ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

Polo scientifico-didattico di Cesena

FACOLTÀ DI SCIENZE MATEMATICHE FISCHE NATURALI

CORSO DI LAUREA IN

SCIENZE DELL'INFORMAZIONE

PROGETTAZIONE E SVILUPPO DI UNA APPLICAZIONE MOBILE PER AGENTI E RAPPRESENTANTI

Tesi di Laurea in

ALGORITIMI E STRUTTURE DATI

Relatore
Prof. Luciano Margara

Tesi presentata da
Fabio Galeppi

Sessione II

Anno accademico 2012-2013

Sommario

Introduzione

Molte aziende hanno la necessità di avere uno strumento completo e performante che gli consente di salvare e sincronizzare il lavoro che ogni agente quotidianamente svolge; questo deve essere il più possibile multi-piattaforma e facile da usare. Occorre pertanto pensare, definire e sviluppare un prodotto all'altezza e che possa essere evoluto, ovvero facilmente esteso secondo le esperienze degli agenti e rappresentanti. E' stato concepito e realizzato secondo le esigenze dell'azienda, in modo da renderlo il più efficace possibile per risolvere i problemi ed essere una soluzione per più classi di agenti. L'unione tra l'azienda sviluppatrice del software e gli agenti che ne fanno utilizzo, permette di crescere insieme verso un prodotto di qualità, sempre più completo e personalizzabile.

Obiettivo di questa tesi è descrivere un percorso di creazione di una applicazione mobile per la realizzazione di un software i cui target sono smartphone e tablet.

Progettare una applicazione di questo genere richiede competenze in campo informatico per quanto riguarda la realizzazione finale, ma anche una figura che sappia elencare tutti gli strumenti di cui deve disporre, ecco perché l'utente finale è molto importante.

Per evitare allo sviluppatore lo sforzo di creare una applicazione mono ambito è obbligatorio creare un codice applicativo già funzionante e modellabile con semplici modifiche.

Dal frutto di questo lavoro è nato Order Sender, un'App per agenti di commercio presente sui mercati delle piattaforme mobile Android ed iOS.

Nel primo capitolo si tratteranno gli aspetti della programmazione su dispositivi mobile basati su Android, facendone una rapida descrizione. Nel secondo sarà fatta una analisi del progetto e degli esempi di utilizzo. Nel terzo capitolo verranno spiegate le

scelte fatte per avere un front-end il più possibile intuitivo ed efficace. Nel quarto saranno spiegate tutte le interfacce in possesso dell'agente per creare e manipolare i dati. Nel quinto sarà spiegato il servizio web associato all'applicazione e le potenzialità aggiuntive offerte. Nel sesto sarà spiegato tutto quello che è il back-end dell'applicazione, passando dal database locale all'interazione con il database remoto, visionando la gestione della cache e una caratteristica importante, la sincronizzazione bidirezionale. Nel settimo e ultimo capito verranno esposte delle conclusioni e dei possibili sviluppi futuri.

1 Sviluppo di applicazioni su mobile

1.1 Ambiente di sviluppo

Sviluppare applicazione per dispositivi mobile significa fornire all'utente finale uno strumento utile sempre a portata di mano. Un dispositivo può avere diverse caratteristiche a seconda dell'utilizzo che ne si deve fare. Nelle categorie principali ci sono smartphone e tablet. Questi dispositivi, con le giuste applicazioni, possono sostituire quasi completamente un pc, ma con qualche vantaggio in più, quali:

- Dimensioni ridotte: questo permette all'utente finale di poter utilizzare queste tecnologie ovunque e in modo veloce, tuttavia date le dimensioni, la vista delle applicazioni è limitata alla grandezza del display.
- Schermo touchscreen: fino a posto tempo fa, si poteva controllare il pc solamente con mouse e tastiera, poi con l'avvento degli schermi touch, è possibile interagire con il sistema operativo anche effettuando dei tocchi allo schermo. Questo è un punto di forza dei dispositivi mobile, la possibilità di andare direttamente e con pochi tocchi dove si vuole.
- Connessione ovunque: tablet e smartphone hanno la possibilità di collegarsi a Internet sia tramite Wi-Fi sia tramite rete di operatori (GPRS,EDGE,3G,LTE,4G).
- Fotocamera: avere la possibilità di scattare foto dove e quando si vuole è un grosso vantaggio.

Ma possiede anche dei contro:

- Caratteristiche hardware limitate: di smartphone ci sono molti modelli per tutte le fasce di prezzo, questo può essere una nota positiva se si prende un telefono di

punta, ma può anche essere una limitazione se invece si sceglie di acquistarne uno di fase medio-bassa, questo inciderà anche sulla fluidità dell'applicazione che andrà a eseguirsi sopra. Ecco perché un bravo sviluppatore deve cercare di supportare al meglio la maggior parte di dispositivi mobile presenti sul mercato. La gestione della RAM è uno dei punti fondamentali della programmazione mobile.

- Batteria: per molti dispositivi, anche top di gamma, la batteria è il proprio tallone d'Achille, perché è difficile dotare il telefono di una batteria sufficientemente potente per permettere di non ricaricarlo per più di due-tre giorni consecutivi. Le batterie, ovviamente, devono essere piccole e leggere e questo indica una capacità limitata. Ecco un altro punto su cui lo sviluppatore ha una grossa responsabilità, perché meno risorse vengono usate, meno viene a lavorare il terminale.

1.2 Piattaforma Android

Android è un sistema operativo per dispositivi mobile organizzato in un'architettura software ideata da Google Inc. che include un sistema operativo di base, i middleware per le comunicazioni e le applicazioni di base. Android è caratterizzato da una struttura open source e si basa su kernel Linux. Essendo Open Source è possibile scaricare e modificare liberamente il codice sorgente, sia che sia stato modificato dalla casa produttrice del device, sia che sia originale.

Esistono anche delle versioni non originali, che permettono di personalizzare a pieno il proprio dispositivo.

Questi fattori hanno permesso ad Android di diventare il sistema operativo mobile più diffuso al mondo, grazie alla sua vasta gamma di dispositivi compatibili.

La prima versione di Android è stata rilasciata nel 2008 e conteneva solamente le applicazioni base, quali: market, browser, posta, gestione della memoria di massa, supporto al Wi-Fi e fotocamera. Ogni anno seguono continui aggiornamenti di versione del sistema operativo e, dopo una serie di aggiornamenti, una nuova release.

Queste permettono ai dispositivi di tenersi in continuo aggiornamento, ove sia supportato dallo smartphone.

Siccome il sistema è basato sul linguaggio Java, i programmi sono eseguiti su una macchina virtuale. La macchina virtuale che viene utilizzata su Android non è la

classica Java Virtual Machine, ma una macchina virtuale ottimizzata per eseguire su dispositivi con scarse risorse hardware. Questa è detta Dalvik Virtual Machine.

La Dalvik VM è dotata di un garbage collector perciò la gestione della memoria non deve essere direttamente gestita dal programmatore.

Ogni applicazione viene eseguita in un processo a sé stante, con una propria istanza della Dalvik VM. Ciò permette di implementare il meccanismo del minimo privilegio, così che le applicazioni abbiano accesso solo alle risorse che gli sono esplicitamente concesse.

Il livello più basso è costituito dal kernel Linux che offre anche i driver per l'hardware sottostante, permettendo di astrarre dalle caratteristiche specifiche del dispositivo.

Le librerie sono a disposizione del programmatore e permettono di sfruttare le caratteristiche della piattaforma, quali l'uso di SQLite, le OpenGL, supporto per i file multimediali, l'uso della telefonia e di tutti i componenti hardware di cui il dispositivo può essere dotato.

L'ambiente runtime di Android permette di implementare il meccanismo per il quale ogni nuova applicazione viene avviata in un processo a parte con la propria istanza della Dalvik Virtual Machine.

L'application framework contiene tutti i componenti di base per lo sviluppo di applicazioni.

Infine nell'ultimo livello abbiamo le applicazioni, che possono essere già presenti nella piattaforma o sviluppate successivamente da terzi.

1.2.1 La panoramica delle versioni di Android:

Ci sono diverse versioni di Android in circolazione. Si riportano brevemente le varie versioni della piattaforma esistenti, con i corrispondenti API level, ovvero l'intero che la identifica univocamente. Esso viene utilizzato all'interno delle applicazioni per indicare quale è la piattaforma su cui un'App è stata progettata per funzionare.

- 1.0 – Bender – API 1 – 2008
- 1.1 – Petit Four – API 2 – 2009
- 1.5 – Cupcake – API 3 – 2009
- 1.6 – Donut – API 4 – 2009
- 2.0 – Eclair – API 5 – 2009
- 2.1 – Eclair – API 6 – 2010
- 2.2 – Froyo – API 8 – 2010

- 2.3 – Gingerbread – API 9 – 2010
- 3.0 – Honeycomb – API 11 – 2011 – Versione ottimizzata per tablet
- 4.0 – Ice Cream Sandwich – API 14 – 2011
- 4.1 – Jelly Bean – API 16 – 2012
- 4.2 – Jelly Bean – API 17 – 2012
- 4.3 – Jelly Bean – API 18 – 2013

Google ad ogni uscita di una nuova versione rilascia il relativo SDK per permettere agli sviluppatori di creare applicazioni compatibili. Tramite Eclipse, un ambiente di sviluppo integrato multi-linguaggio e multipiattaforma, è possibile programmare in Java e quindi per Android. Utilizzando Android Development Tools (ADT), un plugin per l'IDE Eclipse, è possibile disporre di un potente ambiente integrato in cui costruire le applicazioni.

1.2.2 Java, Android e le applicazioni

Per creare applicazioni su Android si parte dal linguaggio Java, che viene poi esteso con una serie di librerie apposite per la programmazione su mobile e per gestire una serie di aspetti, quali ad esempio dispositivi hardware integrati e telefonia.

Un'applicazione Android è formata da una certo numero di componenti. I componenti sono 4, e sono:

- Activity è un pezzo di interfaccia con il relativo codice applicativo.
- Service rappresenta un servizio eseguito in background.
- Content Provider contiene dati e fornisce l'interfaccia per accedervi.
- Broadcast Receiver riceve notifiche di sistema e di altre applicazioni.

Essi costituiscono i nuovi concetti introdotti nella piattaforma. Questi concetti non fanno parte nativamente del linguaggio Java ma sono inclusi tramite librerie. Un ulteriore concetto è quello dell'Intent, che serve come mezzo di comunicazione tra Activity, Service e Broadcast Receiver.

I vari componenti scelti per la propria applicazione devono essere inseriti nel file Manifest che viene poi incluso nell'apk e che contiene le informazioni che costituiscono l'interfaccia pubblica dell'App.

Qualunque applicazione può far partire un componente di un'altra applicazione o dell'applicazione stessa. Ciò accade attraverso gli Intent. Gli Intent sono una forma di messaggio che viene recapitato al sistema per segnalare che si vuole far partire un certo componente. Infatti, siccome le applicazioni vengono eseguite in processi separati e non possono uscire dalla loro macchina virtuale Dalvik, non è possibile far partire un componente di un'altra applicazione o comunque recapitargli un messaggio direttamente. Il sistema viene usato quindi come intermediario.

Esso si occupa di far partire il processo dell'applicazione corrispondente, se essa non è già in esecuzione, dopodiché istanzia un oggetto della classe relativa al componente e lo avvia.

Una diretta conseguenza di questa caratteristica è che un'applicazione non ha un unico entry point, come una funzione main(), ma ne ha tanti quanti sono i componenti che dichiara come pubblici, ovvero che hanno degli Intent Filters.

Se l'applicazione viene fatta però partire dall'utente tramite l'icona dell'App, l'entry point è rappresentato dalla Main Activity dell'applicazione, contrassegnata così tramite il file Manifest attraverso particolari Intent Filters.

Come detto, di default ogni applicazione è ospitata da un processo a sé stante con una propria istanza della Dalvik Machine. In questo processo viene anche eseguito il codice di tutti i componenti che la compongono. Ogni pezzo di codice è eseguito nello stesso thread, a meno che non sia avviato esplicitamente un altro thread utilizzando i meccanismi nativi di Java per la gestione dei thread, oppure usando classi per gestire in modo automatico le interazioni tra main thread e thread secondari, come ad esempio AsyncTask o IntentService.

Il sistema può decidere di uccidere un processo, distruggendo tutti i suoi componenti, rispettando un ordine di priorità:

• I primi ad essere eliminati sono i processi vuoti, che sono considerati tali se sono attivi senza che sia attivo alcun componente. Ciò significa che per eseguire codice in background senza correre il rischio che l'operazione venga interrotta, occorre che esso sia legato in qualche modo a un Broadcast Receiver o a un Service.

• Dopo si passa ad eliminare i processi che contengono le Activity in background.

• A seguire si uccide il processo dell'Activity visibile ma non completamente in foreground. Ciò accade di rado e in situazioni di gravi scarsità di memoria.

• L'ultimo processo che viene distrutto è quello dell'Activity in foreground, considerata la più importante. Essa viene distrutta solo nel caso in cui occupi più memoria di quella effettivamente disponibile nel dispositivo. A quel punto l'uccisione del processo è necessaria per mantenere il sistema ragionevolmente veloce poiché il sistema sta rallentando a causa del paging su disco.

Activity

L'Activity rappresenta una schermata mostrata all'utente. Ogni Activity deve essere sottoclasse della classe Activity, fornita dalla piattaforma. La schermata è contenuta in una finestra che è assegnata all'Activity e che essa si occupa di riempire utilizzando il metodo setContentView(View). Grazie a questo metodo è possibile sia creare un'interfaccia composta da una gerarchia di View, oggetti di classe (o sottoclassi di) View, racchiusi in un ViewGroup che normalmente è un layout, ovvero una direttiva sull'ordine e la disposizione delle View in esso contenute. L'interfaccia può essere specificata sia tramite un file XML che contiene la descrizione della schermata, sia creando ViewGroup e View da codice.

Ogni activity viene fatta partire dal sistema grazie a un Intent. Per gestire il ciclo di vita dell'Activity è necessario implementare alcuni metodi che vengono chiamati dalla piattaforma durante l'evolversi dei cambiamenti di stato, quali onCreate(), onPause(), ecc.

Un'activity può essere in uno dei seguenti stati:

- Resumed/Running è lo stato in cui l'Activity è in foreground e pronta a ricevere eventi da parte dell'utente.
- Paused in questo caso, l'Activity è ancora in foreground ma è parzialmente coperta da un'altra Activity che non copre tutto lo schermo, lasciandola in parte visibile.
- Stopped l'Activity è stata rimossa dal foreground ed è in background. Di fatto è ancora viva ma non è collegata al gestore delle finestre.

Quando un'Activity è Paused o Stopped può essere distrutta da parte del sistema per ragioni di risparmio di memoria. In ogni caso è possibile salvare la condizione dell'Activity al momento della terminazione utilizzando il metodo onSaveInstanceState(), che comunque è già implementato nella classe originale per il

salvataggio di base (ad esempio, quando viene ruotato lo schermo l'Activity viene distrutta e ricreata e in automatico sono salvati, ad esempio, i contenuti delle Text Field).

Quando un'applicazione viene fatta partire, il sistema le riserva uno stack detto back stack che permette di tenere traccia delle Activity e di navigare all'indietro tramite il tasto back. Il comportamento di default prevede che un Intent diretto a una Activity ne provochi l'instanziazione e la porti in foreground; a questo punto essa viene inserita nello stack della propria applicazione con una push. Se l'Activity ne fa partire un'altra, quest'ultima viene collocata in cima allo stack, facendo passare la precedente in background ma mantenendone lo stato. Alla pressione del tasto back, l'Activity in foreground viene distrutta e ne viene fatta la pop dallo stack. A questo punto l'Activity non è più recuperabile e viene portata in foreground l'Activity che si trova in cima allo stack.

Service

Un servizio è un componente che si occupa di eseguire operazioni lunghe in background. Di default è eseguito sul main thread, per cui a meno di non usare interazioni particolari, descritte a breve, occorre gestire un eventuale multithreading.

I servizi si dividono in Started Service (o Unbounded Service o semplicemente Service) e Bounded Service.

Un Service normale non è necessariamente o Bound o Unbound, ma può essere entrambi contemporaneamente.

Content Provider

Un Content Provider si occupa di gestire un insieme di dati condiviso. Questi dati possono trovarsi su un database, su file system, su internet o in un qualsiasi altro tipo di supporto di memoria.

I database che possono essere utilizzati sono di tipo SQLite.

I Content Provider sono necessari per condividere dati tra applicazioni, ma sono utili anche per dati che devono rimanere privati all'interno di un'App. E' possibile sia creare un proprio Content Provider per i propri dati sia utilizzarne uno già esistente, dopo aver ottenuto i dovuti permessi. Per creare un Content Provider occorre fare una sottoclasse della classe ContentProvider.

Occorre perciò implementare una serie di metodi che fanno parte dell'interfaccia:

- query()
- insert()
- update()
- delete()
- getType()
- onCreate()

Implementati questi metodi e definiti i vari identificativi per l'accesso ai dati, non è più necessario lavorare direttamente con il Content Provider. Infatti, i contenuti dei Content Provider possono essere ottenuti utilizzando un Content Resolver e specificando l'identificativo dei dati che si vogliono ottenere. I dati sono identificati da un URI che identifica ad esempio il database, la tabella e l'id del dato.

Broadcast Receiver

I componenti di tipo Broadcast Receiver sono sottoclassi di BroadcastReceiver e servono per ricevere e reagire a determinati eventi, chiamati appunto Broadcast, che possono essere originati dal sistema (ad esempio, per avvisare l'utente che la batteria è quasi scarica) o dalle applicazioni. I Broadcast Receiver sono abilitati a ricevere Intent inviati con appositi metodi e lo scope dell'Intent può essere limitato all'applicazione stessa, oppure tutto il sistema può riceverlo.

Un Broadcast Receiver si limita a implementare il metodo onReceive() che viene chiamato dal sistema al momento della ricezione di un broadcast. Terminato il metodo, il componente non viene più considerato attivo.

Intent

Gli Intent permettono ai componenti di un'applicazione di comunicare tra di loro. E' una forma di comunicazione indiretta poiché è mediata dalla piattaforma sottostante. Ogni componente (esclusi i Content Providers) può far partire un altro componente tra Activity, Service e Broadcast Receivers inviando l'apposito Intent.

L'Intent può essere esplicito o implicito. Nel caso in cui sia esplicito, il componente specifica direttamente il nome della classe di cui vuole che il componente sia istanza. Questo metodo funziona per Intent all'interno della stessa applicazione.

Nel caso degli Intent impliciti, possono essere specificate alcune informazioni quali l'azione che deve essere eseguita, la categoria dell'azione e i dati da utilizzare.

In questo caso, il sistema verifica quali applicazioni presentano nel proprio Manifest componenti in grado di rispondere a questo tipo di Intent e, nel caso ce ne sia più di uno, il componente viene fatto scegliere all'utente.

Per questo tipo di selezione, per ogni componente nel Manifest devono essere specificati degli Intent Filters, che servono a indicare a quali tipi di Intent i componenti sono in grado di rispondere. Un filtro particolare è quello utilizzato per specificare che una certa Activity è la Main Activity di un'applicazione; in quel caso occorre dichiarare che l'Activity risponde ad una azione di tipo MAIN della categoria LAUNCHER. Se non si vuole che i componenti siano accessibili fuori dallo scope dell'applicazione, basta non specificare alcun Intent Filter.

Nel caso degli Intent impliciti, potrebbe essere che l'applicazione che contiene il componente target non sia attiva e in quel caso viene fatta partire, perciò l'Intent non provoca solo la consegna di un messaggio ma anche la creazione di un nuovo processo e l'istanziazione del componente interessato.

Un Intent può contenere anche degli extra. Gli extra sono dati inclusi nell'Intent che vengono serializzati e inviati verso il componente target.

View

Le interfacce grafiche sono formate da una gerarchia di View e View Group. I View Group sono istanze della classe ViewGroup e sono oggetti atti a contenere degli oggetti View, per cui costituiscono layout e contenitori. Le View, dette anche widget, sono istanze di classe View e possono essere di vario tipo, dai bottoni ai selettori di date. E' ovviamente possibile fare una sottoclasse sia di una View che di un widget già presente per creare componenti personalizzati.

La classe View, oltre a racchiudere le caratteristiche di un componente che deve essere disegnato su schermo e in grado di interagire con l'utente, include anche una serie di interfacce per la gestione degli eventi. Per ricevere un evento in un'Activity, occorre associarle un ascoltatore per l'evento specifico; l'ascoltatore deve estendere l'apposita interfaccia e implementare il metodo associato all'evento.

Esistono particolari tipi di ViewGroup, detti Adapter View, che si occupano di effettuare il collegamento tra un insieme di View e dei dati, che possono essere ad esempio array (ArrayAdapter) o provenire da database tramite cursori (CursorAdapter).

Gli Adapter recuperano i dati dalla sorgente indicata e si occupano di costruire un numero congruo di View di conseguenza.

1.3 Caratteristiche di una applicazione mobile

- Funzionare sulla maggior parte degli smartphone

  Più devices sono supportati dalla applicazione, maggior sarà la possibilità di successo della stessa.

- Facile da usare

  Indipendentemente dalla complessità della applicazione, l'utente deve poter accedere in modo semplice e veloce alle funzionalità dell'applicazione, qualsiasi sia il tipo di utente.

- Acquisire i dati dall'esterno e inviarli
  - Consentire di avere una applicazione sempre aggiornata è un punto di forza rispetto al fornire una applicazione statica.
  - Consentire la condivisione di file multimediali, quali foto, audio o video è un punto di forza del prodotto finale. Tramite questa azione è possibile far conoscere il prodotto oltre il comune passaparola.

- Essere il più social possibile

  Consentire all'utente di pubblicare stati o multimedialità attraverso i principali social network è una opportunità di accrescimento degli utenti.

- Supportare il maggior numero di lingue

  Dotando l'App di multilingua è possibile estendere il bacino degli utenti finali.

- Cattura immagini e GPS-coordinate

  Consentono di avere una applicazione dinamica, ove necessario, in modo da fornire informazioni dei punti di interesse vicini.

- Notificarne il funzionamento all'azienda

  Tramite una mail o un form di contatti dovrebbe essere possibile dare l'opportunità all'utente di segnalare delle anomalie oppure dei consigli per migliorarla. Come nel caso di questa applicazione, sicuramente l'agente saprà meglio le potenzialità che l'applicazione dove avere.

2 Order Sender

2.1 Descrizione iniziale

Order Sender è una applicazione per tablet rivolta agli agenti di commercio che vogliano informatizzare il processo di compilazione, memorizzazione ed invio delle commesse dell'ordine. Con questa applicazione l'agente può gestire i propri dati, quelli dei propri clienti e tutto quello che è intorno, come prodotti, fornitori e sconti. Però la funzione principale dell'applicazione è quella di creare e gestire delle commesse, che possono essere create compilando una serie di campi in una interfaccia intuitiva e completa, successivamente è possibile inviarla, dopo una conversione PDF, ai destinatari desiderati. Attraverso un servizio web è possibile importare ogni genere di dato all'interno del tablet. Analogamente è possibile esportare tutti i dati su un altro tablet oppure sul sito web dedicato.

Questo genere di applicazione per essere completa deve essere accompagnata da una serie di servizi web che le permettono di interagire con un database remoto, che contiene tutti i dati utili a eseguire un backup del tablet, questo servizio prende il nome di Order Sender Business.

Ciò avviene tramite un sistema di sottoscrizioni, identificate tramite nome utente e password dell'utente dell'azienda.

Questi servizi sono offerti dalla stessa azienda che ha creato e prodotto il software.

Per consentire all'agente di avere una gestione completa del proprio lavoro è stata introdotta la possibilità di avere una gestione web che replica in modo simmetrico i dati salvati, attraverso cui è possibile effettuare le stesse operazione della applicazione e avere nuove funzionalità non ancora disponibili sul tablet.

Tutto questo è legato alla versione che l'agente o l'azienda decide di comprare, ogni abbonamento offre funzionalità avanzate rispetto al modulo di fasce inferiore, in modo da consegnare un prodotto sempre più completo.

L'applicazione è stata sviluppata sotto piattaforma Android e pubblicata sul Google Play, ed è stata fatta una versione analoga anche per iOS presente sull'App Store di Apple.

2.2 Analisi del progetto e dei requisiti

Un applicazione di questo tipo deve avere della funzioni di base, quali:

- Compilazione dell'ordine
- Invio dell'ordine
- Gestione dei dati
- Importazione
- Esportazione
- Sincronizzazione bidirezionale
- Manuale

Descriviamo ora ogni singolo requisito:

2.2.1 Compilazione dell'ordine

La prima e più importante funzionalità richiesta da questo tipo di applicazione è la possibilità di compilare una commissione d'ordine. Il programma deve avere una interfaccia il più semplice possibile per permettere anche ad utenti meno esperti di avere il pieno controllo dei campi da compilare. Questo aspetto è facilitato dall'auto-completamento dei campi del fornitore e del cliente, ovvero, all'immissione di un carattere contenuto all'interno del nome del fornitore/cliente, verrà fornito automaticamente un elenco dove è possibile scegliere il fornitore/cliente voluto. Alla selezione del campo verranno automaticamente completati i campi riguardanti la suddetta. Questo però implica l'anticipata creazione del genere voluto, quindi è stata introdotta anche la possibilità di creare Runtime i dati inseriti nei campi e salvarli nel database con la semplice pressione di un tasto. Inseriti i dati base quali: fornitore,

cliente, dati della spedizione, dati dell'ordine è possibile passare alla parte relativa all'inserimento dei prodotti e dei propri dati. Infine per completare la completezza della commissione vengono richiesti firma cliente e agente sotto forma di nome e cognome, una casella grafica dove è possibile creare la propria firma da inserire nel documento finale e un campo per delle eventuali note.

2.2.2 Invio dell'ordine

All'avvenuta compilazione di tutti i campi con almeno un fornitore, un cliente e un prodotto è possibile terminare l'ordine con l'invio in un PDF riassuntivo all'indirizzo mail desiderato, di default quello dell'agente. Tutto ciò viene fatto attraverso una richiesta HTTP con metodo POST contenente tutti i dati all'indirizzo specifico che si occupa di elaborati i dati ed estrapolare un file PDF, successivamente inviato per mail. Nel caso non fosse stato possibile recuperare l'indirizzo mail dell'agente sarà richiesto l'inserimento manuale per l'invio.

Quanto l'ordine viene inviato al server, per la creazione del file, in automatico viene salvato anche nel database remoto, quindi si avrà una copia degli ordini sia nel tablet sia online.

2.2.3 Gestione dei dati

Un'altra sezione molto importante è quella per la gestione dei dati. E' molto importante che questa parte sia il più semplice possibile per permettere all'agente di non commettere errori nell'inserimento dei dati. Infatti, sono state create quattro interfacce diverse per l'inserimento di fornitori, clienti, prodotti e sconti. In particolare, i prodotti sono associati a un fornitore e gli sconti a un fornitore, un cliente ed eventualmente a un prodotto (nel caso non sia specificato un prodotto, lo sconto è inteso per tutta la gamma di prodotti del fornitore per quel cliente).

Sono presenti anche delle schermate per la modifica degli ordini creati ma non ancora inviati, mentre per gli ordini creati ed inviti sarà possibile la duplicazione che avrà un identificativo interno diverso rispetto all'originale. Questi ordini possono anche essere filtrati per fornitore o cliente.

Infine l'agente avrà a disposizione un'interfaccia, dove poter inserire i propri dati.

2.2.4 Importazione

La maggior parte delle volte l'agente inizia ad utilizzare l'applicazione avendo già un bacino di fornitori e clienti, presenti o meno in una banca dati. Il sito web di Order Sender permette di importare un file, in diverse codifiche, nella sezione dedicata. Il classico file importato è un foglio elettronico contenente tutti i record. All'avvenuto caricamento del file sarà restituito un codice identificativo di 6 lettere del file, che dovrà essere inserito insieme a username, password, tipologia del dato e l'eventuale fornitore (come nel caso di una importazione di prodotti), all'interno dell'interfaccia del tablet. L'applicazione in automatico recupera il file caricato dal sito e inizia ad elaborarlo. Alla fine l'agente si ritroverà tutti i dati importati come se li avesse inseriti tutti a mano, uno ad uno. Successivamente ci sarà la possibilità anche di modificarli nel caso sia necessario apportare delle modifiche.

La ricezione dei dati da parte del tablet avviene tramite una richiesta HTTP GET con i dati dell'utente e il codice. Il file restituito sarà composta da righe diverse contenenti ognuno un fornitore/cliente/prodotto con il doppio pipe che ne divide i campi corrispondenti. L'applicazione applica un metodo di codifica conforme a quello per la creazione del file e sarà in grado di riconoscere l'esistenza o meno di ogni singolo campo. Fatto questo, vengono effettuate le modifiche ai record della tabella corrispondente nel database. Alla fine del processo il server in automatico rimuove il file per motivi di privacy, comunicatogli con una richiesta HTTP dall'applicazione.

2.2.5 Esportazione

E' composto da una procedura analoga ma con l'unica differenza che il file di esportazione è creato dal tablet e non dal server web. Viene data la possibilità di esportare qualsiasi tipo di dato dai fornitori ai prodotti ai clienti, sempre associato a un username e password. Una volta che l'applicazione ha recuperato i dati dal database locale, viene creato il file che attraverso una richiesta HTTP POST con il file allegato viene inviato al server. In seguito all'invio sarà possibile, nell'area riservata dell'utente, il download o la cancellazione del file.

2.2.6 Sincronizzazione bidirezionale

Una sezione molto importante per la possibilità di creare e modificare qualsiasi tipo di dati è la sincronizzazione bidirezionale. Tutto ciò è possibile attraverso l'uso di sottoscrizioni che sono associate all'utente e che contengono la data dell'ultima sincronizzazione. Le sottoscrizioni sono caratterizzate da un nome identificativo, l'username e la password dell'agente, un codice identificato dell'azienda. Al primo inserimento della sottoscrizione viene chiesto se si desidera sincronizzare i dati con quelli presenti sul tablet. Nel caso sia una nuova sincronizzazione tutti i dati contenuti all'interno del tablet verranno inviati al server web che poi si occuperà di codificarli e inserirli nel database remoto. Invece se si è rispristinata una vecchia sincronizzazione verranno solo mandati i dati che non risultato essere presenti in tutti e due i database. Se si verifica la duplicazione di una dato ma con un campo diverso è possibile scegliere quale tenere tra la versione del tablet e quella online. Viene data anche la possibilità di inviare o no tutti gli ordini contenuti nel tablet a Order Sender Business, che è la piattaforma online. Una volta portata a termine la sincronizzazione si avrà una copia speculare dei dati del tablet e quelli online. Questo è molto utile nel caso di malfunzionamenti del tablet oppure una perdita dei dati. Introducendo la possibilità di modifica e creazione online è possibile lavorare anche da un pc senza la necessità di avere a fianco il tablet.

2.2.7 Manuale

Una parte del programma infine è dedicata alla visualizzazione del manuale d'uso in lingua italiana o inglese dell'applicazione in formato PDF.

2.2.8 Categorie di dato memorizzabili

- Agente

  L'agente è la persona che utilizza il tablet o il sito web, è nominato dall'azienda mandataria e si occupa di recarsi presso le aziende clienti ed effettuare ordini di merce del fornitore.

- Fornitore

  Fornisce tutti i prodotti che verranno inseriti in ogni singolo ordine.

- Cliente

  Può essere un privato o un'azienda ed è il destinatario dell'ordine.

- Prodotto

  Contiene una serie di informazioni relative a ciò che è venduto dall'azienda.

- Sconto

  Nel caso sia stato concesso uno sconto a un cliente su un certo fornitore, all'occorrenza mirato a una serie di prodotti, è possibile gestirlo.

- Ordine

  E' il contenitore di tutti i precedenti elementi in un documento solo con valore legale. Sostanzialmente attesta l'acquisto da parte del cliente di una lista di prodotti, con le relative caratteristiche (quantità e prezzo finale), da un fornitore.

- Sottoscrizione

  Permette la sincronizzazione dei dati con il server web per la visualizzazione sulla sezione dedicata del sito.

2.3 Esempi di utilizzo

L'applicazione deve essere in grado di:

- Memorizzare e garantire ottime prestazioni anche con quantità grandi di dati.
- Interagire con i servizi web.
- Interpretare una codifica uguale a quella usata dal server.
- Fornire all'utente finale un'interfaccia semplice e intuitiva.

Presentiamo ora tre tipici scenari di utilizzo dell'App.

1. Un'agente si presenta davanti a un cliente ed effettua un ordine. Tutti i dati sono inseriti tramite le apposite form in modo semplice e veloce. Creato l'ordine è possibile inviarlo sia al cliente sia, come resoconto, all'agente. Nel caso ci siano dei problemi sul tablet o di invio, l'ordine risulterà salvato ma non inviato.
2. Un'agente, alla fine della giornata di lavoro, decide di immettere i dati riguardanti gli ordini tramite l'interfaccia del sito web. Terminata l'operazione di creazione, è possibile inviare i resoconti direttamente dal sito, oppure, dopo una sincronizzazione, inviare dal tablet l'ordine.
3. Un'agente ha inserito durante la giornata ordini nel tablet e, in un secondo momento, altri dal sito web. Terminato l'ultimo inserimento, avviando una sincronizzazione dal tablet è possibile unire gli ordini e creare una lista uguale sia in locale che online.

2.4 Versioni di Order Sender

Order Sender è disponile sul Play Store di Google e sull'App Store di Apple in due versioni:

1. **Order Sender Lite**

   Versione gratuita contenente la creazione dell'ordine ma escludendo tutta la gestione dei dati, l'importazione, l'esportazione e la sincronizzazione. Nel caso si voglia sbloccare tutte le caratteristiche è possibile acquistare la versione a pagamento.

2. **Order Sender Pro**

   Versione a pagamento dell'applicazione con tutte le features descritte in precedenza.

3 Front-End per l'interazione con il cliente

La parte dell'interfaccia utente è stata resa il più possibile semplice per cercare di far capire subito all'agente dove sono le principali funzioni. Appena si apre la applicazione si ha una chiara idea delle potenzialità e caratteristiche dell'applicazione. Inizialmente verrà visualizzata la caratteristica principale, ovvero la creazione e modifica degli ordini. Si trovano 5 semplici bottoni con cui accedere facilmente alla sezione voluta. Tramite il bottone "Crea nuovo ordine", viene data la possibilità di compilare una form completa di tutti i campi che sono tipici di una copia commissione. "Ultimi ordini" visualizza uno storico degli ultimi 50 ordini creati, premendo "Archivio Ordini" si ha l'archivio completo di tutti gli ordini ordinati per data di creazione. "Cerca per fornitore" e "Cerca per cliente" forniscono una ricerca semplice e veloce applicando un filtro in base alla selezione effettuata successivamente.

3.1 Creazione di un ordine

La prima impressione che questa interfaccia trasmette è sicuramente la facilità di individuazione della sezione voluta.

In alto viene creato ad hoc un codice identificativo dell'ordine sia all'interno del tablet sia on-line, questo permetterà una migliore gestione con una massa di dati notevole. Affianco troviamo un selettore di data dove viene inserita automaticamente la data odierna, ma è possibile modificarla.

Le sezioni sono divise come segue:

3.1.1 Fornitore

Per facilitare l'agente è stato inserito un auto completatore, che appena viene inserito un carattere nel campo contenente il nome del fornitore, viene visualizzata una tendina contenente tutti i risultati contenente quel carattere.

Se si continua con l'inserimento dei caratteri, si arriverà ad un certo punto che avremo solamente un valore selezionabile; nel caso sia stata inserita una stringa che non è contenuta in nesssuno dei fornitori, verrà nascosta la tendina e si passa all'inserimento manuale.

Se è stato selezionato un campo, in automatico vengono compilati tutti i campi ove sono presenti.

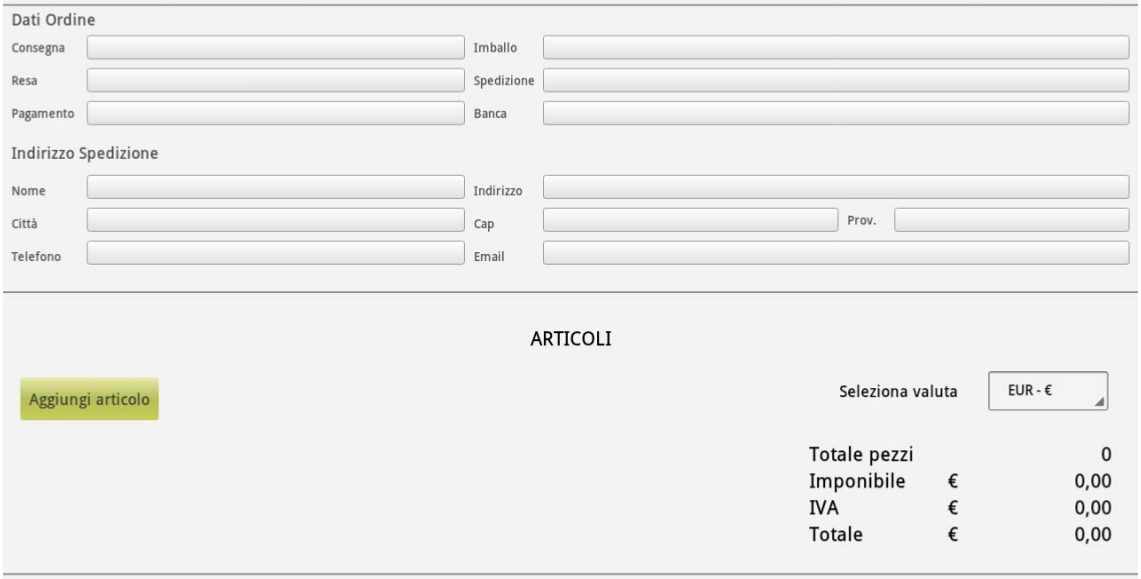
Qualora mancassero dei campi è possibile inserirli per poi premere il tasto "Salva Fornitore" e verranno automaticamente aggiornati tutti i campi nel database.

Mentre se si è scelto di inserire manualmente il fornitore allora, una volta inseriti tutti i dati, sarà possibile salvarlo o meno.

E' presente anche un tasto "Svuota Campi" che elimina tutti i contenuti presenti in questa sezione.

3.1.2 Cliente

Questa sezione è analoga a quella relativa al fornitore, con l'unica differenza che saranno visualizzati nella tendina i clienti.



3.1.3 Dati ordine

Inserito recentemente, su richiesta di utenti finali dell'applicazione, da la possibilità di aggiungere delle informazioni quali consegna, resa, pagamento, imballo, spedizione e la banca.

3.1.4 Indirizzo spedizione

Contiene una serie di campi che indicano l'utente finale che riceverà la merce contenuta nell'ordine.

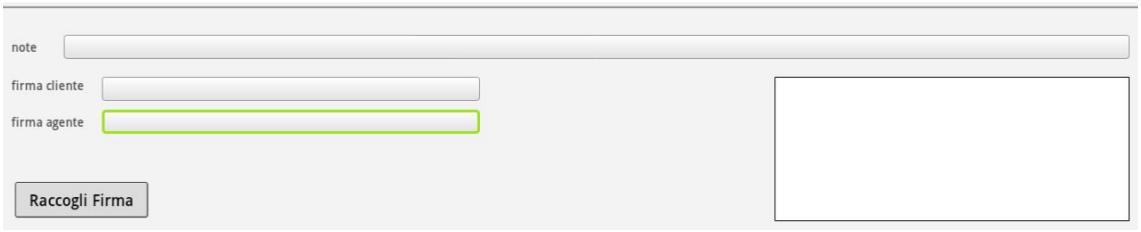
3.1.5 Articoli

Questa è la parte più dinamica di questa schermata.

Contiene principalmente un bottone per l'inserimento di articoli, un resoconto dei prezzi ivati e non, della lista dei prodotti e la possibilità di scegliere la valuta.

Alla pressione del tasto "Aggiungi Articolo" verrà aggiunta una riga alla lista degli articoli, dove sarà possibile inserire, sempre con un auto completatore, i dati relativi ad ogni singolo prodotto.

L'auto completatore all'inserimento del carattere restituirà nella tendina, il risultato del filtro applicato alla lista dei prodotti per il fornitore indicato in precedenza.



3.1.6 Note finali e firme

In fondo è sempre presente una sezione dove è possibile inserire un campo contenente delle note da allegare all'ordine, una firma testuale del cliente, una firma testuale dell'agente e un riquadro contenente una vera firma, che viene inserita nella schermata che viene visualizzata alla pressione del tasto "Raccogli Firma".

3.1.7 Salvataggio ordine e invio della mail

In fondo è presente il tasto Invia e nel caso sia stato aperto un ordine già creato ed inviato è possibile duplicarlo per eventuali modifiche e un novo salvataggio.

Nel caso venga premuto il tasto "Invia" viene aperta una schermata che, nel caso non sia stato possibile recuperare le mail dell'agente e del cliente, sarà necessario inserirlo a mano altrimenti saranno visualizzate e basterà solamente verificare l'esattezza dei dati.

Nel caso siano giusti si può procedere con l'invio definitivo dell'ordine che contiene tutti i dati inseriti. Il server, una volta ricevuti tutti i dati, procede con la decodifica dell'ordine, l'inserimento nel database remoto e l'invio del resoconto tramite mail all'indirizzo inserito.

3.2 Gestione dei dati personali

Compilando la form con i dati personali dell'agente, sarà possibile auto completare gli ordini creati con i dati dell'agente.

In questa sezione è possibile inserire nome, cognome, telefono, cellulare, fax, email, indirizzo e città.

Inoltre la possibilità di forzare l'intera applicazione ad usare la propria valuta e l'IVA inserita. Questi verranno inseriti automaticamente nell'ordine, che poi potranno essere modificati manualmente.

L'IVA di default viene inserita in automatico nell'ordine per i prodotti che non hanno un proprio valore specifico.

3.3 Importazione

E' possibile importare direttamente da un file Excel il proprio elenco di fornitori, clienti e prodotti.

Per farlo è necessario collegarsi al sito http://areariservata.ordersender.com effettuare il login con i propri dati ed entrare nella sezione Importazione.

Seguendo le istruzione è possibile caricare il file contenente i dati, terminato l'upload sarà fornito un codice, sarà l'identificativo univoco del file caricato sul server. Questo codice dovrà essere inserito nel tablet, nell'apposita area, specificando anche la tipologia del dato, cioè se è un fornitore, un prodotto o un cliente. Nel caso si stia importando un file contenente dei prodotti, sarà possibile, tramite una casella di scelta multipla, selezionare il fornitore associato.

Tutti i dati devono essere accompagnati da username e password dell'agente, per consentire l'identificazione dell'azienda.

I dati inseriti verranno aggiunti all'elenco già esistente e sincronizzati automaticamente con il server online.

Questa operazione viene eseguita creando un AsyncTask. Per evitare di gestire i thread manualmente Android mette a disposizione la classe AsyncTask, questa classe fa largo uso dei generics di Java, vediamo di capire come usarla. I generics sono stati introdotti in Java 1.5 per aumentare la "sicurezza in compilazione".

La classe AsyncTask definisce tre generics:

```
AsyncTask<Params, Progress, Result>
```

I metodi principali di AsyncTask:

onPreExecute: eseguito sul thread principale, contiene il codice di inizializzazione dell'interfaccia grafica (per esempio la disabilitazione di un button).

doInBackground: eseguito in un thread in background si occupa di eseguire il task vero e proprio. Accetta un parametro di tipo Params (il primo generic definito) e ritorna un oggetto di tipo Result.

onPostExecute: eseguito nel thread principale e si occupa di aggiornare l'interfaccia dopo l'esecuzione per mostrare i dati scaricati o calcolati nel task che vengono passati come parametro.

Nel nostro caso nell' onPreExecute saranno caricati tutti i dati che mi serviranno per l'inoltro della richiesta al server.

Nel doInBackground saranno eseguite le seguenti operazioni:

- Creazione di un file temporaneo sul device che mi permetterà di ospitare il file scaricato per poter eseguire le operazioni in locale senza dover comunicare con il server per ogni dato inserito.
- Tramite una classe HttpHandler implementata ad hoc e il suo metodo scaricaFile verrà fatta una richiesta al server ad un indirizzo specifico, che sarà composta dal sito dell'applicazione più una stringa di riconoscimento del servizio richiesto ed una serie di parametri, mirati a selezionare i dati da importare, esempio:
  http://business.ordersender.com/funzioni/import/tip/clienti/codice/ATR4O/username/***/pwd/***/versione/new
  Questa chiamata al server mi restituirà un file che attraverso un buffer in entrata, mi leggerò e salverò nel file temporaneo creato in precedenza.
- Terminata la scritture del file posso passare all'importazione vera e propria in base al dato inserito, ovvero richiamerò un metodo diverso per ogni categoria di dato in modo da poterlo codificare per il salvataggio. Tutti i campi saranno separati da un doppio pipe che mi servirà per capire il termine del valore.k
  Le stringhe di ogni singolo campo saranno così composte:
  - Cliente
    nome||responsabile||partitaIva||codiceFiscale||telefono||fax||cellulare||citta||indirizzo||note||email||cap||provincia||iban
  - Fornitore
    nome||responsabile||telefono||fax||cellulare||email||indirizzo||citta
  - Prodotto
    codice||descrizione||qmin||prezzo||fornitore

Potendo sapere esattamente in quale posizione saranno i campi è possibile fissare una regola fissa di passaggio dei dati, in modo che dopo aver associato ogni singolo campo alla sua entità sarà possibile salvarlo sul database.

Nell' onPreExecute saranno eseguite le operazioni grafiche, come la visualizzazione del messaggio di importazione eseguita correttamente o fallita come nel caso di un errore di connessione nel download del file oppure un qualsiasi altro errore. Sarà inoltre cancellato il file temporaneo che mi era servito per importazione.

Questo è il processo di importazione eseguito dal programma per ogni tipo di dato, che è molto simile all'esportazione che andiamo ora a descrivere.

3.4 Esportazione

In questa sezione è possibile esportare in un file CSV, utilizzabile con Excel, il proprio elenco di fornitori, clienti e prodotti.

Per farlo bisogna specificare la categoria di elementi che si desidera esportare, nel caso di prodotti è necessario specificare anche il fornitore.

Inoltre, come per l'importazione specificare l'username e la password.

Terminata l'operazione sarà possibile accedere alla propria area riservata sul sito e navigando nella sezione desiderata, ci si imbatterà in un file nominato con la tipologia dell'elemento esportato e la data di esportazione, che sarà possibile scaricare in qualsiasi momento.

Ma vediamo come viene creato il file di esportazione.

Come per l'importazione tutte le operazioni vengono eseguite utilizzato un AsyncTask che mi permette di essere indipendente dal processo principale.

Gli step eseguiti sono:

- Caricamento dei dati di accesso nell' onPreExecute.
- Creazione ed invio del file temporaneo creato secondo lo schema usato anche per l'importazione, tutto contenuto nel doInBackground.

- Cancellazione del file e visualizzazione dell'esito.

3.5 Manuale

Permette la visualizzazione di un file PDF contenente il manuale dell'applicazione. In base alla lingua di default del tablet verrà selezionato un file diverso; il file è presente in due lingue: italiano e inglese.

Su piattaforma Android è stata fatta la scelta di poter far decidere all'utente con quale applicazione visualizzare il file, in base alle proprie preferenze.

3.6 Business

Questa è la sezione dove chi possiede un account Business ha la possibilità di inserire i propri dati per sincronizzarli e creare una sottoscrizione.

Viene visualizzata una lista di account inseriti con la relativa data dell'ultima sincronizzazione.

Tramite il bottone aggiungi è possibile accedere alla form dove verranno richiesti i dati dell'agente per la sincronizzazione.

Alla pressione del tasto salva viene fatta una richiesta al server di verifica dell'account e viene restituito il tipo di account inserito che è il tipo di abbonamento pagato dall'azienda.

Una volta verificati i dati verranno visualizzate una serie di finestre che chiederanno:

- Quali dati mantenere tra tablet e server nel caso di duplicati con ma con campi di descrizione diversi.
- Se è la prima sincronizzazione oppure si sta ripristinando una vecchia.

- Se si desidera invia gli eventuali ordini fatti dal tablet ma non ancora sincronizzati.

  Inserite queste risposte si passerà all'invio dei dati del tablet.

Verrà creato ancora una volta un file temporaneo che permette l'inserimento di tutti gli ordini, i dati presenti sul tablet e la lista delle preferenze scelte in precedenza. Questo verrà inviato ad un link dedicato:
http://business.ordersender.com/sottoscrizione/sync//codice/codAzienda/username/***/password/***/lastSync//lastSyncNews//udid/********

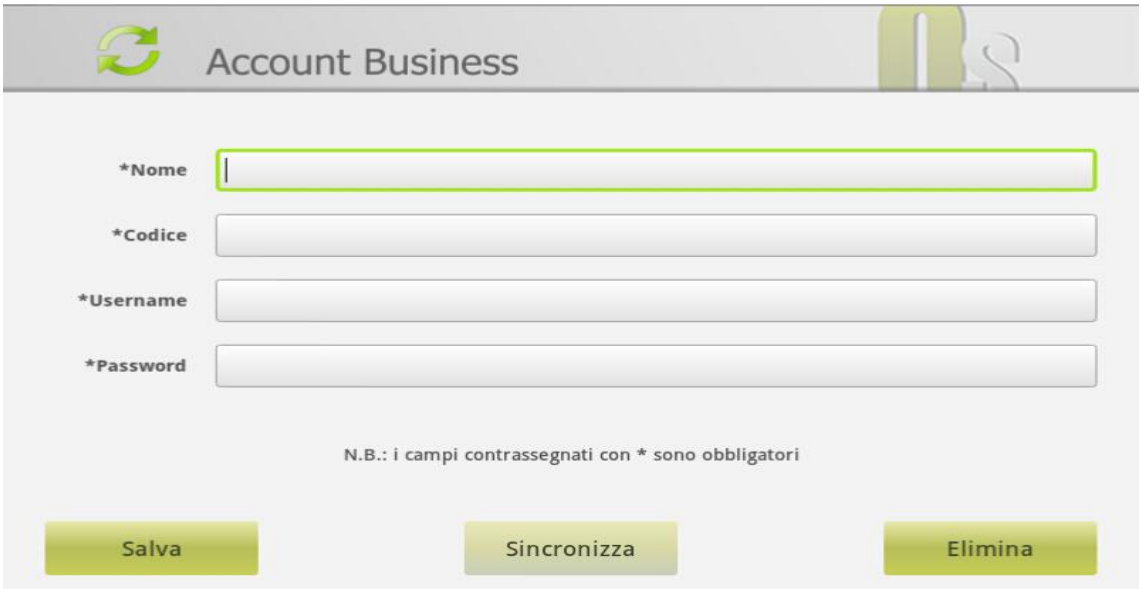
Il server una volta ricevuto il file provvederà all'importazione dei dati nel database remoto. Concluso questo step il server risponderà all'invio del file del tablet con un altro file contenente le eventuali modifiche che sono presenti sul server ma non ancora sul tablet.

Nel caso il file ricevuto dal dispositivo non contenga righe allora la sincronizzazione è terminata altrimenti verrà fatta tutta la procedura per ogni singolo record per ogni tipo di dato per la sottoscrizione inserita.

Come ulteriore conferma di inserimento e di sincronizzazione dei dati, terminata l'importazione viene inviata una striga contenente:

tipoDatoInserito||idLocale||idRemoto

per ogni singolo record modificato. Una volta che il server riceve questa lista viene confrontata con gli elementi inviati precedentemente e invia una risposta contenente "ok" per confermare l'esattezza dei dati importati, al contrario nel caso di problemi oppure anomalie nelle stringhe di conferma il server risponderà "ko". Questa stringa risultante servirà per la scelta della finestra da visualizzare alla fine in caso di operazioni eseguite correttamente oppure no.



3.7 Messaggi

Questo è un metodo di comunicazione tra l'azienda e l'agente per comunicare in modo immediato di eventuali errori sul server, oppure l'uscita di una nuova versione con miglioramenti o correzioni di bug.

Questi messaggi possono essere ricevuti anche senza inserimento di una sottoscrizione e quindi il collegamento a un abbonamento.

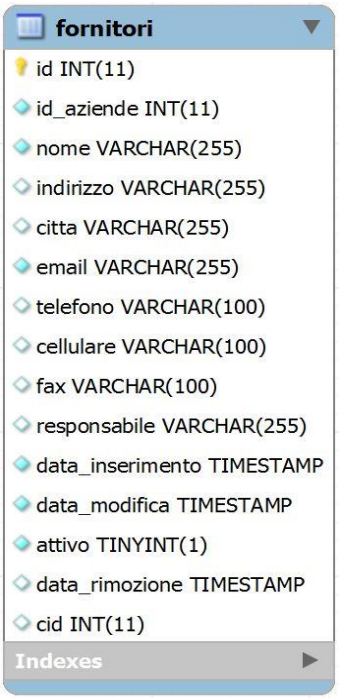
4 Gestione dei dati sul tablet

La gestione dei dati su tablet è stata fatta attraverso un database locale, contenuto nella cartella del programma e non accessibile all'utente, a meno di modifiche particolari al tablet.

Si è cercato di organizzare il più semplice ed efficiente possibile la grande quantità di dati che dovrà contenere. Essendo un programma che può supportare tranquillamente più di 60000 record per tabella, c'era la necessità di creare uno schema solito e standard nel tempo.

Ogni categoria di dato è contenuto nella sua tabella e con relazioni sia gestite dal db che dal programma è possibile recuperare qualsiasi tipo di dato con una query SQL.

4.1 Gestione dei fornitori



Questa struttura contiene tutti i campi che mi consentono di salvare un fornitore e tutte le proprie caratteristiche. Il campo id è la chiave primaria di ogni record e mi consente di riconoscere univocamente il fornitore all'interno del tablet. I campi nome, email sono campi non nulli, quindi obbligatori che saranno inseriti dall'utente.

Data inserimento, data modifica e attivo sono campi assegnati automaticamente dall'applicazione che mi servono al momento della sincronizzazione.

4.2 Gestione dei clienti

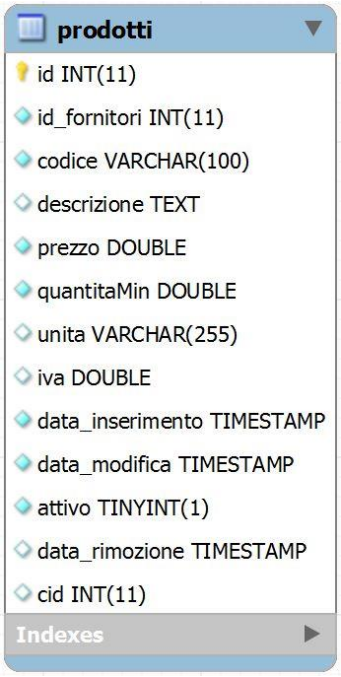
Come per i fornitori contiene una serie di campi che mi permettono di personalizzare ogni singolo cliente e indentificarlo univocamente attraverso il campo id.

Anche qui abbiamo dei campi obbligatori da compilare come nome e email.

Come per la tabella precedente abbiamo un campo denominato cid, che è il codice identificativo, in questo caso del cliente, all'interno del database remoto.

Notiamo ancora la presenza di campi che mi mantengono delle date e un flag attivo, che mi servono in fase di sincronizzazione ma anche per una eventuale memorizzazione anche dopo l'eliminazione del dato.

4.3 Gestione dei prodotti



Questa gestione contiene più campi obbligatori ovvero codice, l'id fornitore del prodotto, il prezzo e la quantità minima.

Questi dati verranno utilizzati in fase di creazione dell'ordine e gli verranno aggiunti.

4.4 Gestione degli sconti

Questa tabella contiene tutti gli sconti per un cliente riguardo un fornitore legato a uno a tutti i prodotti.

Infatti abbiamo che il campo id_prodotti non è obbligatorio, quindi è possibile creare uno sconto che valga che tutti gli articoli proposti da un fornitore verso un cliente.

Invece è obbligatorio l'inserimento di almeno uno sconto.

4.5 Gestione Agenti

Questa non è altro che la tabella legata alla form dei dati personali legati all'agente che contiene alcuni dati dell'agente, tra cui quelli di accesso alla sottoscrizione inserita.

Codice dispositivo è un campo che mi serve per inserire un identificato del tablet, perché per alcuni abbonamenti è possibile associare solo un dispositivo ad ogni utente.

Tipo dispositivo mi contiene l'indicazione del sistema operativo che monto sul tablet, mi sarà utile quando andrò a sincronizzare i dati.

Versione è il campo contenente la versione del programma attualmente installata.

4.6 Gestione delle news - messaggi

Tutti i campi sono obbligatori essendo campi che non possono esser nulli.

Contengono la news mandata dall'azienda ad ogni tablet in due lingue sia per il titolo che per il testo.

Anche qui abbiamo i dati che mi indicato le date e se la news è attiva o no.

4.7 Gestione degli ordini

Questa tabella è il fulcro del programma ed è anche quella con maggiori campi perché devi mantenere quasi tutto dell'ordine.

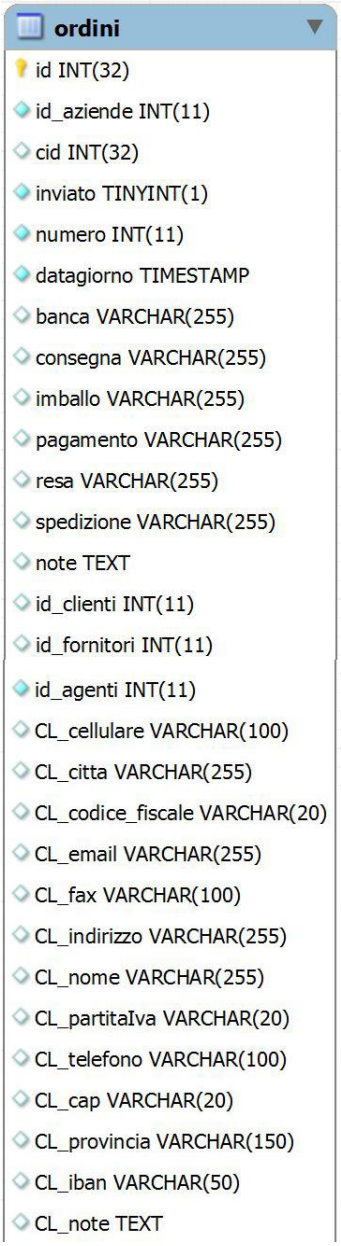
Questi sono solo alcuni dei campi presenti.

Come si può vedere anche l'ordine ha un id univoco locale ed ha anche un id azienda che è lo stesso della sottoscrizione inserita, se presente.

Contiene un flag denominato inviato, il numero dell'ordine e la data di fatturazione dell'ordine.

I campi relativi al cliente e al fornitore non sono obbligatori, ma sono stati inseriti dei vincoli nel codice del programma per inserire almeno il nome del cliente. Altri dati non obbligatori sono quelli relativi all'indirizzo di spedizione, i dati dell'ordine e delle eventuali note.

Negli altri campi abbiamo anche le due firme dell'agente e del cliente, un flag attivo, informazioni sulla data di inserimento e rimozione e, se presente, l'id del database remoto.



5 Gestione dei dati da web

Order Sender oltre a essere una applicazione è anche un servizio web del tutto autonomo.

Se l'applicazione è stata comprata dallo store e non è stato comprato nessuno dei pacchetti business, allora accedendo all'area riservata all'indirizzo web:

http://www.ordersender.com/areariservata

sarà possibile solamente effettuare una importazione o una esportazione di dati, essendo un semplice acquisto.

Mentre se l'azienda ha esigenze diverse, maggiori e più evolute, l'azienda produttrice dell'applicazione offre una versione business (lato server) di Order Sender, ovvero residente su macchine remote e non sui dispositivi degli agenti. E' raggiungibile come qualsiasi sito web e si può accedere semplicemente inserendo username e password.

Order Sender Business offre la possibilità di gestire in modo centralizzato, attraverso un'interfaccia intuitiva e di facile utilizzo, la lista clienti, fornitori, prodotti e sconti, sincronizzando tutti questi dati in modo automatico con i dispositivi degli agenti sui quali è installata l'applicazione.

Nel caso un'azienda voglia personalizzare il software web è possibile acquistare un pacchetto custom che permetterà di replicare il sito, sul proprio dominio ed effettuare modifiche come cambiare le immagini di testata del sito che della applicazione.

Le versioni disponibili sono: Order Sender Small, Medium e Large.

5.1 Funzionalità replicate

Come detto il sito web offre molte delle funzionalità dell'applicazione.

Il menu dell'interfaccia web si presenta cosi:

Tutto è stata architettato per essere come l'applicazione, ovvero il più semplice possibile. E' stato organizzato in bottoni contenenti una immagine che è già di per se descrittiva della funziona contenuta. Inoltre è presente un tasto abbonamento dove è possibile rinnovarlo oppure cambiare il tipo di pacchetto.

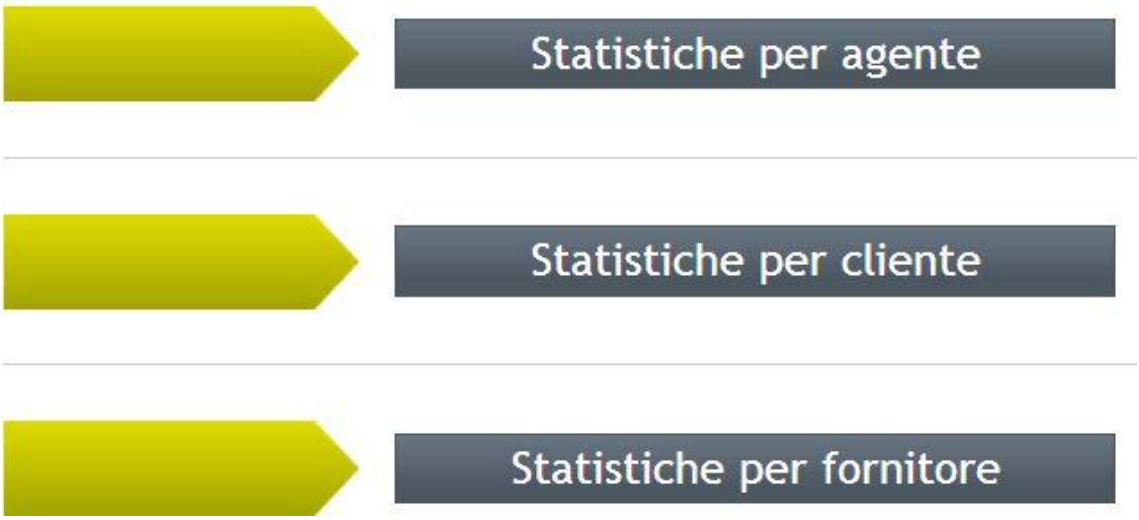
Sono presenti anche due riquadri dove sono riassunti i dati di accesso e dei contatori dei dati presenti.

Il codice azienda nel riquadro di sinistra sarà da inserire nel tablet al momento della creazione della sottoscrizione e sarà identificativo e ogni dato in modo da poter associare sempre ogni record all'agente o azienda che l'ha creata.

5.2 Statistiche

Due funzioni inserite recentemente sono la possibilità di creare ordini e la sezione statistiche.

Alla pressione del tasto viene caricata una pagina dove è possibile scegliere quale statistica si desidera visualizzare.



Scelta la statistica voluta sarà visualizzato un grafico molto intuitivo, per esempio nel caso dell'agente corrente:

Questa funzionalità sarà inserita a breve anche nella applicazione mobile, in modo da replicare in modo speculare le due piattaforme.

6 Back-End

Descriviamo ora tutto quelle che sta sotto l'interfaccia grafica, come l'interfacciamento con il server e la gestione della cache del tablet.

6.1 Database

Il database locale è tipo SQL versione Lite che mi permette di avere dimensioni minori, indicato per il mobile.

L'architettura che gestisce il database è stata progettata a livelli, ovvero ad ogni aggiornamento dell'applicazione che aggiunge o modifica campi all'interno del database, viene fatto un nuovo manager che estende quello precedente.

Il manager non è altro che un contenitore di informazioni riguardanti il database. Contiene tutte le stringhe con i nomi dei campi delle singole tabelle, in questo modo è possibile creare manualmente tutto il database conoscendo perfettamente ogni singolo aspetto. Attualmente il programma contiene sei versioni di manager, ovvero le sei grandi modifiche apportate al database dalla creazione del database ad oggi.

Ogni versione del manager contiene all'interno una variabile denominata DB_VERSION che mi permette di tenere traccia della versione del database, soprattutto nel caso dell'aggiornamento dell'applicazione saprò se devo aggiornarlo o no, per far sì che non si presentino dei problemi nella lettura dei dati.

Nella prima versione della classe viene creato il database, tutte le relative tabelle utilizzate di base e i trigger.

```
@Override
            public void onCreate(SQLiteDatabase db) {

                db.execSQL(create_Fornitori());
                db.execSQL(create_Clienti());
                db.execSQL(create_Prodotti());
                db.execSQL(create_Sconti());
                db.execSQL(create_Ordini());
                db.execSQL(create_ProdottiOrdini());
                db.execSQL(create_Agente());
                db.execSQL(create_Config());
                db.execSQL(create_trigger_fk_prodottiordini());
                db.execSQL(create_trigger_fk_sconti_cliente());
                db.execSQL(create_trigger_fk_sconti_fornitore());
                db.execSQL(create_trigger_fk_sconti_prodotto());
                db.execSQL(create_trigger_elimina_sconti());
                db.execSQL(create_trigger_elimina_prodotti());
              db.execSQL(create_trigger_elimina_prodottiordine());
                }
```

Le versioni successive del manager estenderanno quella precedenza cosi:

```
public class OSDBManagerV3 extends OSDBManagerV2
```

Ad ogni nuova classe deve essere mantenuto il riferimento allo stesso database, altrimenti verranno create più versioni. Questo è possibile grazie all'estensione della classe e alla chiamata della funzione padre.

Nel caso di modifica del database all'aggiornamento dell'applicazione devono essere fatte delle modifiche che sono inserite in un metodo denominato onUpgrade, della classe "SQLiteOpenHelper", che richiama tutti i padri e poi esegue le proprie operazioni, come nel caso di un avanzamento di versione dalla 2 alla 3:

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
      super.onUpgrade(db, oldVersion, newVersion);
      if (oldVersion < 3 && newVersion >= 3) {
            db.execSQL(addColumn(PRODOTTI_TABLE_NAME,
PRODOTTI_UNITA, "text"));
            db.execSQL(addColumn(PRODOTTI_TABLE_NAME,
PRODOTTI_IVA, "double"));
            db.execSQL(addColumn(PRODOTTIORDINI_TABLE_NAME,
                                    PRODOTTIORDINI_UNITA, "text"));
            db.execSQL(addColumn(PRODOTTIORDINI_TABLE_NAME,
                                    PRODOTTIORDINI_IVA, "double"));
```

```
            db.execSQL(addColumn(CONFIG_TABLE_NAME, CONFIG_IVA,
"double"));
            db.execSQL(addColumn(ORDINI_TABLE_NAME,
ORDINI_CL_NOTE, "text"));
                  }
            }
```

Attualmente la versione più recente è la 6 ed è così costituita:

```
public class OSDBManagerV6 extends OSDBManagerV5 {
      protected static final int DBVERSION = 6;
      protected static final String TAG = "6.0TAG";
      public static String CACHE_TABLENAME = "Cache";
      public static String CACHE_ID = "_id";
      public static String CACHE_CID_ELEMENTO = "cid_elemento";
      public static String CACHE_ID_ELEMENTO = "id_elemento";
      public static String CACHE_DATA = "data";
      public static String CACHE_OPERAZIONE = "operazione";
      public static String CACHE_TIPO_ELEMENTO = "tipo_elemento";
      public static String SOTTOSCRIZIONI_TIPO = "tipo";

      public OSDBManagerV6(Context context) {
            super(context);
      }

      @Override
      public void inizializza() {
            helper = new OSOpenHelperV6(context, DBVERSION);
            db = helper.getWritableDatabase();
      }
      protected class OSOpenHelperV6 extends OSOpenHelperV5 {
            public Context contesto = null;
            public OSOpenHelperV6(Context context, int version) {
                  super(context, version);
                  contesto = context;
            }
            @Override
            public void onCreate(SQLiteDatabase db) {
                  super.onCreate(db);
                  db.execSQL(create_Cache());
                  db.execSQL(addColumn(SOTTOSCRIZIONI_TABLE_NAME,
                              SOTTOSCRIZIONI_TIPO, "text"));
                  db.execSQL(addColumn(ORDINI_TABLE_NAME,
IDAZIENDA, "integer"));
                  db.execSQL(addColumn(PRODOTTIORDINI_TABLE_NAME,
IDAZIENDA,
                              "integer"));
```

```
                db.execSQL(addColumn(PRODOTTIORDINI_TABLE_NAME,
CODAZIENDA, "text"));
                db.execSQL(delete_trigger());
            }
            @Override
            public void onUpgrade(SQLiteDatabase db, int
oldVersion, int newVersion) {
                super.onUpgrade(db, oldVersion, newVersion);
                if (oldVersion < 6 && newVersion >= 6) {
                    db.execSQL(create_Cache());
            db.execSQL(addColumn(SOTTOSCRIZIONI_TABLE_NAME,
                                SOTTOSCRIZIONI_TIPO, "text"));
            db.execSQL(addColumn(ORDINI_TABLE_NAME, IDAZIENDA,
"integer"));
            db.execSQL(addColumn(PRODOTTIORDINI_TABLE_NAME,
IDAZIENDA,
                                "integer"));
            db.execSQL(addColumn(PRODOTTIORDINI_TABLE_NAME,
CODAZIENDA,
                                "text"));
            db.execSQL(delete_trigger());
                }
            }

            public String create_Cache() {
            return "create table " + CACHE_TABLENAME + "( " +
CACHE_ID+ " integer primary key autoincrement not null, "  +
CACHE_CID_ELEMENTO + " integer, " + CACHE_ID_ELEMENTO + "
integer, " + CACHE_DATA + " text, " + CACHE_OPERAZIONE + "
text," + CACHE_TIPO_ELEMENTO + " text);";
            }
            public String delete_trigger() {
                return "DROP TRIGGER elimina_sottoscrizione";
            }
      }
}
```

Questa è la gestione delle tabelle all'interno del database, ma ci vuole qualcosa che mi consenta di controllare la versione corrente e che contiene la chiama ai metodi che andranno a leggere o modificare il database. Ecco che quindi entra in gioco un controller, che ha la funzione di inizializzare il manager ad ogni avvio dell'applicazione, nel caso sia già presente un database sarà restituito il riferimento.

```
OSDBManagerV6 manager = new OSDBManagerV6(context);
```

Context è il riferimento al contesto che è attualmente attivo nella applicazione.

Questa classe controller mi deve contenere anche tutti i metodi che saranno richiamabili da qualsiasi punto dell'applicazione semplicemente richiamando il controller seguito dal metodo voluto, esempio:

```
Controller.salva(elemento);
```

Questa chiamata sarà indirizzata al seguente metodo contenuto nella classe:

```
public static boolean salva(Element e) {
            return wrapper.insertOrUpdate(e);
      }
```

E' un elemento fondamentale che permette di essere un tramite da i metodi del database e quelli per l'elaborazione dei dati.

Inizializzando come segue il controller ho la possibilità, con una semplice modifica, di aggiornare il manager del database semplicemente cambiando un valore.

```
public class Controller {

      public static OSDBWrapperV6 wrapper;
      static OSDBManagerV6 manager;
      public static final String TAG = "Controller";
      public static Context context;


…
}
```

Ma per far sì che il controller possa fare da tramite e passare i dati del db serve una classe che contenga tutti i metodi che eseguono le query SQL del database, anche questa classe, come quella del manager, dovrà poter essere estesa in base alla versione del database. Ed ecco, come è possibile vedere nel riquadro precedente, entra in gioco il wrapper.

Tutti i metodi che sono all'interno delle versioni del wrapper puntano, ovviamente, allo stesso database e posso eseguire operazioni di lettura, modifica e cancellazione nel database. Una funzione molto usata è la getElement che mi restituisce tutti i dati dell'elemento voluto.

```
public String getElementName(Element element) {
            String ret = "";
            String nome = this.getNomeColumnName(element);
            if (nome != null) {
      Cursor c =manager.db.query(this.getTableName(element),
```

```
            new String[] { nome }, this.getIdColumnName(element)
      + " = " + element.id, null, null, null, null);
                  if (c.getCount() == 1) {
                        c.moveToFirst();
                        ret = c.getString(0);
                  }
                  c.close();
            }
            return ret;
      }
```

Si noti che viene passato alla funzione un elemento di tipo Element, inizializzato così:

```
public abstract class Element {

      public long id;
      //public String table_name;
      public Element(){
            id = -1;
      }
```

Questo è un elemento generico che può essere un cliente, un fornitore, un prodotto, un ordine, un agente e altri tipi di dati che richiedono di base solamente un campo id per l'identificazione.

Poi esistono i tipi di dato di classe SubscriptionElement che estende Element:

```
public class SubscriptionElement extends Element{
      public String codAzienda;
      public int idAzienda;

      public SubscriptionElement() {
            super();
            codAzienda = null;
            idAzienda = -1;
      }
      public SubscriptionElement(String codAzienda, int
idAzienda) {
            super();
            this.codAzienda = codAzienda;
            this.idAzienda = idAzienda;
      }
      public boolean isInSubscription() {
            return codAzienda != null && ! codAzienda.equals("");
      }
}
```

Questa classe mi permette di avere oltre all'identificativo di Element, anche una relazione con la sottoscrizione inserita.

CodAzienda è l'identificativo dell'azienda per cui lavora l'agente e idAzienda è il codice univoco dell'elemento nel database remoto.

Quindi ogni elemento che deve mantenere dei dati della sottoscrizione avrà la propria classe che estenderà SubscriptionElement. Come nel caso di un prodotto:

```
public class Prodotto extends SubscriptionElement {

	public String codice;
	public String descrizione;
	public double prezzo;
	public double quantitaMin;
	public Fornitore fornitore;
	public String unita;
	public double iva;
…
}
```

6.2 Interazione con il database locale

Il database locale è un database SQLiteDatabase che risiede nella cartella dei dati dell'applicazione, quindi non è leggibile da un file manager semplice.

Il file viene posizionato nella cartella /data/data/packageApp/databases ed ha il nome che è stato inserito manualmente.

All'interno del database vengono salvate tabelle, indici, views ecc.

Contiene tutti i metodi per creare, eliminare ed eseguire query SQL e contiene algoritmi per migliorare le performance.

Per la creazione di una tabella si può utilizzare il metodo execSQL che non può essere usato per ritornare dei dati quindi non utilizzabile ai fini di una selezione.

Esempio di creazione di una tabella:

```
public String create_Clienti() {
	return "create table " + CLIENTI_TABLE_NAME + " (" +
CLIENTI_ID + " integer primary key autoincrement not null, "+
CLIENTI_NOME + " text not null, " + CLIENTI_RESPONSABILE + "
text, " + CLIENTI_PARTITAIVA + " text);";
		}
db.execSQL(create_Clienti());
```

All'interno della variabile db di tipo SQLiteDatabase è contenuto il riferimento al file e mantiene tutti i dati relativi, come percorso e contatore della tabelle.

Come detto in precedenza il database è stato modificato nel corso dell'avanzamento delle versioni, quindi si è reso necessario l'utilizzo di un metodo che mi permettesse di aggiungere le entità ad una o più tabelle. Questo viene fatto sempre tramite la execSQL ma passandogli una stringa, ovvero una query differente da quella della creazione della tabella.

E' stato creato manualmente un metodo che ritorna la query e come parametri di ingresso ha il nome della tabella, il nome della colonna e il tipo di dato.

```
public String addColumn(String table, String column, String
type) {
      return "ALTER TABLE " + table + " ADD COLUMN " + column + "
"+ type + ";";
            }
```

Per richiamarlo:

```
db.execSQL(addColumn(CONFIG_TABLE_NAME, CONFIG_LASTSYNCNEWS,
"text"));
```

Questi sono i metodi utilizzati per modificare la struttura delle varie tabelle.

Le funzioni utilizzate per la selezione o modifica dei dati sono state implementate nel wrapper.

Come detto in precedenza, anche questa classe è stata espansa re implementata più volte a seconda della versione del database. Quindi i metodi sono stati sovrascritti, soprattutto quelli per il ritorno dei dati, per far sì che anche i dati nuovi potessero essere selezionati.

Per la selezione dei record all'interno di una tabella si è reso necessario utilizzo di un cursore, ovvero un'interfaccia che fornisce lettura e scrittura per il set di dati che sono stati restituiti da una selezione nel database. Questo mi permettere di popolare velocemente un adapter per fornire i dati, per esempio, ad una tabella.

L'utilizzo di un Cursor per la selezione può essere usato così:

```
public Cursor getNews() {

Cursor c = manager.db.query(OSDBManagerV5.MESSAGGI_TABLE_NAME,
            new String[] { "*" }, null, null, null, null,
            OSDBManagerV2.IDAZIENDA + " DESC");
            return c;
      }
```

Questa funzione mi restituisce tutti i campi nella tabella messaggi senza filtri ma in ordine decrescente in base all'idAzienda.

Per l'aggiornamento in un campo, per esempio l'ultima data di sincronizzazione dell'App, non utilizzerò un cursore ma una execSQL. In questo caso per poter salvare la data della sync in un modo standard è stato fatto prima un parse.

```
public void saveSyncNews() {
      SimpleDateFormat formatter = new SimpleDateFormat(format);
            String data = null;
            data = formatter.format(new Date());
            manager.db.execSQL("UPDATE " +
OSDBManager.CONFIG_TABLE_NAME + " SET " +
OSDBManagerV5.CONFIG_LASTSYNCNEWS + " = '" + data + "';");
      }
```

Il database ha come metodo proprio anche la funzione update, quindi è possibile anche utilizzarlo per la modifica.

```
public boolean updateElement(Element element) {
String where = getIdColumnName(element) + " = " + element.id;
long ret = manager.db.update(this.getTableName(element),
this.getContentValues(element, false), where, null);
            if (ret == -1)
                  return false;
            return true;
      }
```

Questa è una funzione che passato in ingresso un qualsiasi tipo di dato mi aggiorna il record corrispondente che caso esista, altrimenti lo aggiunge.

GetContentValues è una funzione che a seconda della classe di Element mi ritorna la lista di campi da aggiornare.

Per l'eliminazione possiamo utilizzare una funzione analoga che non fa altro che prendere anch'essi un elemento generico, cercarlo all'interno della tabella corrispondete ed eliminarlo nel caso sia presente.

```
public boolean deleteElement(Element element) {
int deleted = manager.db.delete(this.getTableName(element),
            this.getIdColumnName(element) + " = " + element.id,
null);
            if (deleted != 1)
                  return false;
            return true;
      }
```

6.3 Invio dati per la sincronizzazione

L'invio di un ordine creato e salvato al server viene fatto attraverso una richiesta HTTP POST con allegate tutte le chiavi che indicato ogni singolo valore dell'ordine.

Il metodo inviaOrdine riceve in ingresso una serie di variabili: l'oggetto ordine, l'oggetto agente che l'ha creato, il fornitore, il cliente, l'identificato del dispositivo e la lingua dell'applicazione. Crea un oggetto di tipo HttpPost che mi permetterà di inviare la richiesta al server e come indirizzo web utilizza http://business.ordersender.com/sottoscrizione/sendmail, poi devono essere allegati tutti i dati, per far ciò viene creata una struttura che possa contenere una coppia chiave-valore. Quello che serve è una chiave di tipo NameValuePair, ma essendo una lista di valori avrò bisogno di più coppie.

```
List<NameValuePair> nameValuePairs = new
ArrayList<NameValuePair>(2);
```

L'inserimento del valore all'interno della lista viene fatto cosi:

```
nameValuePairs.add(new BasicNameValuePair("numero",
ordine.numero));
```

Così facendo per tutte le coppie di dati si potrà avere un export totale dei dati nella richiesta HTTP POST, utilizzando:

```
httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs,
"UTF-8"));
```

Per far si che la riceista sia inoltrara e averla possibilità di avere un response mi dovrò creare un oggetti di tipo HttpResponse.

```
      HttpResponse response = httpclient.execute(httppost);
```

All'interno avrò la risposta del server che mi indicherà se la richiesta è arrivata a destinazione oppure ci sono stati dei problemi, la verifica è possibile tramite un semplice controllo:

```
if (response.getStatusLine().getStatusCode() ==
HttpStatus.SC_OK)
                  return true;
```

Nel caso l'invio con sia dell'ordine ma di un file contenente i campi modificati allora non potrò utilizzare lo stesso metodo utilizzato in precedenza ma dovrò creare uno stream di dati in uscita, verso il server, che mi permetta di inviare il file.

Non avremo più un oggetto di tipo HttpPost ma avremo una semplice HttpURLConnection collegata a un DataOutputStream. Come parametri per l'invio avrò bisogno di sapere il buffer massimo che posso utilizzare.

```
            int maxBufferSize = 1 * 1024 * 1024;
```

Un array che mi contenga i byte:

```
            byte[] buffer;
```

Uno stream di input verso un file temporaneo che poi invierò.

```
FileInputStream fileInputStream = new FileInputStream(file);
```

Infine un url a cui inviarlo, collegato all' HttpURLConnection.

```
URL url = new URL(syncSmallURL);
connection = (HttpURLConnection) url.openConnection();
```

Tutti i dati di accesso dell'agente che i dati del database modificati dovranno essere inseriti all'interno del file, codificati in base alla codifica della funzione addetta sul server web.

```
String lineEnd = "\n";
String twoHyphens = "--";
String boundary = "PASDDSAIHG";
…
outputStream.writeBytes("Content-Disposition: form-data;
name=\"username"+ "\"\r\n\r\n" + username + lineEnd);
…
outputStream.writeBytes("Content-Disposition: form-data;
name=\"file\";filename=\"" + file.getAbsolutePath() + "\"" +
lineEnd);
…
```

Anche qui avremo un response dal server che servirà per comunicare l'eventuale errore di invio nella interfaccia grafica.

```
if (serverResponseCode == 200) { // posso avere 201..?
     Toast.makeText(context, "Errore nell'invio del file.
Riprovare.", Toast.LENGTH_LONG).show(); }
```

6.4 Gestione della cache

La cache è molto utile nell'ottica di modifiche offline.

Non tutti i devices dispongono di una connessione ad Internet in mancanza di una rete wireless, quindi si è reso necessario creare una struttura che mi potesse contenere i dati che sono stati modificati ed alla prima modifica, con connessione ad Internet, inviare il tutto al server che si occuperà di confrontare le date di modifica e aggiornare il database remoto.

Questa struttura è stata pensata semplicemente come una tabella, come quella dei clienti o dei fornitori, ma con l'opportunità di mantenere ogni tipo di elemento.

Il tipo di elemento cache, come gli altri, estende la classe SubscriptionElement perché è associato ad un agente e quindi ad una azienda.

La classe cache è così composta:

```
public class Cache extends SubscriptionElement {

	public long cid_elemento;
	public long id_elemento;
	public String data;
	public String operazione;
	public String tipo_elemento;

	public Cache(long cid_elemento, long id_elemento, String
data, String operazione, String tipo_elemento) {
		super();
		this.cid_elemento = cid_elemento;
		this.id_elemento = id_elemento;
		this.data = data;
		this.operazione = operazione;
		this.tipo_elemento = tipo_elemento;
	}

	public Cache() {
		super();
	}
}
```

Come si può vedere, contiene due riferimenti all'oggetto modificato, cid_elemento e id_elemento.

Id_elemento è il codice indentificato all'interno del tablet mentre cid_elemento, se esiste, è il codice identificato nel database remoto.

Data è il campo contenente la data di modifica perché nel caso il dato sia stato modificato sia sul tablet che dal sito web, viene preso per buono quello che ha la modifica più recente.

Operazione è una stringa che indica al server web il tipo di operazione eseguita e può essere ins nel caso di un inserimento, upd se c'è stato un aggiornamento e del se stato eliminato.

La creazione di un elemento della cache viene fatto in questo modo:

```
Cache cache = new Cache();
            cache.cid_elemento = el.id;
            cache.id_elemento = el.idAzienda;
            cache.operazione = tipo;
            if (el instanceof Fornitore)
                  cache.tipo_elemento = OSDBManagerV6.FORNITORI_TABLE_NAME;
            else if (el instanceof Prodotto)
                  cache.tipo_elemento = OSDBManagerV6.PRODOTTI_TABLE_NAME;
            else if (el instanceof Sconto)
                  cache.tipo_elemento = OSDBManagerV6.SCONTI_TABLE_NAME;
            else if (el instanceof Cliente)
                  cache.tipo_elemento = OSDBManagerV6.CLIENTI_TABLE_NAME;
            else if (el instanceof Ordine)
                  cache.tipo_elemento = OSDBManagerV6.ORDINI_TABLE_NAME;
            else if (el instanceof ProdottoOrdine)
                  cache.tipo_elemento
=OSDBManagerV6.PRODOTTIORDINI_TABLE_NAME;

            SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");
            cache.data = format.format(new Date());
            setCache(cache);

…

public static void setCache(Cache element) {
            wrapper.insertOrUpdate(element);
      }
```

Inizialmente viene creato un oggetto di tipo cache ed inizializzato ma non contiene nulla, poi vengono passati i due identificativi e il tipo di operazione.

Vengono fatti una serie di controlli in base al tipo di elemento e inserito quello corretto, poi segue la data. Per ultimo c'è la scrittura dell'oggetto nel database.

Come detto in precedenza, questo serve per permettere di mantenere le modifiche e sincronizzarle. Il procedimento di immissione della modifica all'interno della cache viene fatto sempre, poi avviene il controllo della connessione. Nel caso sia

disponibile allora bisogna creare una stringa contenente tutti i record modificata ed inviarla al server. Insieme alla stringa saranno passati il codice della sottoscrizione, username, password, il codice identificativo del dispositivo e la versione dell'applicazione.

```
if (Controller.testConnessione(context)) {

String data = Controller.svuotaCache();
HashMap<String, String> map = new HashMap<String, String>();
ArrayList<Sottoscrizione> array =
Controller.getSottoscrizionePerTipo("all");
     for (Sottoscrizione s : array)
                    {
                         map.put("codice", s.codice);
                         map.put("username", s.username);
                         map.put("password", s.password);

          TelephonyManager tm = (TelephonyManager) context
               .getSystemService(Context.TELEPHONY_SERVICE);
                         String deviceID = tm.getDeviceId();
                         map.put("udid", deviceID);
          PackageInfo pInfo = context.getPackageManager()
               .getPackageInfo(context.getPackageName(), 0);
                         Integer version = pInfo.versionCode;
          map.put("version", version.toString());

                         data.replace("&", "%26");
                         data.replace("+", "%28");
                         map.put("element", data);

                    }
          new InvioDialog(context, map, InvioDialog.SYNCOUT);
```

Il tutto viene passato alla classe InvioDialog che si occupa di recapitarlo al server, utilizzando l'indirizzo web dedicato.

6.5 Sincronizzazione bidirezionale

Il sistema dovrà essere in grado di memorizzare una serie di sottoscrizioni. Le sottoscrizioni sono caratterizzate da un nome, uno username, una password e un codice azienda.

Il programma dovrà poi verificare se ci sono aggiornamenti rispetto ai dati già presenti nella base dati con una chiamata HTTP apposita e in caso ci siano (la risposta alla chiamata in questo caso sarà updateok, altrimenti updateno), scaricarli: utilizzando una richiesta HTTP GET si otterrà un file simile a quello dell'importazione, ma contenente tutte le tipologie di dato (clienti, fornitori, prodotti e anche, in questo caso, sconti), le loro relazioni e l'indicazione riguardo se ogni dato debba essere aggiunto, modificato o eliminato nel database. Questo è ciò che avviene durante una normale sincronizzazione.

I dati associati a una sottoscrizione sono trattati in modo diverso dai dati inseriti a mano o importati dall'utente e devono essere riconoscibili in altre parti del programma. Nella gestione questi dati sono visualizzati e modificabili (poiché la sincronizzazione è bidirezionale: il flusso va dal web al programma e viceversa).

La sincronizzazione bidirezionale permette all'utente di avere sempre sottomano tutti i dati sia che questi siano stati creati dal tablet sia dal sito web. Questo è un grande vantaggio perché come nel caso di un abbonamento che permetta di associare più dispositivi alla stessa sottoscrizione, appena vi è una modifica, dopo la sincronizzazione, è possibile averla ovunque.

Questa aspetto non è da sottovalutare soprattutto nel caso di smarrimento di un device, i dati sono tutti residenti nel database remoto e una volta reinstallata l'applicazione e reintrodotto la sottoscrizione sarà possibile avere una immagine esatta dell'ultima sincronizzazione.

7 Conclusioni e sviluppi futuri

Lo spunto per questa testi è stato dato un'esigenza dell'azienda di sviluppare un'applicazione per mobile in grado di offrire un servizio impeccabile ad ogni agente e rappresentante.

Per creare questo tipo di applicazione è stato necessario utilizzare più piattaforme e linguaggi di programmazione, quali Java e Objective-C per la parte mobile e PHP per la parte server, senza dimenticare HTML e PHP per la parte di interfaccia web.

La progettazione su iOS non si discosta molto dalla normale progettazione ad oggetti, anzi costringe ad utilizzare il pattern MVC, rendendo l'approccio più ordinato. La programmazione sulla piattaforma però offre anche numerose astrazioni, oltre a quella ad oggetti, che si possono prestare ad altri scopi.

La progettazione su Android offre una serie di nuove astrazioni, quali Activity, Service, Intent e Broadcast Receiver, che prevedono un'interazione a message-passing tra componenti che possiedono una specifica semantica.

Ciò implica che anche per applicazioni concentrate è necessario progettare le varie parti in modo che comunichino a message passing e occorre perciò decidere come separare le funzionalità per inserirle nel componente che possiede la semantica adatta.

Sviluppi futuri potranno essere sicuramente la creazione di statistiche anche nelle applicazioni mobile, in modo da fornire tutto il possibile subito e ovunque. Sicuramente a questo livello l'insieme applicazione e servizio web può essere un pacchetto completo per una azienda che decide di personalizzare il sistema.

Bibliografia

- Android Developers Guide
  http://developer.android.com/

- Sito Order Sender
  http://www.ordersender.com

- Sito Order Sender Business
  http://business.ordersender.com