente l'elemento;
2. **Androi:title** → Titolo del componente, visualizzato all'interno della pagina;
3. **Android:summary** → è la descrizione del componente, visualizzata all'interno della pagina;
4. **Android:default** → campo non obbligatorio che contiene il valore di default dell'elemento.[26]

Adesso viene mostrato il codice della pagina delle Impostazioni sviluppata e la relativa immagine Figura 5.2: Pagina delle Impostazioni Android:

```
<?xml version="1.0" encoding="UTF-8"?>
<PreferenceScreen android:title="Preferences"
xmlns:android="http://schemas.android.com/apk/res/android">
  - <PreferenceCategory android:title="Authentication data">
        <EditTextPreference android:title="User Name"
            android:key="userName" android:defaultValue="apple"
            android:summary="Inserisci nome utente"/>
        <EditTextPreference android:title="Password"
            android:key="password" android:defaultValue="apple2011!"
            android:summary="Iserisci password"
            android:password="true"/>
        <EditTextPreference android:title="URL Server" android:key="url"
            android:defaultValue="http://demo.quix.it/webshare"
            android:summary="Inserisci URL del server"/>
    </PreferenceCategory>
  - <PreferenceCategory android:title="Numero massimo di documenti">
        <EditTextPreference android:title="Numero di documenti"
            android:key="NumDoc" android:defaultValue="10"
            android:summary="Numero massimo di documenti"
            android:inputType="number"/>
        <CheckBoxPreference android:title="Salva Password"
            android:key="savePassword" android:defaultValue="true"/>
    </PreferenceCategory>
</PreferenceScreen>
```

[26] **Android**, Preferences. Disponibile su http://developer.android.com/reference/android/preference/Preference.html

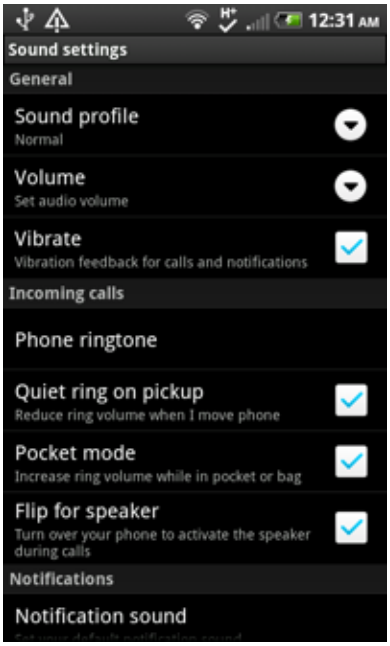


Figura 5.2: Pagina delle Impostazioni Android

5.2.3. Gestione dei dati con Titanium

Una volta creata la pagina delle Preferenze per ogni tipo di sistema operativo mobile il Framework Titanium, attraverso le sue API, permette di gestire queste impostazioni indipendentemente dal tipo di dispositivo, le funzioni implementate sono: la lettura delle impostazioni e la loro modifica. Di seguito vengono riportati alcuni esempi di gestione delle Preferenze.

```
/* Viene letta l'informazione dell'elemento con chiave uguale a 'userName'
*/
Titanium.App.Properties.getString('userName');

/* Viene scritto il valore 'Nome dell'utente' all'interno dell'elemento
 * della pagina delle Preferenze con chiave 'userName'
 */
Titanium.App.Properties.setString('userName', 'Nome dell'utente');
```

La grande differenza fra le due tipologie di dispositivi stà nella visualizzazione di questa pagina, infatti per Apple la si può aprire dalla sezione *Impostazioni* del dispositivo, mentre per Android l'apertura di questa pagina avviene mediante il click di un bottone presente all'interno dell'App. Nel caso di Web Share Mobile questo pulsante viene visualizzato solamente nella prima pagina dell'App (quella che contiene la lista degli archivi) quando viene cliccato il pulsante *Menù* del dispositivo.

```
/**
 * Funzione che visualizza all'interno di una finestra il pulsante delle
 * impostazioni, il quale quando viene premuto permetterà l'apertura
 * della suddetta pagina.
 */
activity.onCreateOptionsMenu = function(e) {
        var menu = e.menu;
        var menuSettings = menu.add({title: tabName.settings});
        //setto l'icona per il pulsante del menu
         menuSettings.setIcon("../Images/settings.png");

        menuSettings.addEventListener("click", function(e) {
        //viene aperta la pagina delle Impostazioni
                Titanium.UI.Android.openPreferences();
        });
};
```

Il problema principale riscontrato con questo metodo di sviluppo è la mancanza di controllo delle modifiche che un utente può apportare alle impostazioni dell'applicazione. Per sopperire a questa mancanza si fa ricorso a una Tabella del data base chiamata "user", la quale contiene le informazioni dell'utente.

I valori di questa tabella vengono paragonati a quelli delle impostazioni al avvio o alla riattivazione dell'App; così facendo vengono rilevate tutte le modifiche inlecite effettuate dall'utente.

I controlli che vengono effettuai sono i seguenti:

- Si controlla che non siano stati modificati User Name e URL del Server, se viene rilevata l'alterazione di uno di questi elementi vengono cancellate tutte le informazioni sui documenti e, se esistono, i relativi file. Al termine la Tabella "user" viene aggiornata con i nuovi valori delle impostazioni.
- Viene inoltre controllato se è stato disattivata la richiesta della password all'avvio dell'app. In caso afermativo viene richiesta un ulteriore volta la password per disattivarla definitivamente. Questo controllo è stato creato per evitare che una persona malintenzionata possa disattivare questo controllo.

5.3. Caratteristiche generali di Web Share Mobile

All'interno di questo capitolo è spiegata la struttura dell'App WS Mobile partendo dalla struttura dell'interfaccia grafica (**GUI**), proseguendo con lo sviluppo grafico di una finestra sia per dispositivi iOS che per Android, e terminando con l'implementazione della navigazione fra le varie finestre.

5.3.1. Struttura dell'interfaccia grafica

L'App è costituita su 3 finestre principali che si susseguono. Di seguito sono descritte le tre pagine che si mostreranno all'utente nell'ordine in cui sono descritte, vengono mostrate anche le relative immagini che si riferiscono, per motivi di spazio, solamente a un dispositivo iPhone.

- La finestra ***archiveList***, che visualizza la lista dei archivi alla quale appartiene almeno un documento Figura 5.3: archiveList;
- La finestra ***documentList***, la quale contiene tutti i documenti oppure tutti quelli appartenenti a un archivio selezionato all'interno della finestra ***archiveList*** Figura 5.4: documentList;
- La finestra ***documentDetail***, che mostra tutte le informazioni riguardanti a un documento selezionato precedentemente Figura 5.5: documentDetail.

Figura 5.3: archiveList

Figura 5.4: documentList

Figura 5.5: documentDetail

All'interno di WS Mobile esiste un'altra pagina che è costituita solamente da una componente di Titanium chiamata ***webView*** che visualizza semplicemente una pagina

http contenente la descrizione dell'App. Dato che questa pagina non è rilevante ai fini didattici e la ***webView*** sarà descritta nel capitolo riguardante la visualizzazione di un file con un dispositivo iOS, si è deciso di non descriverla.

Riconsiderando le tre pagine principali si può notare che hanno una struttura molto simile fra di loro, infatti sono costituite da tre elementi principali:

1. Una ***Navigation Bar*** che mostra all'utente quale pagina sta visualizzando e alla quale,volendo, si può aggiungere un pulsante, come per esempio il pulsante di aggiornamento contenuto nella finestra ***archiveList*** o quello di download di tutti i documenti in ***documentList***.

   Vengono anche aggiunti automaticamente dal Framework, all'interno della seconda e terza pagina dell'App, i pulsanti di navigazione che permettono di tornare alla pagina precedente. Questi pulsanti non sono presenti per dispositivi che supportano Android perché già vigenti in maniera fisica all'interno dell'apparecchio.
2. Una tabella, creata mediante l'**API** `createTableView()`, che contiene le informazioni.
3. La ***Tool Bar*** alla quale si possono aggiungere diversi elementi, ovvero viene mostrata una etichetta che visualizza la data dell'ultimo aggiornamento con il Server. Solo per dispositivi iOS l'etichetta viene sostituita da un indicatore di scaricamento, che mostra all'utente lo stato di avanzamento del download richiesto; al termine viene nuovamente visualizzata l'etichetta con la data aggiornata.

Nella seguente Figura 5.6: Composizione di una finestra vengono mostrati i vari componenti e le differenze grafiche fra dispositivi iOS e Android.

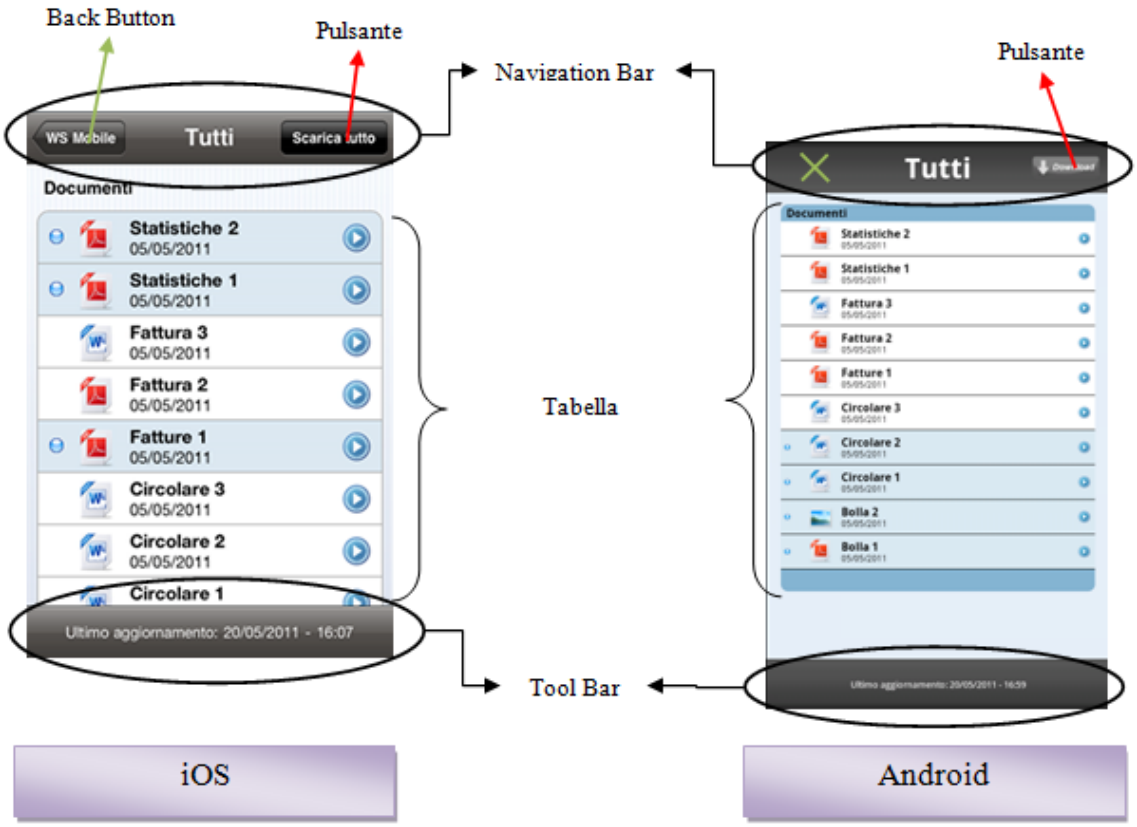


Figura 5.6: Composizione di una finestra

5.3.2. Implementazione della GUI su dispositivi iOS

Lo sviluppo di una finestra per dispositivi con Sistemo Operativi mobili iOS è molto semplice, infatti le due barre si inseriscono semplicemente tramite un istruzione di Titanium e mediante dei metodi specifici si inseriscono al loro interno vari componenti aggiuntivi. Come dichiarato in precedenza il pulsante per la navigazione viene inserito automaticamente dal Framework. Di seguito viene riportato il codice per gestire l'inserimento delle due barre:

```
// L'istruzione non nasconde la Navigation Bar
win.navBarHidden = false

// Visualizzazione della ToolBar nella pagina mediante un animazione
win.setToolbar([flexSpace,lblLastUpdate,flexSpace],{animated:true});

//Definizione e inserimento di un pulsante nella Navigation Bar
var cmddownloadAll = Titanium.UI.createButton ({
       title:button.downloadAll,
       style:Titanium.UI.iPhone.SystemButtonStyle.DONE
}); flexSpace
win.setRightNavButton(cmddownloadAll);
```

Come si può notare la *Navigation Bar* per visualizzarla basta specificare di non nasconderla anche se il più delle volte non è necessario perché di default la barra è già presente, per quanto riguarda l'aggiunta di un pulsante viene usato un metodo dell'oggetto `Titanium.UI.Windows` che inserisce automaticamente a destra o a sinistra della barra un oggetto bottone: più precisamente i due metodi sono: `setRightNavButton(pulsante)` e `setLeftNavButton(pulsante)`.

La stringa mostrata nella *Navigation Bar* viene preso dal titolo associato alla finestra.

La *Tool Bar* per essere mostrata nella finestra viene usato il metodo `setToolBar` che richiede un array di oggetti grafici e volendo l'aggiunta dell'animazione per rendere più piacevole la sua visualizzazione. Come si può vedere dal codice, all'interno dell'array è presente una etichetta, che mostra la data dell'ultimo aggiornamento, e un oggetto chiamato `flexSpace` presente due volte. Questo elemento è utilizzabile solo per iPhone ed è uno spazio vuoto da inserire fra due oggetti il quale si ridimensiona in modo da posizionare simmetricamente i vari oggetti che sono contenuti nella *Tool Bar*.

Viene mostrato il codice per definire questo oggetto speciale:

```
var flexSpace = Titanium.UI.createButton({
        systemButton:Titanium.UI.iPhone.SystemButton.FLEXIBLE_SPACE
});
```

Anche nel caso della tabella, nei dispositivi Apple, non ci sono difficoltà di implementazione, infatti problematiche come: l'altezza variabile della tabella, causata da un numero e da una dimensione delle righe non conosciuta a priori e l'aspetto grafico gradevole alla vista; sono già gestite dal Framework.

Di seguito viene mostrato il codice che riguarda la definizione della tabella.

```
/*Definizione della variabile tableviewListDoc che rappresenta l'oggetto
 *TableView
 */
var tableviewListDoc = Titanium.UI.createTableView({
        style:Titanium.UI.iPhone.TableViewStyle.GROUPED,
        backgroundColor:'transparent',
        top: 0,
        rowBackgroundColor:'white'
});
```

Il parametro `style:Titanium.UI.iPhone.TableViewStyle.GROUPED` permette di mostrare le varie righe della tabella raggruppate ma , soprattutto, visualizza di default una tabella con uno stile grafico bello da vedere.

5.3.3. Implementazione della GUI su dispositivi Android

Lo sviluppo dell'interfaccia grafica per dispositivi che supportano Android è molto più complicato per diversi fattori:

- Non esistono *Navigation Bar* o *Tool Bar* quindi devono essere sostituite da immagini che conterranno i vari componenti come bottoni e etichette;
- I pulsanti non hanno una bella grafica e si è dovuto creare un immagine da inserire nei bottoni;
- Difficoltà nel creare una tabella con altezza variabile
- Difficoltà di scorrere gli elementi della tabella in modo tale da non spostare o sovrastare la *Navigation Bar* e la *Tool Bar*
- Ci sono sei tipi di risoluzioni grafiche diverse;
- Mancanza di diversi effetti grafici rispetto dispositivi con iOS

Per tutte queste motivazioni all'interno del codice di ogni finestra è presente una parte di codice adibita allo sviluppo della GUI per la App di Android che viene mostrata di seguito.

```
if (Titanium.Platform.osname == 'android'){
//Creazione della Nav Bar e della Tool Bar tramite due viste
        var navBar = Titanium.UI.createView({
                height: "9%",top:0,
                backgroundImage:'../Images/barra_sup.png'});
        var toolBar = Titanium.UI.createView({
                height: "9%", bottom:0,
                backgroundImage:'../Images/barra_inf.png'});
//Creazione del pulsante con relativi immagini quando è premuto o no
        var cmddownloadAll = Titanium.UI.createButton ({
                backgroundSelectedImage:'../Images/Button/download_on.png',
                backgroundDisabledImage: '../Images/Button/download_off.png',
                width:94,height:31,   right:"3%"});
//Creazione dell'etichetta che contiene il titolo della finestra
        var titolo=Titanium.UI.createLabel({
                title: category,color:'white',font:{fontSize:40}});
//Creazione di una scrollView che contiene la tabella
        var scrollView=Titanium.UI.createScrollableView({
                views:[tableviewListDoc],
                top:"11%",width:"92%",height: "78%"});
//Si inserisce nella Nav Bar titolo e pulsante
        navBar.add(titolo);   navBar.add(cmddownloadAll);
//Si inserisce nella Tool Bar la data dell'ultimo aggiornamento
        toolBar.add(lblLastUpdate);
//inserimento dei vari componenti nella finestra principale
        win.add(toolBar);     win.add(navBar);  win.add(scrollView);
}
```

Oltre al codice appena mostrato è stata creata una funzione usata per ridimensionare ogni componente grafico a seconda della risoluzione del monitor. Solitamente questa

funzionalità viene garantita da Titanium tuttavia si è notato, durante i vari test su diversi dispositivi Android, che ha diversi problemi infatti in alcuni casi la grafica generata era pessima; questo era causato da errati ridimensionamenti dei componenti, mancanza delle linee che dividevano le varie righe della tabella oppure un alternanza di linee di dimensione 1 px e 2 px.

Per queste ragioni si è scelto di gestire mediante una funzione il riadattamento dell'App alle varie risoluzioni.

Di seguito viene mostrata la funzione implementata.

```
function setComponentDimension(screenHeight,page){
	if(screenHeight<480){
	//vengono settati i font delle etichette che contengono il titolo e
	//la data dell'ultimo aggiornamento
		if(page=="main"){
			//se richiamo la funzione dalla pagina archiveList.js
			//posiziono i pulsanti di update e info
		}
		if(page=="listaDocumenti"){
			//se richiamo la funzione dalla pagina listaDocumenti.js
			//posiziono il pulsante di download
		}
	}
	else{
		//stesso procedimento visto sopra per le altre risoluzioni
	}
	//setto le dimensioni della scrollview con la condizione che non
	//superi l'78% delle dimensioni della finestra questo per evitare che
	//la tabella non copra la Tool Bar
	var dimension;
	if(page !="Documento"){
		switch(screenHeight){
			//Si settano le dimensioni delle righe delle tabelle per
			//ogni tipo di display.
			//La dimensione della scrollView è data da il numero di
			//righe X l'altezza della riga
	if (dimension > 78) {
		scrollView.height = "78%";
	}
	else {
		scrollView.height = dimension+"%" ;
	}
}
```

La funzione mostrata permette, nella prima parte, di dimensionare i vari componenti grafici come etichette o bottoni, nella seconda viene dimensionato dinamicamente, a seconda nel numero di righe della tabella, il componente `scrollView` che la contiene.

5.3.4. Riempimento delle tabelle

L'operazione di riempimento delle tabelle è uguale per tutti i dispositivi che supportano come sistema operativo iOS e Android. Tutto questo è stato sviluppato mediante due funzioni principali: la prima che legge le informazioni dei documenti dal DB; la seconda che inserisce queste informazioni all'interno di componenti grafici che andranno a popolare una riga della tabella.

Di seguito viene riportato il codice delle due funzioni che creano la tabella della finestra ***documentList***. Funzioni molto simili sono state implementate anche per le altre pagine del software.

```
function readDB(){
        var rows;
        data.length=0;
        Titanium.API.info(category);
        try {
                if (category == label.main.allDoc) {
                        //Leggo le informazioni principali di tutti i documenti
                        rows = win.db.execute('SELECT * FROM Documenti
                                                ORDER BY ID DESC');
                }
                else {
                        //Leggo le informazioni principali dei documenti che
                        //appartengono a un archivio
                        rows = win.db.execute("SELECT * FROM Documenti
                                                WHERE ArchDesc=?
                                              ORDER BY ID DESC", category);
                }
                var i=0;
                        while (rows.isValidRow()){
                                createTable(rows.fieldByName('Title'),
                                        rows.fieldByName('RefDate'),
                                        rows.fieldByName('FileType'),
                                        rows.fieldByName('Downloaded'),
                                        rows.fieldByName('New'));
                                rows.next();
                        }
                rows.close();
                tableviewListDoc.setData(data);
                }catch(e){
                        Titanium.API.error("database error"+e);
        }
}
```

Questa funzione memorizza all'interno della variabile `rows` le informazioni ritornate dall'istruzione SQL eseguita. Dopo di che ogni riga del DB contenuta all'interno della variabile viene letta mediante un ciclo e a ogni iterazione viene richiamata la funzione `createTable()` che crea la riga da inserire nella tabella. Al termine del ciclo la tabella viene popolata da un oggetto, chiamato `data`, composto da tutte le righe create con la seconda funzione.

Proseguendo viene mostrato il codice della suddetta funzione.

```
function createTable(title,refdate,fileType,downloaded,newDoc){
        var index=data.length;
        var icona =  Titanium.UI.createImageView();
        var titolo = Titanium.UI.createLabel({text:title, …});
        var date = Titanium.UI.createLabel({text:refdate, …});

        //Se non sono state ancora lette le informazioni del documento
        //evidenzio la riga
        if(newDoc==1){
                row[index].backgroundColor="#dce9f2";
        }
        else{
                row[index].backgroundColor="white";
        }
        //se non ho scaricato il file mostro il titolo e la data in grassetto
        //e un pallino azzurro
        if(downloaded==0){
                row[index].leftImage="../Images/LedBlue.png";;
        }
        else{
                row[index].leftImage="../Images/empty.png";
        }
        //chiamo la funzione che setta l'icona corretta
        icona.image=setIcon(fileType);
        //aggiungo all'n-ima riga i vari componenti
        row[index].add(icona);
        row[index].add(titolo);
        row[index].add(date);
        row[index].hasChild=true;
        //Compponente della rig ache contine l'indirizzo della pagina figlia
       //che viene aperta al suo click
        row[index].test='documentDetail.js';
        row[index].backgroundSelectedColor = '#abd5f2';
        //Aggiungo all'array data l'oggetto riga
        data.push(row[index]);
}
```

Alla funzione `createTable()` vengono passata cinque dati: `title`; `refdate`; `fileType`; `downloaded`; `newDoc`. I primi due dati sono inseriti all'interno di due oggetti `Label` di Titanium, `downloaded` e `newDoc` sono usati per effettuare due controlli precisi:

- ✓ Se `newDoc==1` (non sono ancora state lette le informazioni del documento) allora evidenzio la riga, in caso contrario no
- ✓ Se `downloaded==0` (il file non è ancora stato scaricato) allora mostro a destra della riga un immagine che rappresenta un pallino blu, in caso contrario mostro un immagine vuota

La variabile `fileType` viene passata ad un ulteriore funzione `setIcon(fileType)`, la quale restituisce all'oggetto `icona.image` l'url che fa riferimento alla posizione dell'immagine che descrive il tipo di file (file word, excel, pdf, ecc).

In fine tutti questi elementi vengono inseriti all'interno di una riga che andrà successivamente a comporre l'array `data`.

5.4. Gestione delle informazioni dei documenti

Le informazioni dei documenti sono gestite da tre funzioni principali che rispettivamente permettono di: acquisire l'oggetto JSON inviato dal server, interpretarlo e memorizzare le informazioni in esso contenute all'interno del data base e cancellare le informazioni, con gli eventuali documenti scaricati, che superano la soglia specificata dall'utente.

5.4.1. Acquisizione dell'oggetto JSON

La seguente funzione ha il compito di richiedere al Server, mediante una richiesta http, le informazioni, contenute all'interno di un oggetto JSON, relative ai documenti più recenti. Questa richiesta per essere soddisfatta deve inviare al Server nome utente, password (per l'autenticazione) e il numero di documenti dei quali vuole conoscere le informazioni, se tutto questo ha successo il Server invia a WS Mobile l'oggetto JSON. Al termine del download da parte del Client dell'oggetto richiesto, viene richiamato un metodo `onload` della richiesta HTTP il quale richiama in sequenza due funzioni principali: la prima, `Update_DataBase`, che aggiorna il DB con le nuove informazioni; la seconda, `deleteDocuments`, che elimina i documenti e le rispettive informazioni che superano il limite imposto dall'utente.
In caso in qui si verifica un errore nella comunicazione tra Client e Server viene invocato il metodo `onerror` il quale mostra, all'interno di una finestra di pop up, il messaggio di errore; per esempio l'errore 401 che indica che i dati per l'autenticazione sono errati oppure l'errore 503 che determina l'inaccessibilità del Server.
Di seguito viene mostrato il codice relativo alla funzione tratta in questo capitolo.

```
function Create_Connection(username, pwd, url, NumDoc){
	var httpRequest = function(params){
		var xhr = Titanium.Network.createHTTPClient();
		var nMaxDoc=parseInt(NumDoc);
		xhr.setTimeout(30000);
		xhr.onload = function(){
			Titanium.API.info('http connection ONLOAD');
			if (params.hasOwnProperty('callback')) {
				try {
					params.callback(this.responseText);
					var json = JSON.parse(this.responseText);
					//funzione che aggiorna il DB
					Update_DataBase(json, params.urlServer);
				//calcolo il numero di elementi contenuti nel DB
					totDoc=numberDocuments();
					if (totDoc > nMaxDoc) {
						var differenza = totDoc - nMaxDoc;
						//funzione che elimina gli elementi in
						//eccesso ne DB
						deleteDocuments(differenza);
					}
					createTable();
				}
				catch (e) {
					Ti.API.error(this.status + error.noFile);
				}
			}
		};
		xhr.onerror = function(){
			//Visualizzo un messaggio di errore
		};
		xhr.open("POST", params.urlServer);
		xhr.send(params);
	};

	//Call it like this
	httpRequest({
		urlServer: url,
		user: username,
		password: pwd,
		numDocument: NumDoc,
		callback: function(resp){
		}
	});
}
```

5.4.2. Aggiornamento delle informazioni nel DB

Le informazioni contenute all'interno della tabella *Documents* sono inserite o aggiornate tramite la funzione `Update_DataBase`, la quale è richiamata dopo il corretto download dell'oggetto JSON all'interno del suddetto metodo, `Create_Connections`.

`Update_DataBase` ha come parametri di ingresso: il JSON e l'url del server, quest'ultimo viene utilizzato per calcolare l'url che permette la localizzazione del file desiderato. L'oggetto JSON invece viene scorso tramite un ciclo e a ogni iterazione

controlla se le informazioni che fanno riferimento a un documento sono nuove o no; nel primo caso questi dati sono inseriti nella tabella del DB altrimenti, in caso contrario, si aggiornano le informazioni. Inoltre viene verificato se il file è stato scaricato (`Downloaded==1`) e modificato (dimensioni o nome del file diversi da quello presente nel dispositivo), il documento viene cancellato.

Al termine della lettura della prima parte del JSON, che contiene le informazioni sui documenti, viene letta una seconda parte che contiene le descrizioni degli archivi. Questi dati vengono associati a ogni documento a seconda del archivio alla quale appartengono.

```
function Update_DataBase(json, urlServer){
	//Riempio la tabella Documenti nel DB lato client
	for (var i = 0; i < json.DocumentList.length; i++) {
	//Controllo se le informazioni del documento sono gia' presenti nel DB
		var dbrows=win.db.execute("SELECT COUNT(*) AS esiste,… WHERE ID=?",
								json.DocumentList[i].id);

		if (dbrows.fieldByName('esiste') == null) {
		//Le informazioni non esistono e le inserisco nel DB

			win.db.execute('INSERT INTO Documenti(...)...');
		}
		else{
		// Le informazioni esistono quindi aggiorno il DB
			if (dbrows.fieldByName('Downloaded') == 1 &&
				(dbrows.fieldByName('FileSize')!=size ||
				dbrows.fieldByName('FileName')!=
								json.DocumentList[i].file)) {
			// Se il file è stato scaricato e non è aggiornato lo cancello
				Cancello il file
			}
		// Aggiorno le informazioni
		win.db.execute("UPDATE Documenti SET...");

		}
	}
	//Aggiungo la descrizione dei archivi contenua nella seconda parte del JSON
	for (var j = 0; j < json.ArchiveList.length; j++) {
		win.db.execute("UPDATE Documenti SET ArchDesc=? WHERE Archive=?",
								json.ArchiveList[j].description,
								json.ArchiveList[j].code);
	}
}
```

5.4.3. Cancellazione delle informazioni e dei relativi file in eccesso

La cancellazione delle informazioni meno recenti è implementata mediante la funzione `deleteDocuments(differenza)`. Questa funzione vine richiamata dopo l'aggiornamento del DB all'interno della funzione `Create_Connections` nel caso in cui

il numero di record contenuti all'interno della tabella *Documents*, la variabile `totDoc` e calcolata dalla funzione `numberDocuments()`, superi la soglia specificata dall'utente (`nMaxDoc`).

Al metodo adibito alla cancellazione dei record viene passato il parametro `differenza` che contiene il valore ottenuto dalla differenza fra le variabili `totDoc` e `nMaxDoc`, agendo in questo modo si riesce a conoscere il numero di elementi da eliminare per ritornare sotto la soglia definita.

Di seguito viene mostrato il codice della funzione.

```
function deleteDocuments(differenza){
	try {
	//Acquisisco tutte le informazioni in eccesso più vecchie
		var d = win.db.execute("SELECT ID, FileName, FileType FROM
				Documenti ORDER BY ID LIMIT ?", differenza);
		while (d.isValidRow()) {
			var id = d.fieldByName('ID');
			var filename = d.fieldByName('ID') + '-' +
					d.fieldByName('FileName');
			var file;
			if (Titanium.Platform.osname == 'iphone') {
				file = Titanium.Filesystem.getFile(
				   Titanium.Filesystem.applicationDataDirectory,
				  filename);
			}
			else{
				file = Titanium.Filesystem.getFile(
					webShareFolder.nativePath, filename);
			}
		//Controllo se il file esiste, in caso affermativo lo cancello
			if (file.exists()) {
				file.deleteFile();
			}
			//elimino il record in ecceso
			win.db.execute("DELETE FROM Documenti WHERE ID = ?",
					id);
			d.next();
		}
	}catch(err){
		Titanium.API.error("DB Error"+err);
	}
}
```

Entrando più nel particolare, questa funzione legge dal DB tre informazioni: l'id, il nome e il tipo di file; per un numero di documenti che raggiunge il valore dato dal parametro `differenza`. Dopo di che viene ricavato il nome del file e cercato all'interno del File System, se esiste viene eliminato altrimenti si passa direttamente alla fase di cancellazione del record all'interno della tabella *Documents*. Tutto questo procedimento viene seguito per ogni documento vecchio che supera il limite imposto.

5.5. Gestione dei File

Il seguente paragrafo spiega come viene gestito lo scaricamento di un file e la sua visualizzazione. Nel primo caso non esistono differenza di codice fra i dispositivi, mentre nel secondo caso ci sono due sviluppi differenti a seconda della piattaforma usata.

5.5.1. Download del File

Il download di uno o più file è gestito da una funzione chiamata `Download_File()`, la quale ha come parametri di ingresso i seguenti valori:

- ID → identificativo del documento;
- URL → url del server per scaricare il file;
- fileName → Nome del file;
- fileType → Tipo di file;
- open_file → numero di documenti da aprire;

La funzione, una volta definito l'oggetto che genera la barra per visualizzare lo stato del download (per iOS si usa l'oggetto `ProgressBar` mentre per Android `l'ActivityIndicator`), crea una richiesta http, all'interno della quale è presente l'url del server per scaricare il file, il nome utente e la password per l'autenticazione del richiedente. Al termine dello scaricamento, in caso di successo, il file viene memorizzato e se è stato richiesto il download di un singolo file, esso viene anche visualizzato. Questo avviene se la funzione viene richiamata dalla finestra *documentDetail*.

Per gestire lo scaricamento di più file, che può essere richiesto all'interno della finestra *documentList*, l'App richiama questa funzione tante volte quanti sono i documenti da scaricare e al termine aggiorna la pagina visualizzata.

Di seguito viene riportata la parte della richiesta http che viene richiamata in caso di successo.

```
function Download_File(ID, URL, fileName, fileType, open_file){
	…
	var httpRequest = function(params){
		//creazione connessione http
		var c = Titanium.Network.createHTTPClient();
		c.setTimeout(30000);
		c.onload = function(){
			if (params.hasOwnProperty('callback')) {
				try {
					if(Titanium.Platform.osname == 'android'){
						//Memorizzazione su Android
						var webShareFolder =
							Titanium.Filesystem.getFile(
							Titanium.Filesystem.
								externalStorageDirectory,
							  "WebShareDoc");
						var f = Titanium.Filesystem.getFile(
							webShareFolder.nativePath,
							filename);
						f.write(this.responseData);
			}
					//aggiorno il DB
					if (open_file == 1) {
						View_File(ID, fileName, fileType);
					}
				}
				catch (er) {
						Ti.API.error(this.status +
							' - File not found'+er);
					}
				}
				barraDownload.hide();
			}
		};
		// open the client
		c.open('POST', params.url);
		…
		if(Titanium.Platform.osname == 'iphone'){
			//Memorizzazione su iOS
			c.file = Titanium.Filesystem.getFile(
				Titanium.Filesystem.applicationDataDirectory,
				 filename);
		}
		c.send(params);
	};
		httpRequest( {
			url: URL,
			user: userName,
			password: pwd,
		callback:  function(resp) {
		}
	});
}
```

Come si può notare dal codice l'unica differenza fre i dispositivi iOS e Android è nella memorizzazione del documento. Nel primo caso il file deve essere memorizzato all'interno della cartella nella quale è installata l'App, ciò avviene tramite un metodo dell'oggetto che gestisce la chiamata http. Nel secondo caso invece, il documento deve

essere memorizzato al di fuori della cartella che contiene l'App. In questo caso si è scelto di memorizzarli sulla scheda SD presente in tutti i smartphone che supportano Android, e il salvataggio avviene grazie una funzione di gestione dei file di Titanium.

5.5.2. Visualizzazione del documento

La visualizzazione di un documento risulta completamente differente per le varie piattaforme. Per iOS una qualsiasi tipologia di documento può essere visualizzata all'interno di un componente di Titanium chiamato *web view*, mentre per i dispositivi basati su Android i vari tipi di file devono essere aperti con App esterne e già installate. Questo requisito viene gestito dalla funzione `View_File(ID, fileName, fileType)` richiamata o al termine del download di un file o all'interno della pagina *documentDetail*, all'evento click della prima riga della tabella se il documento è già presente all'interno del File System.

```
function View_File(ID,NomeFile,estensioneFile){
	var filename = ID + '-' + NomeFile;
	var f;
	if(Titanium.Platform.osname=='iphone'){
		f = Titanium.Filesystem.getFile(
			Titanium.Filesystem.applicationDataDirectory, filename);
		Titanium.API.info(f.nativePath);
	//Creo una nuova finestra che permettera' la visualizzazione del file
		var ViewDoc = Titanium.UI.createWindow({
			url: '../main_windows/documentView.js',
		//passo ilpercorso del file che deve essere visualizzato
			file: f,
		});
		Titanium.UI.currentTab.open(ViewDoc,{animated:true});
	}
	else{
		var webShareFolder = Titanium.Filesystem.getFile(
			Titanium.Filesystem.externalStorageDirectory,
			"WebShareDoc");
		f = Titanium.Filesystem.getFile(
			webShareFolder.nativePath, filename);
		if (Ti.Platform.openURLWithType(
			f.nativePath, getContentType(estensioneFile))) {
			// success!
		}
		else {
		  alert(error.fileNotSupported);
		}
	}
}
```

Di seguito vengono esposte le due differenti implementazioni.

❖ *Visualizzazione di un documento su iPhone, iPod e iPad*

In questo caso la visualizzazione di un documento è molto semplice, infatti una volta aperta la nuova pagina, definita dalla funzione precedentemente mostrata, basta mettere come campo *url* del componente *web view* la posizione del file nel dispositivo.

In questo caso la posizione la si acquisisce tramite la variabile `file` passata al momento della creazione della nuova finestra.

```
var win = Titanium.UI.currentWindow;
var wv = Ti.UI.createWebView({
       url: win.file.nativePath
});
win.add(wv);
```

La *web view*, per iOS, è un componente molto potente, infatti grazie ad esso si elimina ogni problema di eterogeneità per la visualizzazione di ogni tipo di documento, proprio indicando solo la posizione del file, si riesce ad aprire qualsiasi tipologia di documenti; inoltre è ingrado di visualizzare anche pagine HTML locali o veri e propri siti Web.

❖ *Visualizzazione di un documento su qualsiasi dispositivo Android*

La visualizzazione di un documento su Android è molto più complessa, infatti Titanium non fornisce una funzione in grado di soddisfare questo requisito, la *web view* non ha le stesse potenzialità rispetto a quando è usata su un dispositivo iOS.

Per provvedere a questa mancanza si è dovuto creare un modulo aggiuntivo da inserire all'interno del Framework. Quindi si è dovuto scaricare il codice sorgente dell'SDK di Titanium e all'suo interno è stata aggiunta la funzione `openURLWithType(<path del file>,<tipo di file>)`, che viene mostrata di seguito, e infine è stato tutto ricompilato. L'esatto procedimento per la ricompilazione è descritto all'interno del seguente link: http://developer.appcelerator.com/doc/mobile/mobile-build-osx.

```
public boolean openURLWithType(String url, String type) {
       if (DBG) {
              Log.d(LCAT, "Launching viewer for: " + url); }
       Uri uri = Uri.parse(url);
       Intent intent = new Intent(Intent.ACTION_VIEW, uri);
       intent.setDataAndType(uri, type);
       intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
       try {
              getTiContext().getActivity().startActivity(intent);
              return true;
       }catch (ActivityNotFoundException e){
              Log.e(LCAT,"Activity not found: " + url, e); }
       return false;
}
```

Questa funzione, data la posizione e il suo tipo di file, permette di aprire i documenti mediante un'applicazione esterna, che deve essere già presente all'interno dello smartphone o del tablet; per esempio per aprire un file PDF si deve installare prima una app tipo *PDF Reader*. Se la funzione non riesce ad aprire il file viene mostrato un messaggio di errore.

Per ricavare l'esatta definizione di una determinata tipologia di file è stata sviluppata la funzione `getContentType(estensioneFile)` che restituisce la il valore richiesto dato in input l'estensione del file che si desidera aprire. Di seguito viene mostrato il codice:

```
function getContentType(fileExtension){
	switch(fileType){
		case 'doc':
			return 'application/msword';
		case 'pdf':
			return 'application/pdf';
		…
	}
}
```

5.6. Invio di un File tramite Email

Nel seguente paragrafo viene mostrata la funzione che implementa l'invio di una email con allegato il file che l'utente sta visualizzando. La seguente funzione viene usata da tutti i dispositivi che supportano sia iOS che Android. L'unico vincolo per il suo corretto funzionamento è l'esistenza di una casella di posta elettronica all'interno del **Mail User Agent** (**MUA**) del dispositivi

Di seguito viene mostrato il codice:

```
function createEmail(subject,recipients,attachment){
	if (Titanium.Network.online) {
		var emailDialog = Titanium.UI.createEmailDialog();
		emailDialog.subject = subject;
		emailDialog.toRecipients=recipients;
		emailDialog.addAttachment(attachment);
		emailDialog.addEventListener('complete', function(e){
		});
		emailDialog.open();
	}
	else {
		alertDialog.title = error.err;
		alertDialog.message = error.noConnection;
		alertDialog.show();
		Titanium.API.error(error.noConnection);
	}
}
```

Come si può notare alla funzione createEmail vengono passati tre parametri contenenti: il soggetto, i riceventi e la posizione dell'allegato all'interno del File System; i quali sono passati a un oggetto di Titanium chiamato: `createEmailDialog()`.
Questo componente del Framework aprirà la finestra di creazione di una nuova email del **MUA** con i campi già precompilati.
Tutto questo viene effettuato se il telefonino è connesso a internet e se il file è stato scaricato correttamente, in caso contrario verrà visualizzato un messaggio di errore.

Conclusioni

Concludendo, al termine del periodo di stage, i requisiti funzionali richiesti per lo sviluppo dell'applicativo sono stati completamente soddisfatti.

Le maggiori difficoltà riscontrate sono state causate dal fatto che Titanium è un Framework ancora in fase di sviluppo, infatti è ancora privo di alcuni moduli e di componenti grafici per i dispositivi Android, come il modulo per l'apertura dei file e le *Tool* o *Navigation Bar*. Inoltre durante la fase di sviluppo dell'applicativo la casa produttrice di Titanium non aveva ancora rilasciato un editor con tutti i vari componenti per l'aiuto alla programmazione, come per esempio l'auto completamento delle istruzioni e il debug, questa mancanza è stata risolta pochi mesi fa con l'uscita dell'editor **Titanium Studio**.

In definitiva ritengo che Titanium sia un ottimo Framework per lo sviluppo di applicazioni mobili multipiattaforma perché oltre a garantire la creazione di applicazioni native è un prodotto in rapida evoluzione (sei aggiornamenti delle SDK di Titanium in sei mesi) e un ottimo supporto on-line per i programmatori. Tutto ciò lascia intendere a un incessante investimento lavorativo rivolto a questo prodotto che in questo modo risulta qualitativamente valido.

Sviluppi futuri

L'implementazione dell'applicativo ha garantito la piena efficienza e la piena utilizzabilità da parte degli utenti finali. Tuttavia è possibile migliorare ulteriormente il software effettuando le seguenti modifiche:

- Gestire i documenti di più utenti che utilizzano lo stesso dispositivo;
- Implementare una gestione documentale più avanzata permettendo la cancellazione di singoli elementi scelti dall'utente;
- Gestire, oltre che i documenti, anche gli archivi;
- Creare un modulo che permette di caricare i documenti sul Server;
- Scaricare le informazioni sui nuovi documenti in modalità push;
- Implementare l'applicativo Web Share Mobile anche per dispositivi Blackberry.

Glossario

DBMS SQL: è un sistema software progettato per consentire la creazione e la manipolazione efficiente di database (ovvero una collezione di dati strutturati) solitamente da parte di più utenti.

ACID: deriva dall'acronimo inglese **Atomicity**, **Consistency**, **Isolation**, e **Durability** (**A**tomicità, **C**oerenza, **I**solamento e **D**urabilità). Queste quattro proprietà devono essere rispettate per fare in modo che le transazioni operino in modo corretto sui dati.

Basic Authentication: all'interno di una transazione http, la *basic access authentication* è un metodo usato dal Web browser, o da un altro programma Client, di fornire nome utente e password all'interno di una richiesta per effettuare un autenticazione. Prima della trasmissione il nome utente viene aggiunto con i due punti e concatenato con la password, alla fine la stringa risultante viene codificata con l'algoritmo di codifica *Base64.*

DOM: Il **Document Object Model** (spesso abbreviato come **DOM**), letteralmente *modello a oggetti del documento*, è una forma di rappresentazione dei documenti strutturati come modello orientato agli oggetti. DOM è lo standard ufficiale del W3C per la rappresentazione di documenti strutturati in maniera da essere neutrali sia per la lingua che per la piattaforma.

Mail User Agent (MUA): è un programma che consente di gestire la composizione, la trasmissione, la ricezione e l'organizzazione di email (i messaggi di posta elettronica) da e verso un Server di posta.

Scheda SD: Secure Digital (chiamate più brevemente SD) è il più diffuso formato di schede di memoria, dispositivi elettronici utilizzati per memorizzare in formato digitale grandi quantità di informazioni all'interno di memorie flash.

Bibliografia

- **Appcelerator**, Documentazione on-line di Appcelerator Titanium
  http://developer.appcelerator.com/apidoc/mobile/latest
- **Appcelerator**, Forum per l'aiuto alla programmazione
  http://developer.appcelerator.com/questions/newest
- **Appcelerator**, Manuali
  http://developer.appcelerator.com/doc/mobile/guides
- **Android**, Portale Android per gli sviluppatori
  http://developer.android.com/index.html
- **Android**, Guida allo sviluppo
  http://developer.android.com/guide/index.html
- **Android**, Pubblicazione su Android Market
  http://developer.android.com/guide/publishing/publishing.html
- **Apple**, Portale Apple per gli sviluppatori
  https://developer.apple.com/ios/manage/overview/index.action
- **Apple**, Guida allo sviluppo
  http://developer.apple.com/devcenter/ios/index.action
- **Apple**, Pubblicazione su Apple Store
  https://itunesconnect.apple.com/docs/iTunesConnect_DeveloperGuide.pdf
- **JQuery**, JQuery Mobile
  http://jquerymobile.com/
- **Autori Vari**, Wikipedia
  http://www.wikipedia.it
- **Autori Vari**, Svillupare app con Appcelerator Titanium
  http://www.appuntisoftware.it/appcelerator-titanium-sviluppare-app-per-android/

Ringraziamenti

Ringrazio il **Chiar.mo Prof. Sonia Bergamaschi** e il **Prof. Riccardo Martoglia** per l'aiuto e la Loro disponibilità durante la realizzazione del progetto e la stesura della tesi.

Ringrazio il mio tutor aziendale, **Andrea Prandini**, e tutti gli altri colleghi, in particolare **Roberto Pellegrino** e **Matteo Sironi**, per avermi ospitato nella loro azienda e per avermi aiutato durante la progettazione e lo sviluppo del mio applicativo.

Ringrazio la mia **famiglia** per avermi sopportato e supportato in tutti questi anni e ne avranno ancora.

Ringrazio tutti i miei amici dell'università **Peva**, **Cally** e **Lazza**.

Ringrazio i miei amici di una vita **Michi**, **Marci**, **Alle** e **Anto** per le magnifiche estati passate a Pinarella di Cervia, ma soprattutto perché mi sopportano e mi aiutano da quando eravamo insieme alle elementari (per **Michi** dalla scuola materna).

Ringrazio tutti i miei amici con qui passo il mio tempo libero **Matte**, **Alle, Nik, Dommi**, **Arletti**, **Pello** e **Fecchia.**

Un particolare ringraziamento alla mia ragazza **Giulia Caiumi** per avermi sostenuto per tutti questi mesi, per sopportarmi e per avermi aiutato nella correzione della tesi.

E infine ringrazio me stesso che al di sopra di ogni aspettativa sono riuscito ad arrivare in fondo a tutti e cinque gli anni di Ingegneria Informatica.