ALMA MATER STUDIORUM - UNIVERSITÁ DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI INGEGNERIA E ARCHITETTURA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA E SCIENZE INFORMATICHE

TITOLO DELLA TESI

MOBILE BUSINESS APPLICATION

Tesi in

MOBILE WEB DESIGN

Relatore

Dott. Mirko Ravaioli

Correlatore

Prof.ssa Antonella Carbonaro

Presentata da

Tomas Fedeli

Sessione: III

Anno accademico: 2013/2014

INDICE

INTRODUZIONE

Le nuove tecnologie ICT (Information and Communication Technology), grazie alle loro caratteristiche di portabilità, accessibilità e adattabilità, permettono l'abilitazione di nuovi modelli di lavoro orientati allo Smart Working.
Con questo termine viene identificato un nuovo modo di lavorare generato dall'elemento portante e costante di ogni autentico salto di paradigma, ovvero la tecnologia. Quest'ultima non deve essere vista in sostituzione della persona, ma semplicemente come strumento in supporto ad essa, con l'obiettivo di semplificarne il lavoro e, allo stesso tempo, di aumentarne la produttività.
Il cambiamento associato allo Smart Working richiede tempo, ma agendo contemporaneamente su cultura e comportamenti delle persone, può avere un impatto profondo e duraturo sull'organizzazione e le prestazioni.
Le tecnologie più utilizzate per abilitare lo Smart Working sono quelle che supportano la collaborazione, la socialità e l'accessibilità delle informazioni, permettendo alle persone di lavorare in modo efficace a distanza e all'esterno della sede dell'azienda. In particolare, tra queste vi sono le Mobile Business App, le iniziative Social e il Cloud Computing, paradigma tecnologico fondamentale per garantire l'accessibilità a dati e applicazioni da qualunque luogo e con qualsiasi device.
Inoltre, risultano fondamentali i device mobili che rendono possibile accedere alle informazioni e lavorare anche al di fuori di spazi e orari di lavoro tradizionali. Grazie alle capacità di questi nuovi dispositivi, come smartphone e tablet, nasce quindi una nuova grande opportunità, ovvero quella di mantenere e processare dati al loro interno, in sostituzione dei classici personal computers.
Quindi, il nuovo mondo mobile, insieme alle recenti tecnologie sopracitate, ha un inevitabile impatto su ciò che viene definito il Business Process. Per Business Process si intende l'insieme delle attività svolte all'interno di un'azienda che dato un certo insieme di risorse portano alla realizzazione di un prodotto finale destinato

ad un determinato cliente collocato internamente o esternamente all'organizzazione.

Il processo aziendale è solitamente allineato con le tecnologie più attuali e, di conseguenza, le nuove opportunità possono contribuire ad un cambiamento parziale o totale del Business Process.

Concludendo, la tecnologia mobile sta ricevendo una significativa attenzione nel mondo del Business e, ovviamente, dell'Information Technology, grazie alle sue capacità di offrire notevoli vantaggi in termini economici, di semplificare e velocizzare la ricerca del cliente e, soprattutto, di condurre la reingegnerizzazione di processi aziendali e prodotti software.

1 IL MONDO MOBILE E GLI EFFETTI SUL BUSINESS

Internet ha cambiato la comunicazione, i social media hanno cambiato Internet, il mobile cambierà il mondo. Questo è quanto pronunciato da Jack Haber, vice presidente di "Global Adversiting Digital for Colgate-Palmolive", durante una recente conferenza internazionale di IAB (Interactive Advertising Bureau), associazione volta al settore pubblicitario che si occupa dello sviluppo di standards per le industrie, conduzione di ricerche e supporto per la pubblicizzazione di industrie tramite il Web.

Il fenomeno di migrazione degli utenti Internet da desktop a mobile è ormai conclamato per i Paesi del vecchio mondo, mentre per i Paesi del nuovo mondo, quelli che si approcciano ora all'utilizzo di Internet (ad esempio, Cina e Brasile), è il punto d'accesso principale.

Tuttavia, il termine "Mobile" racchiude al suo interno qualcosa di più grande e più articolato rispetto a una serie di device; rappresenta una vera e propria specie digitale che popolerà un mondo diverso. Si può quindi affermare di essere entrati in una nuova era.

A sostegno di questa tesi, vi sono anche i ricercatori di Gartner, società multinazionale leader mondiale nella consulenza strategica, ricerca e analisi nel campo dell'Information Technology. Essi sostengono che i dispositivi mobili sono destinati a diventare il principale strumento per le comunicazioni e l'accesso ai contenuti online.

Entro il 2018, oltre il 50% degli utenti utilizzerà uno smartphone o un tablet per svolgere qualsiasi attività online. Il pc, sempre a detta dei ricercatori Gartner, sarà impiegato solo per compiti specifici complessi.

1.1 RETI

1.1.1 Rete cellulare

Rappresenta la rete per eccellenza che supporta la comunicazione tra telefoni mobili. Una rete cellulare è costituita da un numero variabile di celle, che forniscono la copertura radioelettrica e il collegamento tra i terminali mobili e la rete telefonica fissa. Il numero delle celle e la loro grandezza dipende dalla quantità di traffico presente in una data zona.
La rete cellulare ha avuto una rapida evoluzione, della quale è possibile e utile identificare cinque stadi principali:

0G(1952) Questi primi sistemi di telefonia mobile si distinguono dai precedenti sistemi di radiotelefonia in quanto non erano disponibili solo per network chiusi, come ad esempio le radio della polizia, ma anche per l'utilizzo commerciale.

1G(1980) La prima generazione di rete cellulare si basava su standard di comunicazione analogica: il segnale radio inviato dal telefono veniva ricevuto da torri radio, che attraverso un segnale di tipo digitale si connettevano poi ad altre torri e al resto del network.

2G(1989) Gli standard di seconda generazione sanciscono il passaggio dal segnale radio analogico a quello digitale. Adottato in Finlandia nel 1991, questo standard ha introdotto il trasferimento del segnale vocale attraverso la commutazione a pacchetto, permettendo la trasmissione della voce in digitale.
Le tecnologie 2G hanno permesso un grande passo in avanti grazie alla migliore copertura e alle funzionalità aggiuntive, prima fra tutte gli sms.
Il primo standard sviluppato è stato il GSM che permetteva di usufruire del telefono cellulare anche all'estero e di memorizzare i dati sulla carta sim, rendendoli disponibili all'utente anche nel caso dell'utilizzo di altri dispositivi. Oltre alla rete gsm sono state sviluppate tecnologie più avanzate della seconda generazione: GPRS e EDGE.

3G(2001) La terza generazione, la cui introduzione è stata sancita dall'ottimizzazione delle tecnologie edge, è contraddistinta dalla combinazione di commutazione a circuito e a pacchetto.

Questa tipologia di rete era in grado di fornire servizi più avanzati e allo stesso tempo network a maggiore capacità, con una velocità di trasmissione dei dati superiore. I servizi che determinarono il successo di questa tecnologia erano legati alla possibilità di downloading.

Lo standard simbolo di questa generazione è l'UMTS.

4G(2006) Dal 2006 vengono presentati in Giappone i primi prototipi di smartphone che supportano nuove tecnologie di trasmissione dei dati, ovvero LTE e HSOPA.

Le tecnologie 4G rappresentano un insieme di standard wireless compatibili con una totale commutazione a pacchetto e sostituiranno progressivamente i network presenti permettendo una trasmissione in qualsiasi luogo e momento di tutti i contenuti voce, dati e veicolati via streaming. Le tecnologie 4G sono progettate per fornire agli utenti accesso a una molteplicità di contenuti, come streaming audio e video, digital video broadcast, videochat e altro ancora.

1.1.2 Rete circoscritta

WLAN E WIFI La tecnologia WLAN (Wireless Local Area Network) consiste in un accesso a internet a corto raggio reso disponibile attraverso access point senza fili, detti anche hotspot.

Il Wi-Fi, sebbene sia spesso associato al WLAN, ne rappresenta in realtà una particolare tecnologia, certificata dalla Wi-Fi Alliance, ente non profit con l'obiettivo di promuovere, a livello globale, l'adozione di uno standard comune per le WLAN ad alta velocità.

1.1.3 Rete locale

INFRAROSSI(IRDA) Per anni è stato l'unico metodo di trasmissione radio per dispositivi mobili.

Si tratta di un sistema di trasmissione bidirezionale tra dispositivi a infrarossi, che devono essere posizionati lungo la cosiddetta LoS (line of sight, ovvero in condizioni di visibilità reciproca). Questa tipologia di rete presenta alcune limitazioni fondamentali: raggio d'azione molto limitato (massimo 1-2 metri), impossibilità per la radiazione di attraversare barriere solide (anche una lastra di vetro influisce sulla trasmissione delle informazioni) e bassa velocità di trasmissione (mediamente 4 Mbit/s).

BLUETOOTH Questa tipologia di rete locale, sviluppata nel 1994 ma diffusasi solamente intorno alla fine degli anni Novanta, si basa su un broadcast radio che permette ai dispositivi nelle vicinanze il riconoscimento reciproco e lo scambio di dati.

È possibile collegare tra loro molteplici dispositivi, in quella che viene comunemente chiamata Personal Area Network (PAN).

NFC(NEAR FIELD COMMUNICATION) Consiste in un insieme di tecnologie che forniscono connettività wireless a corto raggio. Permette il riconoscimento senza contatto e offre la possibilità di uno scambio bidirezionale tra due apparecchi NFC, detti initiator e target, rispettivamente il dispositivo che dà inizio alla comunicazione e quello che la riceve. Se accostati, i due apparecchi possono dare luogo ad altre tipologie di interazioni:

- card emulation, il device agisce come una card contactless, ossia una carta elettronica il cui funzionamento non necessita il contatto con il relativo lettore, ma la sola vicinanza;
- p2p mode, due dispositivi danno inizio a uno scambio di informazioni in modalità peer-to-peer.

Le applicazioni di questa tecnologia sono molteplici:

- mobile ticketing, ad esempio nel trasporto pubblico;

- mobile payment, il device funge da sistema di pagamento, ad esempio come una carta di credito;
- biglietteria elettronica, ad esempio per voli aerei, eventi, concerti e così via.

Questi sono solo alcuni esempi, ma mostrano la versatilità e le grandi potenzialità di questa tipologia di rete in contesti applicativi differenti.

1.2 DISPOSITIVI

Tra i dispositivi mobile più importanti, attualmente, possono essere individuate tre famiglie: PC portatili, telefoni cellulari e wearable technologies. Questa classificazione è strettamente correlata con la loro diffusione nel tempo: la prima categoria ha avuto una forte diffusione nel passato, ma attualmente si tratta di dispositivi utilizzati solo per compiti specifici, in quanto pienamente sostituiti dalla classe di telefoni cellulari comprendente dispositivi affermatissimi come smartphone e tablet, e, in fine, l'ultima famiglia rappresenta ciò che potrebbe avere una certa rilevanza all'interno del mondo mobile in un futuro prossimo.

PC PORTATILI I pc portatili possono essere visti come i primi dispositivi ad introdurre l'idea della mobilità nel settore tecnologico. Il primo prodotto incentrato sul concetto di portabilità fu l'Epson HX-20, proposto come monoblocco all'interno di una valigetta delle dimensioni di un foglio A4. Tuttavia, per la svolta definitiva nella categoria dei laptop bisogna attendere l'inizio degli anni Novanta, con il lancio da parte di Apple del Powerbook, che introdusse il nuovo sistema di movimento e puntamento sul display (il balltrack) e, nel design, lo spazio per l'appoggio delle mani durante l'uso. Da quel momento, di anno in anno, aumentò notevolmente la vendita di portatili fino ad arrivare al 2008, nel quale il numero di portatili venduti pareggiò quello dei desktop PC. Il PC portatile diventa cosi il più importante prodotto per poter accedere in qualsiasi momento e da qualsiasi luogo a contenuti digitali:

oltre che essere uno strumento di lavoro, rappresenta un centro d'intrattenimento con la possibilità di processare musica, immagini, video e molto altro. Dal 2007 al 2011, ci fu la diffusione di una nuova tipologia di portatile dalle dimensioni ridotte, ovvero il Netbook. Questi ultimi dispositivi meritano di essere ricordati come il risultato degli enormi progressi nella miniaturizzazione dei componenti hardware e per l'emergere del fenomeno del cloud computing, grazie al quale è possibile alleggerire di molte applicazioni i PC client durante la navigazione in Internet.

TELEFONI CELLULARI Nel 1984 esordì sul mercato il primo telefono cellulare destinato al consumo di massa, il Motorola DynaTAC 8000X. Tuttavia, il prezzo altissimo lo rendeva un lusso per pochi e la necessità di una borsa apposta per trasportarlo ne limitava la portabilità.

Successivamente, fu la stessa Motorola a produrre il primo telefono dalle dimensioni più ridotte, rendendolo veramente tascabile: era il MicroTAC, che aveva uno sportellino per coprire i tasti e un display da sole due righe. Da allora c'è stata una continua rincorsa a produrre telefonini sempre più piccoli. Si pensava che la vera sfida del futuro fosse creare dispositivi minuscoli, quasi invisibili.

All'inizio degli anni novanta non esisteva il concetto di smartphone, internet non era diffuso, i display erano molto limitati e il touchscreen era qualcosa di futuristico.

All'inizio degli anni Duemila, il mercato dei cellulari subì una svolta: uscì il Nokia 3310, uno dei telefoni portatili più venduti di sempre, che entrò nelle tasche degli adulti ma, per la prima volta, anche in quelle degli adolescenti.

Nel 2002, negli Stati Uniti, BlackBerry diffuse su scala internazionale un telefono cellulare in grado di connettersi facilmente a internet e ricevere e-mail. Quello fu l'inizio dell'inversione di tendenza: il display rubò sempre più spazio ai tasti, fino a sostituirli completamente nel 2007, quando Apple di Steve Jobs presentò il primo iPhone. Da allora in poi il telefonino fu utilizzato soprattutto per navigare su internet, sui social network, per mandare e-mail e per usare applicazioni sempre più complesse.

In questo modo, fu abbandonata l'idea di produrre cellulari sempre più piccoli e si iniziarono a realizzare schermi sempre più grandi, arrivando alla nascita del tablet e del phablet, un ibrido fra smartphone e tablet. Nel corso del tempo, quindi, il telefono cellulare ha avuto una notevole evoluzione, portandolo a diventare un 'cellulare intelligente', il cosiddetto smartphone.

In realtà, la storia di questi dispositivi ha inizio già nel 1992, quando IBM produce il primo smartphone in assoluto, chiamato Simon, con la presenza di calendario, rubrica, orologio, block notes, e-mail, giochi e la novità del pennino per scrivere sullo schermo. Tuttavia, la definitiva affermazione dello smartphone si ha con l'avvento dell'Iphone che riesce a cambiare il modo in cui si percepisce un telefono cellulare. Il cambiamento coinvolge anche la navigazione su Internet, per la quale nel 2013 si è registrato il sorpasso degli utenti Internet via mobile su quelli da postazione fissa, e il rilascio di contenuti tramite le applicazioni. Inoltre, l'Iphone porta con sé lo stigma dell'isolamento tipico di Apple, nascendo con il sistema operativo proprietario iOS.

Inizia, cosi, la rincorsa a questo nuovo modello di business anche con Google che sviluppò e riuscì a distribuire sui terminali di produttori leader il suo sistema operativo Android.

A sostegno dell'importanza della trasformazione del telefono cellulare in smartphone, vi è anche l'analisi della società specializzata Flurry, che sostiene:

> *Il tasso di adozione dei devices iOS e Android è stato superiore a quello di ogni altra tecnologia consumer della storia. Comparato con altre recenti tecnologie, l'adozione di 'smart devices' è stata dieci volte più rapida di quella della rivoluzione dei PC negli anni Ottanta, due volte più veloce del boom di Internet negli anni Novanta e tre volte di quella recente dei social network.*

WEARABLE TECHNOLOGIES Il 2013 può essere considerato come l'anno di nascita commerciale delle wearable technologies.

Come suggerisce il termine stesso, per tecnologia indossabile s'intende tutto ciò che si può attaccare, appendere o allacciare al corpo e che in diversa mi-

sura interagisce con il corpo stesso o con quello che si sta facendo.
Si potrebbe affermare che gli apripista di questa idea sono stati i Google Glass che permettono a chi li indossa di filmare, richiamare informazioni sull'ambiente circostante e molto altro. Le applicazioni più semplici e immediate sono quelle legate al fitness, quindi bande elastiche, bracciali e orologi che se indossati rilevano diversi dati del corpo e dei suoi movimenti diventando utilissimi partner per gli allenamenti e le gare. Un ulteriore esempio di tecnologia indossabile relativa al benessere è il braccialetto che rileva la quantità di raggi UV ricevuta e consiglia il modo migliore per proteggersi.
Questi sono solo alcuni esempi di queste nuove tecnologie che, lentamente, si stanno diffondendo facendosi spazio nelle nostre vite.

1.3 SISTEMI OPERATIVI

Un sistema operativo mobile è un sistema operativo che controlla un dispositivo mobile con lo stesso principio con cui Mac OS, Unix, Linux o Windows controllano un desktop computer oppure un laptop. Tuttavia, deve affrontare problematiche diverse legate alla natura del dispositivo mobile, tra cui: limitatezza delle risorse, differenti tecnologie per l'accesso a Internet, nuovi metodi d'immissione e dimensioni del display.

1.3.1 Storia dei sistemi operativi mobile

La diffusione dei sistemi operativi mobile avviene in parallelo a quella dei devices.
I primi ad essere rilasciati sono Symbian OS di Symbian Foundation e quelli di Micosoft: Pocket PC 2000, Windows Mobile 2003 e Windows Mobile 5. Successivamente, come già detto, nel 2007 si ha il grande cambiamento, attraverso il quale le applicazioni diventano il cuore del telefono e Apple diffonde il paradigma de-

gli Application Store per dispositivi mobile. Nascono, cosi, le prime versioni iOS di Apple e Android di Google, mentre Microsoft rilascia il suo Windows Mobile 6. Da quel momento in poi, vengono realizzate innumerevoli versione di questi tre principali sistemi mobile fino ad arrivare a quelle attuali di iOS 8, Android Lollipop e Windows Phone 8.

1.3.2 iOS

iOS è un sistema operativo sviluppato da Apple per Iphone, Ipad e Ipod Touch. Il sistema è una derivazione di UNIX [1] e usa un microkernel XNU Mach [2] basato su Darwin.

1.3.3 Android

Android è un sistema operativo Open Source [3] per dispositivi mobili sviluppato da Google Inc. sulla base del kernel Linux. Android è stato progettato principalmente per smartphone e tablet, con interfacce utente specializzate per televisori (Android TV), automobili (Android Auto), orologi da polso (Android Wear), occhiali (Google Glass), e altri.
Una delle peculiarità del sistema Android, che contribuisce al suo dominio nel mercato Mobile, è il fatto di essere adottato da moltissimi dispositivi, sia di fascia alta che bassa, come HTC, Samsung, LG, Sony e Huawei.
Il sistema è caratterizzato da librerie e API scritte in C (o C++) e utilizza la Dalvik virtual machine [4] con un compilatore just-in-time per l'esecuzione di Dalvik dex-code (Dalvik Executable), che viene tradotto da bytecode Java. Le applicazioni sono Java-based; questo significa che il codice dell'applicazione C/C++ deve esse-

1 Unix è un sistema operativo per computer inizialmente sviluppato da un gruppo di ricerca dei laboratori ATT e Bell Laboratories, nel quale figurarono anche Ken Thompson e Dennis Ritchie.
2 XNU è il kernel utilizzato nel sistema operativo open source Darwin che Apple Inc. usa come base per il suo sistema operativo Mac OS X.
3 Software pubblicato con una licenza che permette a chiunque di utilizzarlo e che ne incoraggia lo studio, le modifiche e la redistribuzione.
4 Dalvik è una macchina virtuale progettata da Dan Bornstein (dipendente Google) e ottimizzata per sfruttare la poca memoria presente nei dispositivi mobili.

re inserito all'interno di un progetto Java, il quale produce alla fine un pacchetto Android (APK).

1.3.4 Windows Phone

Windows Phone è il sistema operativo mobile di Microsoft, presentato per la prima volta al Mobile World Congress il 15 febbraio 2010. Al contrario del suo predecessore Windows Mobile, indirizzato al mercato enterprise, si rivolge al mercato consumer e si presenta radicalmente ridisegnato nella struttura e nell'interfaccia: quest'ultima, denominata Modern UI o linguaggio di design Microsoft, ha debuttato su Windows Phone per poi estendersi a tutto l'ecosistema Microsoft, comprendendo il sistema operativo desktop Windows 8 e le dashboard di Xbox 360 e One.
Il problema principale per gli utenti di Windows Phone è la presenza di un minore numero di applicazioni presenti nel suo Store rispetto ai principali concorrenti, dovuta al tardo arrivo del sistema sul mercato in confronto a quest'ultimi; divario che si è assottigliato nel corso del 2013 e del 2014 con l'arrivo di quasi tutte le principali applicazioni del mondo mobile.
Windows Phone si trova principalmente su smartphone Nokia, grazie alla partnership stabilita tra l'azienda finlandese e Microsoft.

1.3.5 Confronto

Il mercato degli smartphone ha raggiunto un importante traguardo nel 2013, quando, per la prima volta, sono state effettuate più di un miliardo di spedizioni in tutto il mondo. Secondo International Data Corporation (IDC) Worldwide Quarterly Mobile Phone Tracker, i sistemi Android e iOS dominano il mercato rispetto a Windows Phone, Blackberry e tutti gli altri sistemi mobile presenti.

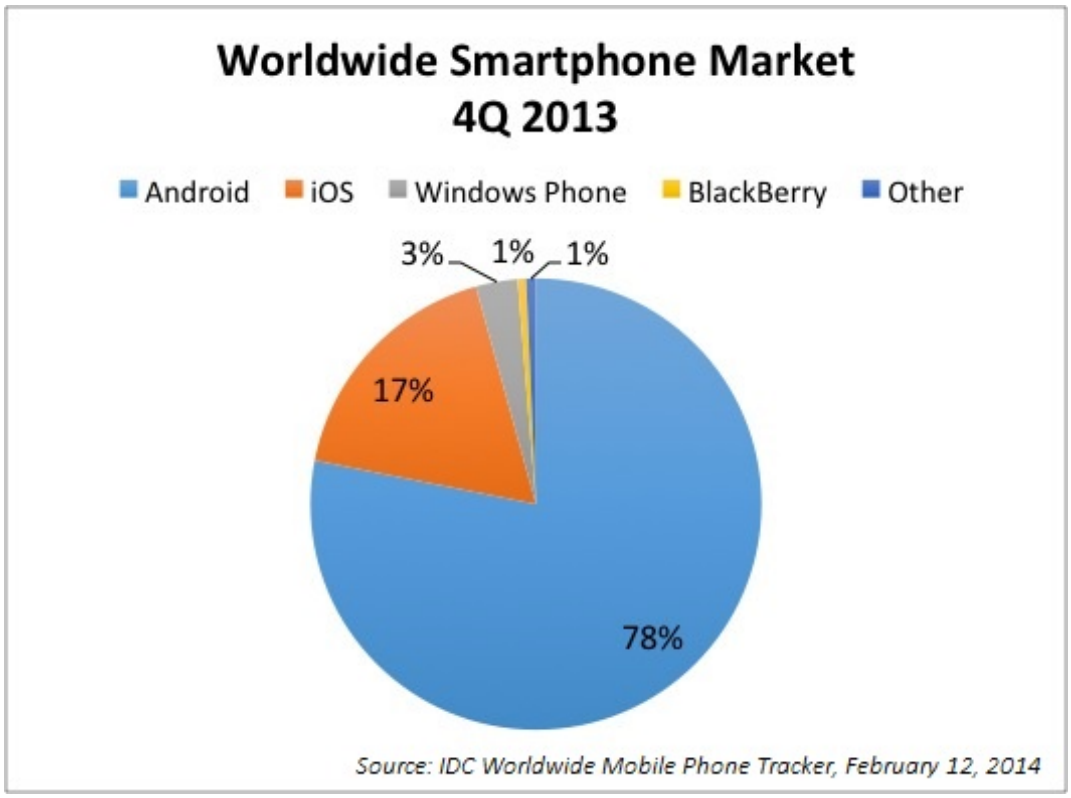

Figura 1: Confronto diffusione sistemi operativi mobile (quarto trimestre 2013)

Di seguito viene effettuato un confronto in base agli aspetti più importanti dei tre principali sistemi operativi mobile: iOS, Android e Windows Phone.

STORE E APPLICAZIONI Per quel che riguarda qualità e quantità di applicazioni nello store, Windows Phone rimane più indietro rispetto ai concorrenti.
I seguenti numeri, seppur indicativi, mostrano le evidenti differenze:

- Android, 1.3 milioni di app;
- iOS, 1.2 milioni di app;
- Windows Phone, 300,000 app;

Tradizionalmente, iOS è una piattaforma più redditizia per gli sviluppatori e, per questo motivo, fino ad oggi vi è stata la tendenza di dare una maggiore priorità a questo sistema nel rilascio di nuove applicazioni. Tuttavia, Android sta continuando a crescere, sotto tutti i punti di vista, e quindi questa abitudine potrebbe anche cambiare.

In fine, un punto a favore del sistema operativo di Google è dato dall'elevato numero di applicazioni gratuite rispetto alla concorrenza.

SICUREZZA Partendo dalla considerazione che nessun sistema è completamente sicuro, i tre colossi delle principali piattaforme mobile, nel corso del tempo, hanno cercato di apportare correttivi in tal senso.

In particolare, Apple ha introdotto il Touch ID, ovvero il lettore di impronte digitali presente nel tasto home dell'iPhone. Quest'ultimo viene utilizzato per sbloccare il dispositivo semplicemente appoggiando il dito sul tasto home e per acquisti in App Store e iTunes Store.

Per quel che riguarda Android, in termini di sicurezza, Google gode anche del supporto delle specifiche aziende, come Samsung, che lavorano al fine di migliorare quest'aspetto sui propri dispositivi.

INTERFACCIA iOs offre un'interfaccia luminosa e moderna e, come i suoi concorrenti, fornisce un apposito pannello a scorrimento dall'alto vero il basso per le notifiche e le impostazioni più rapide.

Per quel che riguarda Android, i grandi marchi hanno aggiunto delle loro personalizzazioni sopra quella che è l'interfaccia standard del sistema. Inoltre, Google con la recente versione, Android Lollipop, ha introdotto un notevole cambiamento a livello di interfaccia tramite il 'Material Design'. Si tratta di un nuovo stile caratterizzato da un ambiente 3D, dove gli elementi vari dell'interfaccia si sovrappongono creando un interessante effetto luci e ombre. Altre peculiarità della nuova interfaccia sono i colori vivaci, le icone minimal e una particolare attenzione al contrasto del font per il raggiungimento della massima leggibilità.

In fine, l'interfaccia di Windows Phone si basa principalmente su una griglia di 'Live Titles' che mostrano contenuti personalizzabili dall'utente.

RISPARMIO BATTERIA La batteria rimane ancora un punto critico nei devices di ultima generazione a causa dell'eccessivo consumo. Si tratta di un aspetto legato, soprattutto, al singolo dispositivo. Tuttavia, per minimizzare il problema, le varie piattaforme mobile hanno cercato di apportare modifiche al

fine di prolungare, quanto possibile, l'utilizzo del device.

In questo senso, un ulteriore passo in avanti effettuato con Android Lollipop è l'integrazione della funzionalità di risparmio batteria consentendo la disattivazione automatica di applicazioni non essenziali.

Anche Windows Phone dispone di un opzione per il risparmio con la disattivazione di applicazioni non essenziali. A differenza di Android e Windows Phone, iOS non offre la funzionalità di risparmio batteria, ma solo della statistiche dettagliate sul suo utilizzo.

CONNETTIVITÀ Tutte le piattaforme mobile supportano Bluetooth e WiFi.

Android e Windows Phone hanno introdotto la novità del NFC per trasferimenti wireless e pagamenti mobile, mentre Apple ha incluso questa tecnologia all'interno dei recenti iPhone 6 e iPhone 6 Plus, aggiungendo anche Apple Pay [5].

CHIAMATE E MESSAGGISTICA Relativamente a chiamate e messaggistica, Apple offre dei servizi proprietari come FaceTime per le videochiamate e iMessage per lo scambio di messaggi.

Google ha introdotto un interessante servizio chiamato Hangouts che permette l'invio di messaggi tramite Wi-Fi, rete dati o SMS e la possibilità di effettuare videochiamate.

Ai servizi di messaggistica dei concorrenti, Microsoft ha scelto Skype, anche se si tratta di un servizio separato dagli SMS standard e, inoltre, installabile su qualsiasi piattaforma.

CLOUD Sempre più utilizzato è il servizio Cloud che consente il backup dei propri dati, ad esempio contatti e immagini, su un proprio spazio di archiviazione online.

A tal proposito Apple offre un proprio servizio, chiamato iCloud, mentre Android può disporre del servizio cloud di Google, chiamato Google Drive, e anche Microsoft dispone di un proprio servizio, ovvero One Drive.

5 Sistema di pagamento contactless in grado di funzionare con i maggiori circuiti di pagamento.

1.4 MOBILE BUSINESS PROCESS REENGINEERING

L'utilizzo e l'importanza della tecnologia nelle imprese è cambiata radicalmente negli ultimi decenni.
L'utilizzo dell'informatica a livello operativo è stato sostituito da soluzioni industriali e sistemi ERP complessi che consentono il supporto a tutta la catena di fornitura.
Un problema critico si verifica se processi speciali non possono essere supportati interamente da stazioni fisse, ma necessitano di mobilità.
La massima espressione di mobilità è data da dispositivi come smartphone e tablet, in quanto possono essere a portata di mano della persona in qualsiasi momento e luogo. Questo suggerisce alle organizzazioni la possibilità di utilizzare questi device in contatto col cliente, permettendo quindi di mostrare dei propri prodotti o servizi oppure di svolgere subito determinate attività con eventuali riduzioni delle tempistiche e migliorie nella qualità del servizio.
A questo punto, un'altra domanda che ci si pone è come sfruttare le nuove potenzialità del mobile e in alcuni casi la risposta potrebbe essere realizzare una versione mobile di un sito Web che l'organizzazione già dispone, mentre, in altri contesti, questo non può essere sufficiente o soddisfacente e quindi si va a sfruttare il nuovo paradigma del mobile, ovvero quello dell'app. Un'applicazione mobile fornisce un forte collegamento con la propria organizzazione, grazie al fatto che interfaccia e dati statici sono sempre disponibili sul dispositivo riducendo cosi la quantità di dati che deve essere trasferita dalla rete al device. Inoltre, altra peculiarità dell'app è che può fornire l'accesso ad alcune funzionalità anche senza una connessione alla rete disponibile. A tal proposito potrebbe essere considerato uno scenario in cui sia necessario salvare informazioni con urgenza ed effettuare, poi, un'eventuale transazione con la disponibilità della rete.
Uno degli aspetti più interessanti relativo al Mobile è quindi l'impatto che ha, quest'ultimo, sui processi di Business.
I processi sono progettati in base alla tecnologia disponibile attualmente. Quindi, quando la tecnologia cambia drasticamente vengono abilitati nuovi modelli.

Le aziende stanno prestando notevole attenzione al mobile sia per i vantaggi a cui può portare che per necessità di competizione.
Per poter utilizzare le nuove potenzialità relative a soluzioni mobile è necessario la reingegnerizzazione dei processi di business (BPR, Business Process Reengineering).
Il business process reengineering (BPR) ha ricevuto notevole attenzione nelle imprese e nel mondo accademico. Per anni, le organizzazioni hanno investito una quantità significativa di denaro nell'informatica, senza però ottenere aumenti significativi di produttività. Dopo una serie di attente analisi è emerso che una delle cause di questo problema era data dal fatto che le organizzazioni stavano utilizzando le nuove tecnologie solo per automatizzare i processi esistenti. Questo approccio ha prodotto miglioramenti, ma era incrementale piuttosto che rivoluzionario. Cosi le varie aziende hanno iniziato a ripensare interi processi per sfruttare tutto il potenziale offerto dalla tecnologia: in questo modo è nato il BPR.
Tuttavia, viene sempre tenuta in considerazione anche la semplice automazione, specialmente per piccole aziende in cui viene richiesta la riduzione dei costi, la semplificazione dei processi o la necessità di piattaforme portatili.

1.5 BPR

Il BPR è la completa revisione dei processi aziendali al fine di massimizzare il valore derivante dalle singole componenti, attraverso interventi sia di riduzione dei costi sia di massimizzazione dell'efficacia.
Il BPR è caratterizzato da un insieme di passi e attività, supportati da opportune metodologie e tecniche, e costituisce un intervento di rimodellazione dei processi primari e critici (legati al "core business") di un'organizzazione, guidato dal valore per il cliente e volto ad eliminare attività e flussi che non ne generano.
Il ridisegno dei processi può focalizzarsi su tre tipi di cambiamento:

STREAMLINING Si tratta di un approccio incrementale sui processi con l'obiettivo di introdurre delle migliorie. Può consistere, ad esempio, nella modifica delle sequenze, nella semplificazione di attività, nell'automatizzazione delle attività, nella ridefinizione di input e output o nel bilanciamento dei carichi di lavoro.

BUSINESS PROCESS REENGINEERING Si tratta di un approccio radicale sui processi attuali. Può consistere, ad esempio, nell'eliminazione di un processo, nella combinazione di più processi, nell'introduzione di nuove competenze o nell'implementazione di tecnologie avanzate che impattano sull'intero processo.

ENTERPRISE TRANSFORMATION Si tratta di un approccio radicale sul business. Può consistere, ad esempio, nello sviluppo di alleanze strategiche o nel ribilanciamento del portafoglio di business.

L'obiettivo del BPR è, quindi, quello di individuare un possibile cambiamento che possa portare a delle migliorie: questo miglioramento può essere espresso in termini di costi, qualità, servizio e tempi di ciclo. E' necessario individuare quello che viene chiamato 'Breakthrough', ovvero il punto di rottura sul quale intervenire per ottenere dei benefici.

Il breakthrough può essere individuato in due modi: attraverso il dialogo col cliente o con il benchmarking. La prima tecnica consiste nel capire quali sono le aspettative del cliente, mentre con la seconda avviene un confronto delle performance interne con standard esterni di eccellenza.

Dopo aver individuato il punto in cui è necessario un intervento, quello che si fa è analizzare la situazione attuale ed effettuare una rivisitazione del processo o dei processi correnti.

A questo seguiranno una fase di progettazione dei nuovi processi e una successiva implementazione della soluzione integrando le nuove tecnologie che offre il mercato in base a quelle che sono le necessità.

In fine, sarà necessaria una verifica per riuscire a capire se il cambiamento ha portato a migliorie in termini di riduzione dei costi, aumento del profitto o della qualità

del servizio.

Il ciclo di vita del business process reengineering può essere riassunto con la seguente figura:

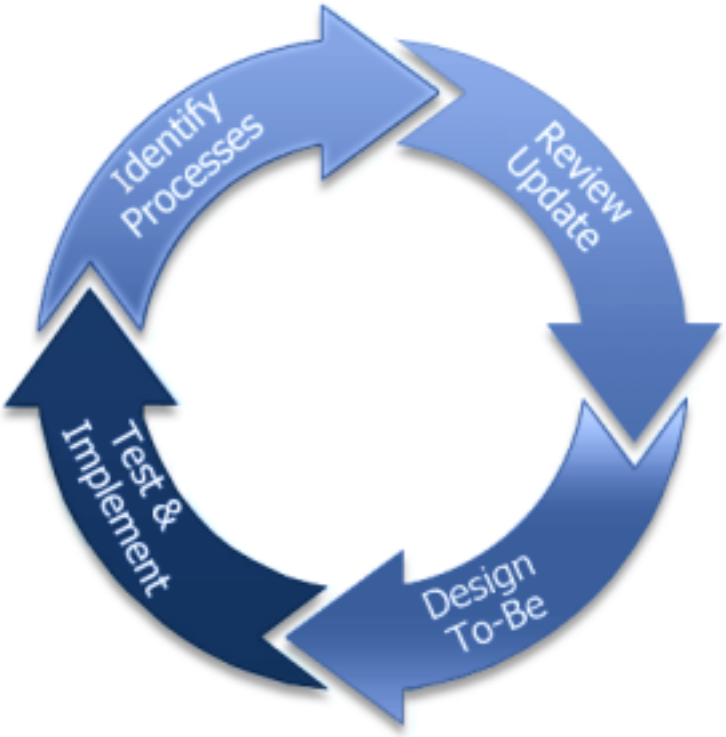

Figura 2: Ciclo di vita BPR

Più in particolare, applicare tale metodologia significa:

- cancellare invece di automatizzare; vengono eliminate le duplicazioni e le attività poco rilevanti;
- accorpare più attività in un'unica posizione organizzativa; l'accorpamento delle mansioni consentito dalla tecnologie e dalla crescita del personale, permette di attribuire mansioni più complete;
- ridisegnare il processo oltre i confini dell'impresa, coinvolgendo clienti e fornitori in un'ottica di partnership;
- misurare i risultati e ricominciare; il cambiamento deve diventare una condizione naturale dell'organizzazione; si deve adottare una concezione dinamica che consenta un adattamento continuo dei processi al mercato.

I vantaggi derivanti da questo insieme di attività sono i seguenti:

- semplificazione del lavoro;
- riduzione della burocrazia;
- ridefinizione ed ampliamento di ruoli e mansioni;
- eliminazione dei colli di bottiglia.

Per Business Process Reengineering si intende, quindi, un radicale intervento di ristrutturazione organizzativa, volto a ridefinire i processi aziendali, facendo leva sull'analisi del valore delle attività che li costituiscono. In questo modo è possibile misurare il reale valore che le attività (e quindi i processi) aggiungono all'organizzazione in termini di produttività.

2 PROGETTAZIONE

2.1 DEFINIZIONE E OBIETTIVI DEL PROGETTO

Il progetto nasce, principalmente, da una rivisitazione dei processi di presa dell'ordine da parte degli agenti di commercio di molteplici ditte del territorio nazionale. Il primo obiettivo che ci si pone è quindi quello di ottimizzare questo tipo di servizio in termini di:

- riduzione dei costi;
- miglioramento della qualità del servizio;
- aumento della produttività.

Come obiettivo più ambizioso, invece, vi è quello di riuscire a realizzare una sorta di 'gestionale portatile', quindi sempre a portata di mano dell'agente, che consenta di accedere rapidamente ad un enorme mole di dati in tempo reale.
Il progetto si rivolge principalmente alla figura dell'agente, ma ne consente l'utilizzo anche da parte di specifiche persone autorizzate all'interno di una specifica ditta, come l'imprenditore.

2.2 DESCRIZIONE DEL DOMINIO

Per comprendere al meglio le origini e gli obiettivi del progetto si effettua un'attenta analisi della situazione attuale.
Aziende del territorio nazionale, che si occupano della vendita di determinate categorie di prodotti, si affidano a molteplici agenti per l'acquisizione di nuovi clienti e la vendita della loro merce. Gli agenti, quindi, sono degli intermediari commercia-

li che hanno il compito di promuovere la distribuzione e la vendita dei prodotti e dei servizi delle aziende, ricercando la clientela e concludendo contratti per conto della casa mandante nelle zone loro assegnate; ogni agente può operare per più imprese.
La casa mandante ha il dovere di mettere a disposizione dell'agente la documentazione riguardante i beni o servizi trattati e tutte le altre informazioni necessarie all'esecuzione del contratto.
Quindi, i principali compiti dell'agente sono due:

- ricerca e mantenimento della clientela;
- raccolta di ordini.

Oltre a questo, possono essere previste a carico dell'agente delle obbligazioni strumentali:

- assistenza post-vendita per i beni commercializzati nella propria zona;
- organizzazione di campagne pubblicitarie e promozionali;
- partecipazione a fiere e ad eventi congressuali;
- deposito di beni, sia per la cessione sia come campionario;
- trasporto dei beni ai clienti.

I prodotti e servizi che formano oggetto del mandato conferito all'agente devono essere descritti con attenzione e precisione, attraverso uno specifico allegato in cui sono individuati in dettaglio con i relativi prezzi di listino.
Il mandante ha il diritto di ottenere informazioni dall'agente sullo stato delle vendite, sulle condizioni del mercato, sulla clientela e molto altro. In tal senso, l'agente può organizzarsi istituendo uno schedario della clientela nel quale riportare l'anagrafe dei clienti attuali e potenziali ed evidenziando, per ciascuno, dimensioni e caratteristiche aziendali, fatturati e vendite per periodi, condizioni di pagamento e regolarità di riscossione, modalità di contrattazione e molto altro.

L'imprenditore della ditta, oltre ad avere ampio accesso a tutte queste informazioni, può contattare nuovi clienti e raccogliere ordini, cosi come opera l'agente di commercio. Gli ordini, raccolti dall'agente, non hanno efficacia vincolante per il preponente, il quale si riserva il diritto d'accettare o respingere le proposte inviategli dall'agente. Questa considerazione è strettamente legata agli innumerevoli fattori esterni e imprevedibili che condizionano le forniture.
Nella creazione dell'ordine è necessario definire molteplici informazioni, tra cui la più importante è quella del prezzo che può essere individuato tramite listini oppure, con appropriata autorizzazione, può essere definito dall'agente stesso.
La stessa considerazione si effettua nel caso degli sconti, i quali possono essere predefiniti per certi clienti o prodotti, oppure, sempre con l'autorizzazione del mandante, possono essere stabiliti dall'agente.

2.3 BPR

2.3.1 Analisi del processo di business

Nel contesto sopra descritto, riguardo al processo di raccolta degli ordini, si effettuano le seguenti osservazioni:

- il processo di raccolta degli ordini richiede agli agenti la necessità di recarsi dal cliente, mostrare la documentazione relativa a prodotti e servizi offerti dal mandante e raccogliere dati del cliente e dell'eventuale ordine richiesto;
- alla fine della propria giornata lavorativa, l'agente si reca all'ufficio appropriato della ditta dove consegna i vari moduli con gli ordini raccolti e, la persona incaricata si occuperà di inserire i dati nel gestionale dell'azienda.

All'interno di questo processo è presente una ripetizione di attività: la raccolta delle informazioni riguardante l'ordine viene effettuata sia dall'agente, trascrivendo queste in appositi moduli, sia dall'impiegato dell'ufficio aziendale che deve riportare i dati, inseriti nei moduli, nel software gestionale.
Inoltre, per consentire all'agente di promuovere prodotti e servizi dell'azienda e di

raccogliere le ordinazioni dei clienti, è necessario fornirgli una quantitativo enorme di documenti e moduli contenenti, appunto, informazioni su azienda mandante, prodotti, listini, sconti, clienti e tutto ciò che serve al rappresentante per eseguire le proprie attività.
Nasce, cosi, la richiesta di ottimizzazione dell'intero processo.

2.3.2 Riprogettazione del processo di business

L'analisi relativa alla raccolta dell'ordine suggerisce una rivisitazione del processo.
In particolare, constatando la presenza della ripetizione di attività, si rende necessaria l'eliminazione della seconda attività svolta: le informazioni relative all'ordine ma anche quelle sui nuovi clienti devono essere raccolte una sola volta. Quindi, si va ad eliminare il passaggio di documenti all'ufficio perché è solamente l'agente che raccoglie dati e si preoccupa di inserirli nel gestionale.

2.3.3 Benefici ottenuti

In seguito alla re-ingegnerizzazione, i benefici ottenuti sono notevoli:

- riduzione dei costi legati alla gestione dell'ufficio per l'inserimento degli ordini e all'utilizzo di materiale cartaceo;

- minimizzazione della possibilità di errori nella raccolta dei dati relativi a clienti e ordini;

- riduzione dei tempi nella raccolta degli ordini con un eventuale aumento della produttività.

2.4 CASI D'USO

Dopo aver capito qual'è il principale cambiamento che si vuole ottenere, si riassumono i requisiti attraverso il diagramma dei casi d'uso che facilita la comprensione delle principali funzionalità del sistema e degli attori coinvolti.

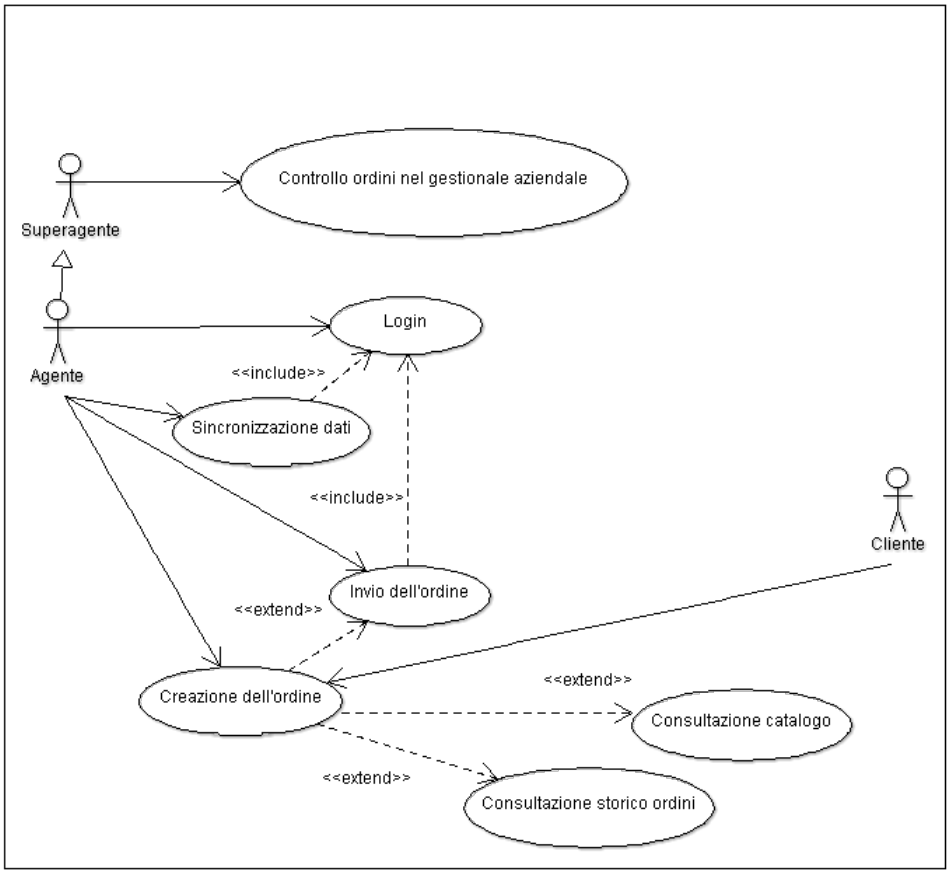

Figura 3: Diagramma dei casi d'uso - raccolta degli ordini

Come si può vedere dal diagramma, i principali attori individuati sono raggruppati in agente e superagente. Il primo può raccogliere ordini accedendo alle informazioni per cui dispone l'autorizzazione, mentre il secondo è sempre un agente ma senza alcune limitazione e quindi può accedere a qualsiasi informazione e verificare gli ordini salvati nei gestionali. Quindi, un superagente può essere con-

siderato l'imprenditore o un responsabile aziendale.

2.5 ARCHITETTURA DEL SISTEMA

In seguito alle considerazioni precedenti, è necessario capire come si può fare ciò che è stato richiesto.
In questo tipo di scenario, il primo fattore da mettere in evidenza è quello della portabilità.
La priorità dell'agente è quella di spostarsi da paese a paese, avendo sempre sottomano i dati necessari per svolgere il proprio lavoro. In questo senso, ci vengono incontro le nuove tecnologie mobile: l'idea è quella di dotare l'agente di un device, come il tablet, dove all'interno ci sia un'apposita applicazione installata.
L'applicazione deve presentare le seguenti funzionalità per raggiungere gli obiettivi prefissati:

- sincronizzazione dei dati, la rivisitazione del processo di raccolta degli ordini ha suggerito l'eliminazione di tutto il materiale cartaceo e quindi diventa indispensabile la possibilità di ottenere dati attuali, come quelli relativi a clienti e prodotti, sul proprio device;
- calcolo di listini e sconti, al fine di velocizzare e semplificare l'attività dell'agente risultano necessarie delle funzionalità di calcolo per la determinazione di parametri fondamentali per l'ordine;
- salvataggio degli ordini, fondamentale è la procedura di presa di un ordine che ha l'obiettivo di minimizzare gli errori nell'inserimento dei dati e quello di mantenere all'interno del dispositivo tutti gli ordini presi nel corso del tempo;
- invio degli ordini, si tratta della funzionalità che consente di concludere il processo della raccolta degli ordini in quanto permette il loro salvataggio nel gestionale aziendale.

Analizzando le funzionalità di base che l'applicazione deve avere, si può subito intuire la necessità di un collegamento con i diversi sistemi informativi aziendali. Da questa considerazione, si rende necessaria la progettazione di una parte server che permetta la comunicazione tra l'applicativo client e i sistemi aziendali; infatti, non è possibile una comunicazione diretta tra il client e il gestionale aziendale data la diversa natura di quest'ultimo che può variare da azienda ad azienda. Cosi la soluzione è quella di utilizzare una sorta di interfaccia tra l'applicazione installata sullo smart device e il gestionale dell'azienda.
A tal fine l'architettura indicata è la seguente:

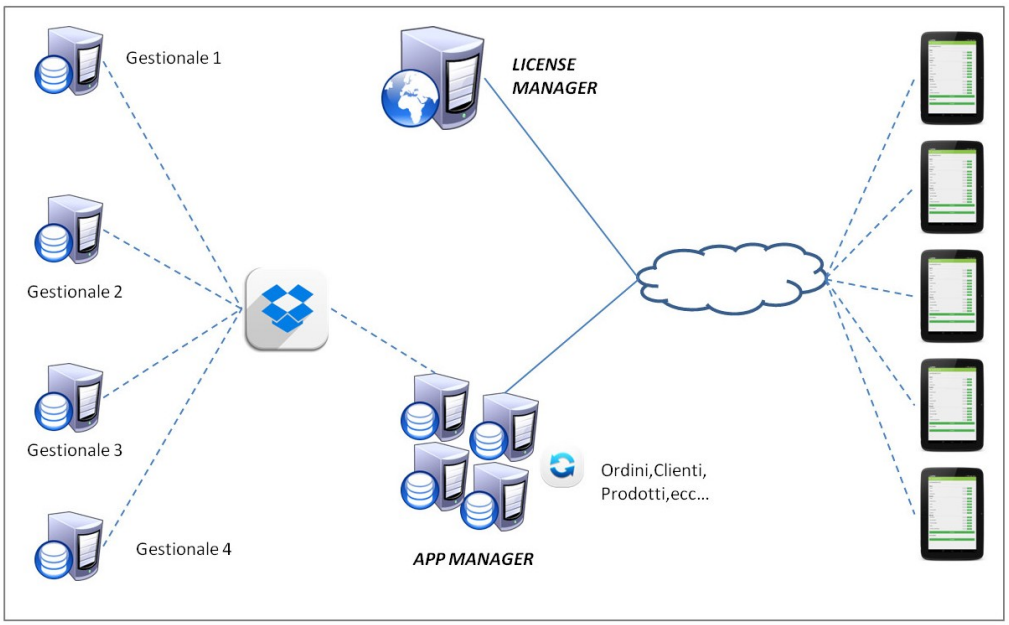

Figura 4: Architettura del sistema

Come è possibile vedere dalla figura, l'architettura può essere vista come la combinazione di due componenti: una client-server e una cloud based.
La prima componente è costituita dal collegamento tra gli smart devices (lato client) e il server. Il lato server della struttura è costituito dal license manager per la gestione degli accessi e dall'app manager per la gestione dei dati veri e propri.
La seconda componente è data dal servizio cloud di Dropbox che contiene i cosiddetti tracciati, ovvero dei files organizzati in apposite directory dell'azienda,

utilizzati per lo scambio di dati tra gestionali e app manager. L'utilizzo di quest'ultimi risulta fondamentale per consentire l'interscambio di dati tra app manager e gestionali. Dopo l'esportazione o l'importazione di dati tra gestionali e app manager, trascorso un certo intervallo di tempo, i tracciati vengono cancellati. Questo avviene perché i dati sono già stati archiviati su app manager in caso di esportazione o sul gestionale in caso di importazione e quindi non è necessario conservare ulteriori informazioni all'interno della propria directory Dropbox, con conseguente risparmio di spazio di archiviazione sul servizio cloud.
I dati rimangono, invece, disponibili sull'app manager, al quale si collegano gli smart devices per il download e l'upload di informazioni.

2.6 CLIENT

La parte client del sistema è rappresentata dall'applicazione installata sullo smart device.
L'obiettivo è quello di realizzare un'applicazione standard interfacciabile con qualsiasi tipo di gestionale, consentendo, poi, la possibilità di definire delle personalizzazioni con aggiunte di moduli e modifiche grafiche su misura.

2.6.1 Modalità di accesso

Come già detto, gli utenti dell'applicazione possono essere suddivisi in due macro-figure:

AGENTE si riferisce all'agente o venditore che ha il compito, per conto di una o più aziende, di ricercare nuovi clienti e raccogliere ordini per i propri prodotti con la possibilità di accedere solo ad una ristretta quantità di informazioni;

SUPERAGENTE si riferisce al titolare di un'azienda o ad una figura incaricata da quest'ultimo, avente il permesso di accedere a qualsiasi informazioni e modulo dell'app.

Una delle necessità della figura dell'agente è quella di lavorare per più aziende. A tal fine, si fornisce una gestione multi-progetto, in modo che l'agente possa selezionare la ditta per la quale, in quel momento, deve operare.
Concludendo, si rende, quindi, necessaria la definizione di diverse modalità d'accesso all'applicazione che saranno stabilite da apposite configurazioni dettate dalle aziende stesse.

2.6.2 Modalità di utilizzo

Una delle peculiarità dell'applicativo è la possibilità dell'utilizzo in modalità online e offline.
Mentre con la modalità online si ha la possibilità di sfruttare a pieno le funzionalità dell'applicazione, per quella offline è necessario un ulteriore approfondimento. Quest'ultima nasce dal modo di agire dell'agente, il quale spostandosi molto e incontrando clienti in svariati luoghi può avere la necessità di svolgere il proprio lavoro anche in situazioni di totale assenza della connettività.
Ad esempio, si consideri un possibile scenario nel quale l'agente si trova col proprio dispositivo ad una fiera e, a causa dell'eccessivo numero di utenti connessi, nel momento di raccolta dell'ordine per un cliente, non riesce a stabilire una connessione. Prevedendo solo una modalità online, l'agente non sarebbe in grado di prendere l'ordine, mentre con quella offline si permette di salvare gli ordini sul proprio smart device e inviargli al gestionale dell'azienda quando sarà possibile stabilire una connessione di rete.
In questo modo, l'agente o l'utente utilizzatore dell'applicazione può operare con continuità senza perdere delle ordinazioni e dei potenziali clienti.
L'unico requisito richiesto da una modalità offline è che ci siano dei dati locali, in particolare quelli riguardanti clienti e prodotti, sui quali effettuare le ordinazioni: altrimenti, con la totale assenza di informazioni sul device, non sarebbe possibile raccogliere nuovi ordini.

2.6.3 Gestione dati

Il cuore dell'applicazione è dato dal database locale; la gestione dei dati aziendali e quella degli ordini non può avvenire se non vi è un contenitore di dati che rispetti le proprietà ACID:

ATOMICITÀ le transazioni che avvengono sul database devono essere indivisibili, ovvero le loro esecuzioni devono essere totali o nulle e non sono ammesse esecuzioni parziali;

CONSISTENZA eventuali vincoli di integrità devono essere sempre rispettati, in modo che non vi siano inconsistenze nel DB;

ISOLAMENTO eventuali fallimenti di transazioni effettuate sul database non devono influire sull'esecuzione di altre;

DURABILITÀ O PERSISTENZA una volta che la transazione è stata completata, i cambiamenti apportati non dovranno essere persi.

Lo schema concettuale definito tramite il diagramma delle classi UML è il seguente:

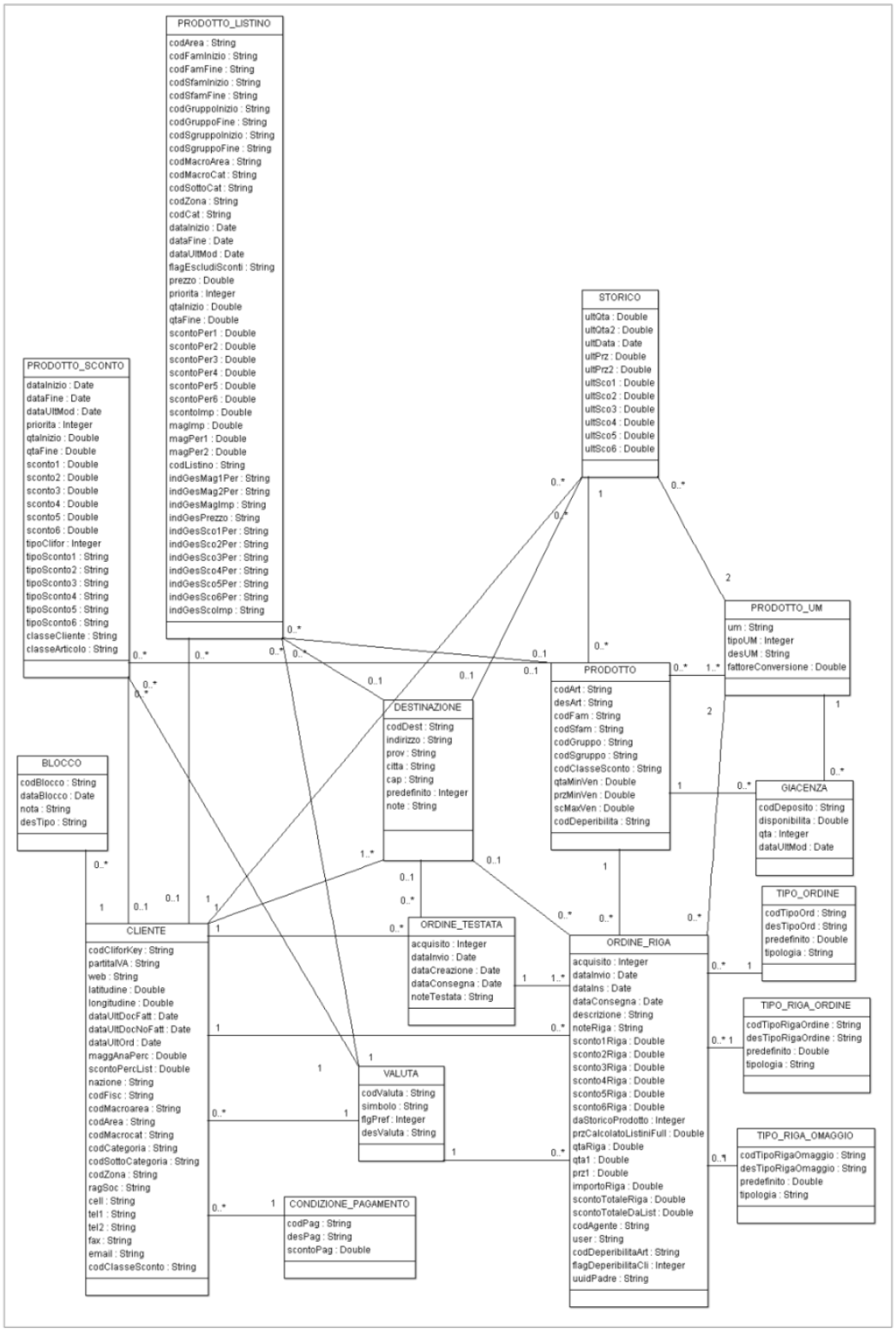

Figura 5: Diagramma delle classi

Utilizzando il linguaggio di modellazione UML (Unified Modeling Language) è possibile comprendere al meglio il dominio di interesse.
Le classi modellate, come mostrato nel diagramma, sono:

CLIENTE La classe Cliente modella semplicemente un generico individuo per il quale l'agente può raccogliere degli ordini.
Oltre alle classiche informazioni di anagrafica e di contatto, può disporre di eventuali maggiorazioni e sconti sui prezzi.
Può essere associato con uno o più blocchi, dove per blocco si intende un divieto di presa dell'ordine per il cliente specifico a causa di vari motivi, come il mancato pagamento di ordini passati.
In fine, per ogni cliente possono essere definite una o più destinazioni, che verranno prese poi in considerazione in fase di raccolta dell'ordine.

DESTINAZIONE Si tratta della semplice classe per la rappresentazione delle destinazioni con la presenza di un opportuno attributo chiamato 'predefinito' che permette di capire se una determinata destinazione è quella di default per il cliente specifico.

BLOCCO Come già indicato, la classe Blocco rappresenta un divieto di raccolta di ordini per un determinato cliente.

PRODOTTO La classe Prodotto rappresenta un generico articolo prodotto e venduto dall'azienda.
In particolare, i prodotti sono suddivisi in base a famiglie, sottofamiglie, gruppi e sottogruppi.
Relativamente alla loro deperibilità si gestisce un opportuno codice sul quale verranno effettuate appropriate considerazioni lato server.
In fine, ogni articolo presenta informazioni riguardo al prezzo e quantità minimi di vendita e ad un eventuale sconto massimo applicabile al prezzo.

PRODOTTO_UM Si tratta della classe modellante il concetto di unità di misura relativa all'articolo.
Per ogni prodotto esiste un'unità di misura principale ed eventualmente una

o più unità di misura secondarie.

In fine, ad ogni unità di misura è associato un fattore di conversione in modo da consentire le conversioni di quantità da unità di misura secondaria a principale.

GIACENZA La classe Giacenza permette di rappresentare, per ogni deposito, la quantità di prodotto depositata e quella effettivamente disponibile.

Per ognuna delle quantità e disponibilità indicate deve sempre essere associata un'unità di misura.

TIPO_ORDINE, TIPO_RIGA_ORDINE, TIPO_RIGA_OMAGGIO Si tratta di classi modellanti concetti aggiuntivi relativi agli ordini.

La tipologia dell'ordine permette di stabilire se l'ordinazione per un determinato articolo deve essere trattata come ordine vero e proprio oppure solo come preventivo.

Per tipologia di riga dell'ordine, invece, si intende qualcosa di più specifico: permette di capire se la riga dell'ordine ha come oggetto un articolo o altro.

In fine, la tipologia dell'omaggio permette di rappresentare il caso di un possibile omaggio per l'ordine.

VALUTA Si tratta della semplice classe rappresentante il concetto di valuta.

Nell'insieme delle valute utilizzabili, una di queste dev'essere specificata come default.

CONDIZIONE_PAGAMENTO Permette di esprimere informazioni sulle modalità di pagamento associate ai vari clienti.

Contiene principalmente informazioni descrittive ed un eventuale sconto per quel determinato tipo di pagamento.

ORDINE_TESTATA La testata di un ordine è un ordine vero e proprio contenente uno o più articoli ordinati da un determinato cliente.

Ad ogni cliente possono essere associate più testate, ma tra queste solo una può essere aperta, ovvero quella corrispondente all'ordine corrente non ancora inviato ai gestionali. La classe è caratterizzata da un codice che permette

di sapere quali sono le testate acquisite e chiuse, ovvero quelle già inviate ai gestionali e non più modificabili, e quali sono quelle aperte e in fase di elaborazione.

Per la testata è necessario gestire anche la data di consegna, corrispondente quindi a quella dell'intero ordine, e un campo descrittivo per le note.

Inoltre la testata ha associato un determinato cliente e una destinazione disponibile per quel cliente.

ORDINE_RIGA Il concetto di riga dell'ordine indica l'ordinazione, di uno specifico prodotto, inserita nella testata aperta per il cliente selezionato.

La riga dell'ordine contiene tutti i dettagli per l'ordinazione. In particolare, vi sono l'unità di misura principale, quella selezionata, la quantità in unità di misura selezionata, la quantità convertita in unità di misura principale, il prezzo calcolato da listino, l'eventuale prezzo modificato, lo sconto ricavato da listino, l'eventuale sconto aggiuntivo, l'importo totale di riga e il riferimento all'agente che ha creato l'ordine. Inoltre, anche per la riga dell'ordine si mantengono informazioni sulla data di consegna, sulla destinazione e sulle note, in quanto possono differire da quelle specificate per la testata. In questo modo, si fornisce la possibilità di distribuire i singoli prodotti, appartenenti alla stessa testata, in modalità diverse.

La singola riga dev'essere associata sempre a un cliente e a un articolo.

Ulteriori informazioni, senza le quali non è possibile creare una riga ordine, sono: valuta, tipologia dell'ordine, tipologia della riga ordine e tipologia dell'omaggio.

Concludendo, per quel che riguarda la deperibilità del prodotto si gestisce un apposito dato sul quale vengono effettuate delle considerazioni dalla componente server del sistema.

PRODOTTO_LISTINO Rappresenta l'insieme di informazioni relative al prezzo di un articolo.

In particolare, per ogni prodotto possono essere disponibili una serie di listini dipendenti da molteplici fattori, tra cui, eventualmente, anche l'area e la zona

di appartenenza del cliente. Inoltre, il listino può dipendere da famiglia, gruppo e categoria dell'articolo e da data di consegna e quantità selezionata.
Per il listino si modellano anche degli indici e una priorità che permettono di capire all'apposito algoritmo come devono essere trattati gli sconti e il prezzo sul listino specifico.
Concludendo, al listino viene associata anche una destinazione, altro fattore determinante nel calcolo del prezzo finale per il prodotto selezionato.

PRODOTTO_SCONTO Rappresenta l'insieme di informazioni relative allo sconto sul prezzo di un articolo.
Ogni sconto può essere associato con un cliente e con un prodotto, aventi entrambi una propria classe sconto di appartenenza.
Come il listino, anche lo sconto ha degli intervalli di data e quantità per i quali risulta applicabile e una priorità utilizzata dall'algoritmo per il calcolo dello sconto totale.

STORICO La classe Storico rappresenta l'ordine, salvato nel database aziendale, preso in passato relativamente ad un cliente e ad un articolo.
E' quindi caratterizzato da tutti i dettagli relativi all'ordine.

Lo schema definito risulta abbastanza efficiente dal punto di vista dell'organizzazione dei dati. Tuttavia, considerata la grande quantità di dati e la necessità di effettuare principalmente operazioni di lettura sulla base dati si preferisce l'utilizzo di un database completamente denormalizzato.
La denormalizzazione, infatti, è quel processo che permette di ottimizzare le performance in lettura di un database aggiungendo ridondanza nei dati o raggruppandoli. Il processo di denormalizzazione permette di ottenere una base dati in seconda forma normale, ma non in terza: questo significa che all'interno di una stessa relazione si possono avere attributi che dipendono da altri attributi non-chiave.
L'altra motivazione che conduce alla denormalizzazione è dovuta alla sincronizzazione, in quanto questa operazione si effettua singolarmente su ogni entità e,

quindi, la presenza di relazioni e vincoli tra queste introdurrebbe delle limitazione nello svolgimento di questa attività. Lo schema finale ottenuto è quindi il seguente:

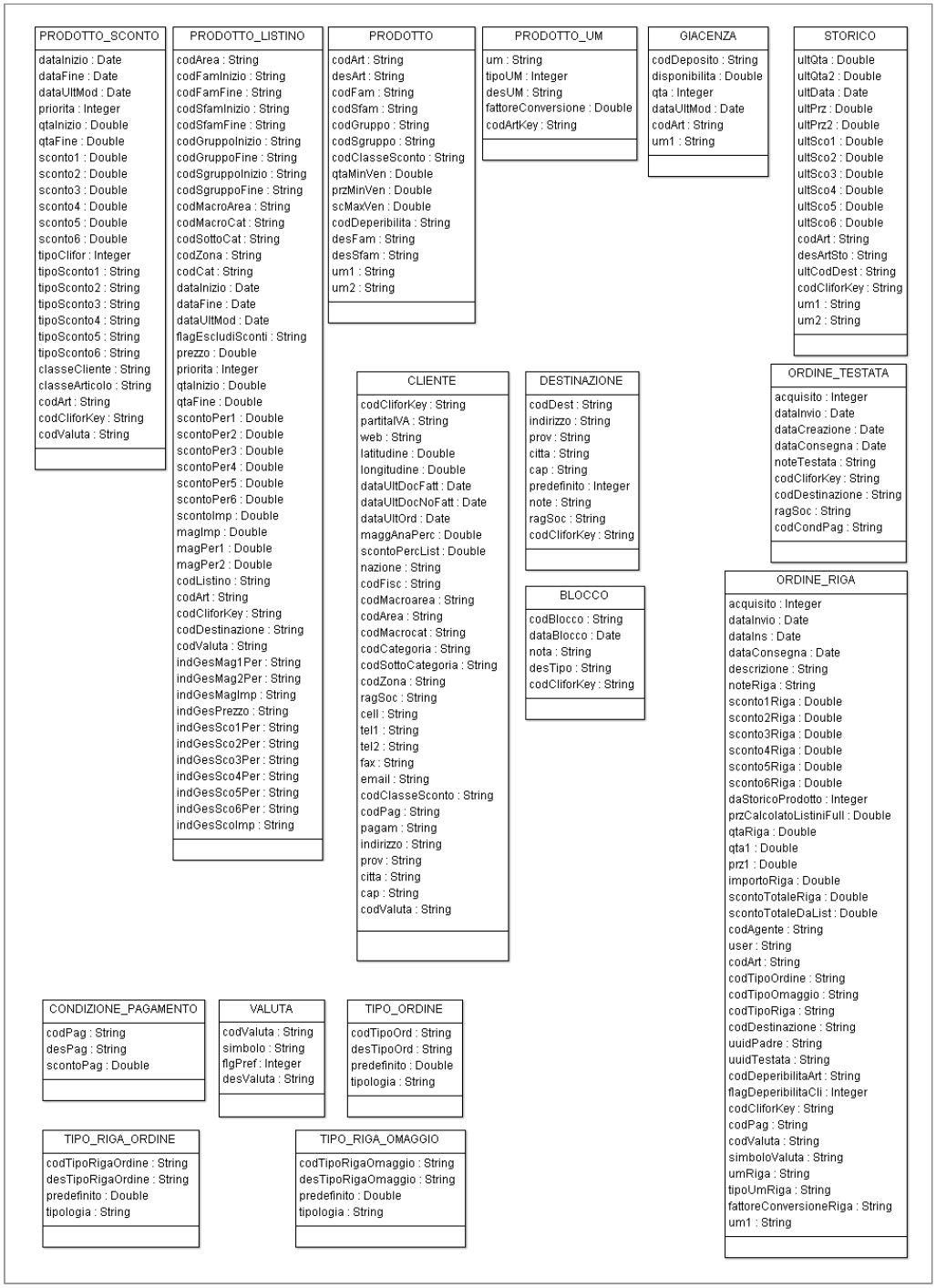

Figura 6: Schema del database

2.6.4 Sincronizzazione dei dati

La sincronizzazione dei dati può essere definita come la funzionalità più delicata dell'applicativo.
Il suo scopo è quello di ottenere i dati dal server, il quale, a sua volta, deve prelevarli dallo specifico sistema informativo aziendale.
I suoi prerequisiti sono quelli di aver effettuato il log-in all'applicativo, di aver selezionato un progetto, ovvero l'azienda con la quale si vuole operare, e di avere una connessione di rete disponibile. Come azione preliminare, si avvia un ulteriore procedura di login al server, attraverso la quale si controlla se l'utente ha l'autorizzazione per proseguire con la sincronizzazione. I possibili scenari in cui si impedisce all'utente di sincronizzare i dati sono i seguenti:

LICENZA SCADUTA l'utente non è più autorizzato all'utilizzo dell'applicativo in quanto è scaduto il termine da contratto;

APPLICAZIONE OBSOLETA l'applicazione non è aggiornata all'ultima versione;

SUPERATO IL NUMERO MASSIMO DI DEVICE l'utente ha effettuato l'accesso con troppi dispositivi.

Nel caso in cui l'utente non si trovi in nessuno di questi casi, si avvia la sincronizzazione.
La procedura si suddivide, quindi, in due fasi: download dei dati dal server e successiva persistenza sul database locale.
La sincronizzazione deve consentire all'utente di poter scegliere su quali dati avviare la richiesta attribuendo a quest'ultimi un certo grado di indipendenza. Quindi, in tal senso non si pongono vincoli restrittivi, ma solo il fatto che certe informazioni devono essere obbligatoriamente presenti per la presa dell'ordine. Tra queste vi sono:

TIPOLOGIA DELL'ORDINE consente all'agente di intraprendere attività diverse dalla vera raccolta dell'ordine; questo risulta molto utile, ad esempio, nei casi in cui voglia proporre un semplice preventivo;

TIPOLOGIA DELLA RIGA DELL'ORDINE si tratta di un informazione aggiuntiva sulla specifica riga ordine che permette di capire se si tratta di un articolo o altro;

TIPOLOGIA DELL'OMAGGIO indica se si è in condizione di un eventuale omaggio;

VALUTA è la semplice valuta utilizzata per l'inserimento di prezzi e importi;

CONDIZIONE DI PAGAMENTO si tratta di informazioni aggiuntive sul pagamento da effettuare per l'ordine.

Le altre entità, per le quali non è obbligatoria la sincronizzazione, sono le seguenti:

- clienti;
- blocchi sui clienti;
- destinazioni;
- prodotti;
- unità di misura;
- listini;
- sconti;
- storico dei prodotti;
- giacenze.

Inoltre, questa funzionalità comprende un'attenta gestione degli errori per segnalare con tempestività ciò che è stato aggiornato correttamente e ciò che non è andato a buon fine.

2.6.5 Creazione dell'ordine

La logica applicativa riguardo alla creazione degli ordini non è per niente banale. L'ordine è considerato come la composizione di due parti fondamentali:

TESTATA DELL'ORDINE con questo termine si definisce una sorta di carrello associato allo specifico cliente nel quale vengono inserite le cosiddette righe ordine con i prodotti o servizi richiesti; la testata può essere di due tipi:

APERTA è definita in questo modo una testata creata in locale sul device per uno specifico cliente, ma non ancora salvata nel gestionale aziendale;

CHIUSA è definita in questo modo una testata già salvata nel DB aziendale e, quindi, non più modificabile.

RIGHE DELL'ORDINE con questo termine vengono definite le ordinazioni relative ai singoli prodotti o servizi richiesti.

In questo modo, per ogni cliente non è possibile avere più testate aperte contemporaneamente e per ogni testata è possibile avere molteplici righe ordine. Inoltre la distinzione tra testata aperta e chiusa permette di capire quali sono gli ordini già inviati ai sistemi informativi aziendali e quali invece devono ancora essere spediti. Per la creazione di un nuovo ordine, o meglio, di una nuova riga ordine è necessario controllare l'esistenza o meno di una testata aperta per il cliente selezionato: nel primo caso l'ordine viene aggiunto alla testata già esistente, mentre nel secondo caso è necessario creare una nuova testata per il cliente e aggiungergli, successivamente, la riga ordine. L'ordine richiede, inoltre, la selezione di un prodotto e l'inserimento delle seguente informazioni:

DATA DI CONSEGNA DELL'ARTICOLO si riferisce alla data di consegna della riga ordine corrente; generalmente coincide con la data di consegna inserita per la testata, ma il cliente potrebbe desiderare di ricevere il prodotto corrente in una data diversa rispetto agli altri prodotti compresi nell'ordine;

NOTA PER L'ARTICOLO si tratta di una semplice nota che può essere inserita o meno all'interno della specifica riga ordine;

TIPOLOGIA DELL'ORDINE come indicato precedentemente è un'informazione obbligatoria per distinguere gli ordini dai preventivi o da altre iniziative da parte dell'agente;

DESTINAZIONE individua l'indirizzo di consegna della riga ordine; per uno stesso cliente possono essere presenti più destinazioni e nel caso in cui non venga selezionata nessuna destinazione, sarà compito poi del server capire la destinazione di default per quel cliente;

TIPOLOGIA DELLA RIGA DELL'ORDINE come indicato precedentemente è un dato obbligatorio per definire se la riga ordine corrente riguarda un articolo o altro;

DESCRIZIONE DELL'ARTICOLO si tratta di un semplice campo descrittivo riguardante il prodotto dell'ordine;

TIPOLOGIA DELL'OMAGGIO come indicato precedentemente è un'informazione obbligatoria su eventuali omaggi;

UNITÀ DI MISURA individua l'unità di misura selezionata per la riga ordine; in base all'articolo dell'ordine possono essere presenti più unità di misura disponibili;

QUANTITÀ RIFERITA ALL'UNITÀ DI MISURA SELEZIONATA individua la quantità di articolo desiderato;

UNITÀ DI MISURA PRINCIPALE individua l'unità di misura principale dell'articolo;

QUANTITÀ IN UNITÀ DI MISURA PRINCIPALE individua la quantità convertita in riferimento all'unità di misura principale;

PREZZO individua il prezzo per una singola unità del prodotto; si calcola con una procedura articolata dal listino prezzi, ma può essere modificato dall'agente se ha l'autorizzazione;

SCONTI individua eventuali sconti; in base al progetto selezionato, si abilitano un certo numero di sconti fino ad un massimo di sei;

SCONTO TOTALE DAL LISTINO individua lo sconto calcolato con un'apposita procedura;

TOTALE SCONTI individua il totale dello sconto applicato all'ordine;

PREZZO NETTO individua il prezzo scontato per una singola unità di prodotto;

IMPORTO individua l'importo totale della riga ordine.

Su queste informazioni, fondamentali per la presa dell'ordine, al fine di minimizzare gli errori che avvenivano con molta probabilità in passato e al fine di automatizzare il tutto, semplificando l'attività dell'agente o del venditore, si definiscono opportuni controlli e procedure di calcolo per alcuni dati specifici.
Le procedure di calcolo utilizzate nella raccolta degli ordini sono le seguenti:

CALCOLO DEL PREZZO DAI LISTINI RELATIVO ALLA SINGOLA UNITÀ DI PRODOTTO

Il prezzo relativo alla singola unità di prodotto si calcola attraverso un'apposita procedura. Quest'ultima si suddivide in tre fasi:

- rilevamento delle informazioni sui prezzi;
- esecuzione dell'algoritmo per il calcolo del prezzo lordo;
- calcolo del prezzo netto.

La prima fase consiste nel rilevare una serie di listini prezzi, salvati localmente, in base a determinate condizioni basate sul confronto tra le informazioni del singolo listino e quelle inserite attualmente per l'ordine corrente:

- il codice del cliente del listino deve essere uguale a quello selezionato per l'ordine oppure nullo;
- il codice della destinazione del listino deve essere uguale a quello selezionato per l'ordine oppure nullo;
- il codice dell'articolo del listino deve essere uguale a quello selezionato per l'ordine oppure nullo;
- il codice del listino deve essere uguale a quello del cliente selezionato per l'ordine oppure nullo;

- i codici di macro-categoria, categoria, sotto-categoria, macro-area, area e zona devono essere uguali a quelli del cliente selezionato per l'ordine oppure nulli;
- i codici di famiglia, sottofamiglia, gruppo e sottogruppo dell'articolo selezionato per l'ordine devono essere compresi in un intervallo di valori specificato per il singolo listino oppure devono essere nulli;
- la valuta del singolo listino deve corrispondere a quella selezionata per l'ordine;
- la data di consegna selezionata per l'ordine deve essere compresa in un intervallo di valori specificato per il singolo listino;
- la quantità selezionata per l'ordine deve essere compresa in un intervallo di valori specificato per il singolo listino.

Da questa prima fase, si ottengono, se presenti, una serie di listini prezzi dalla tabella PRODOTTO_LISTINO, che soddisfino le condizioni sopra indicate, sui quali applicare l'algoritmo che, tenendo conto della diversa priorità dei listini prezzi ottenuti, porterà alla definizione di prezzo lordo, sconti e maggiorazioni.

In fine, da queste informazioni è possibile ricavare il prezzo netto per l'ordine.

Questa procedura si esegue ogni volta che si modifica un dato sensibile dell'ordine che si sta creando, in modo che il prezzo sia sempre aggiornato. Inoltre, gli agenti, in base a precise autorizzazioni, possono modificare il prezzo dell'ordine, rifiutando quello proposto dai listini prezzi.

CALCOLO DELLO SCONTO TOTALE DAL LISTINO Si tratta di una procedura molto simile a quella appena descritta che permette di rilevare un eventuale sconto per l'ordine.

In questo caso, si rileva una serie di sconti dall'apposita tabella del database, in base a delle condizioni fondate sul confronto tra le informazioni dell'ordine che si sta creando e quelle dello sconto specifico:

- il codice di classe sconto del cliente e del prodotto selezionati per l'ordine devono corrispondere a quelli dello specifico sconto oppure nulli;
- il codice del prodotto e del cliente selezionati per l'ordine devono corrispondere a quelli presenti nelle informazioni dello sconto oppure nulli;
- la valuta dell'ordine deve corrispondere a quella presente nello sconto;
- la quantità e la data di consegna specificati per l'ordine devono essere compresi in un intervallo di valori specificato per lo sconto.

Da questa query applicata alla tabella PRODOTTO_SCONTO, si ricava l'insieme dei possibili sconti che rispettino le condizioni indicate.
Una volta ricavate queste informazioni, si esegue l'algoritmo che porta alla definizione di eventuali sconti fino ad un massimo di sei. In fine, da questi è possibile ottenere lo sconto totale da applicare all'ordine corrente.
Anche in questo caso, la procedura si esegue ogni volta che si modifica una determinata informazione dell'ordine che si sta creando e, inoltre, l'agente, sempre su apposita autorizzazione, può modificare gli sconti calcolati dalla procedura.

CALCOLO DELLO SCONTO TOTALE PER LA RIGA ORDINE Questa procedura permette di ricavare lo sconto totale da applicare all'ordine tenendo conto di quello già ricavato in precedenza ed eventuali ulteriori sconti introdotti dall'agente.

CALCOLO DELLA QUANTITÀ IN UNITÀ DI MISURA PRINCIPALE Considerata la possibilità di selezionare un'unità di misura differente da quella principale, se presente, è necessario mantenere sempre sia la quantità riferita all'unità di misura selezionata che quella relativa all'unità di misura principale.
Per ogni unità di misura si gestisce il fattore di conversione che permette cosi di ricavare la quantità in unità di misura principale da quella inserita nell'ordine corrente.

CALCOLO DEL PREZZO AL NETTO DEGLI SCONTI Il prezzo netto determinato dai listini o specificato dall'agente non considera ancora eventuali sconti.
Cosi è necessario determinare il prezzo finale riferito alla singola unità di

prodotto che sia comprensivo anche degli eventuali sconti calcolati con l'apposita procedura o inseriti manualmente dall'agente.

CALCOLO DELL'IMPORTO TOTALE DELLA RIGA ORDINE L'importo totale della riga ordine è semplicemente dato dal prodotto tra il prezzo al netto degli sconti e la quantità in unità di misura principale.

2.6.6 Invio degli ordini al server

Ogni volta che si crea un ordine per un determinato cliente, questo deve essere salvato nel database locale sul proprio dispositivo. Fino a quando l'ordine rimane esclusivamente in locale potrà essere modificato.
Alla disponibilità di una connessione di rete, l'agente può inviare, con un'unica richiesta al server, tutti gli ordini salvati localmente e non ancora inviati. In questo modo, si inviano al server gli ordini di tutti i clienti, compresi di testate e righe. Una volta che gli ordini sono giunti al server non potranno essere più modificati ma sarà possibile solo visualizzarli in un'apposita sezione dell'applicazione.

2.6.7 Ordini veloci

La funzionalità Ordini veloci dell'applicazione nasce dalle necessità di molti agenti che trattando con clienti abituali raccolgono ordini simili con cadenza periodica.
Cosi si fornisce una sorta di storico degli ordini effettuati in base a prodotti e clienti. In base allo storico, al momento delle presa dell'ordine di un cliente, si leggono gli ordini presi in passato e attraverso le semplici azioni di conferma, incremento o decremento della quantità, l'agente può raccogliere ulteriori ordini in una frazione di secondo.
Gli ordini presi dallo storico possono anche essere filtrati in base alla destinazione del cliente relativo all'ordine e ordinati a seconda di data, quantità e articolo. Inoltre, per una maggior velocità e usabilità, è possibile anche definire lo step di incremento e decremento della quantità.

Questa funzionalità permette anche di raccogliere gli ordini partendo sempre dallo storico, ma apportando maggiori modifiche rispetto alla semplice variazione di quantità: in questo caso si apre l'ordine in modalità modifica caricando inizialmente tutte le informazioni prese dallo storico degli ordini.
Come nel caso della raccolta classica dell'ordine, anche in questo caso il salvataggio delle informazioni si effettua, inizialmente, solo nel database locale.

2.6.8 Flusso di creazione dell'ordine

Il flusso interno dell'applicativo per la creazione di un ordine può essere riassunto con il seguente diagramma delle attività UML:

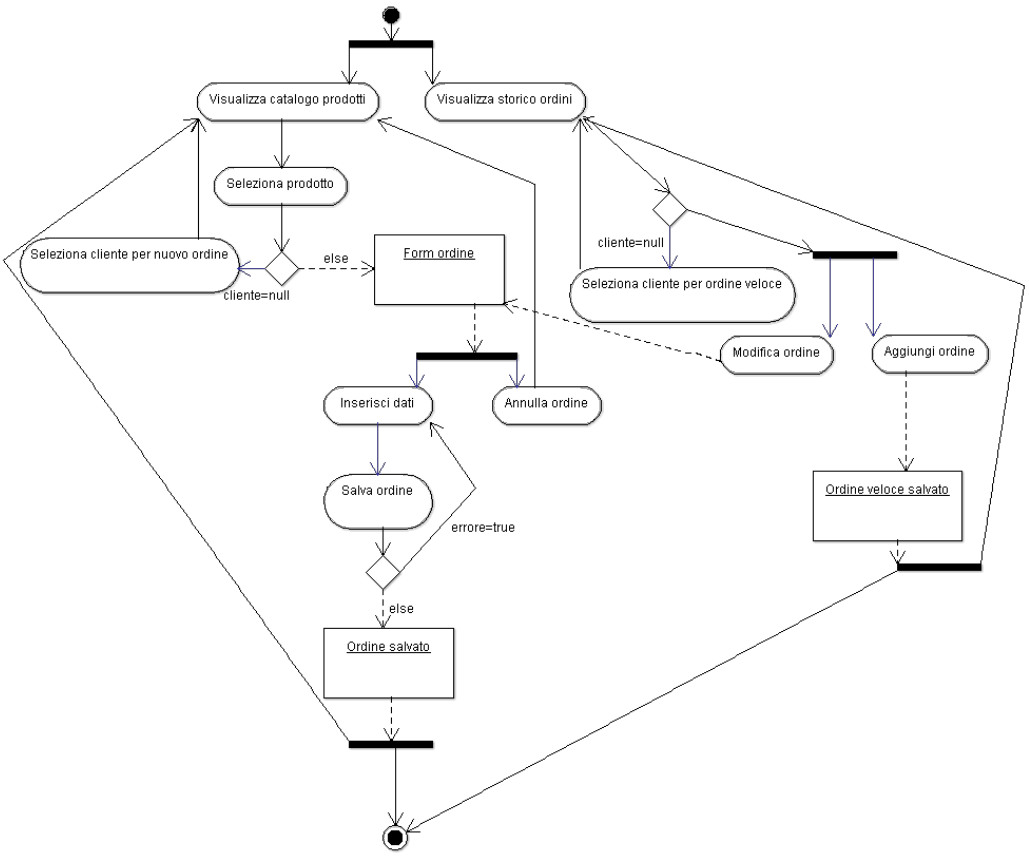

Figura 7: Diagramma delle attività - creazione dell'ordine

In particolare, l'ordine può essere creato partendo dal catalogo prodotti oppure dallo storico degli ordini.

Per poter procedere alla creazione dell'ordine è necessario che ci sia un cliente selezionato.
Per una maggior facilità di utilizzo dell'applicativo è sufficiente selezionare il cliente una sola volta; questo perché si presuppone che l'agente non lavora con più clienti contemporaneamente ma raccoglie una serie di ordini per un cliente e successivamente si sposta verso un altro: a quel punto l'agente dovrà semplicemente cambiare il cliente selezionato all'interno dell'applicazione.

2.7 SERVER

La parte Server dell'architettura risulta fondamentale per autenticarsi al sistema, poter ricavare informazioni relative all'azienda in qualsiasi momento (ove presente una connessione di rete) e inviare ordini o altre tipologie di dati ai gestionali aziendali.
Si suddivide principalmente in due parti:

- LICENSE MANAGER
- APP MANAGER

2.7.1 License Manager

Il License Manager è la componente che permette di gestire e autenticare gli accessi degli utenti all'applicazione.
Per poter sfruttare tutte le funzionalità dell'applicativo, l'utente (che sia agente o meno) deve aver effettuato il login con le proprie credenziali.
In realtà non si tratta di un classico login, ma è qualcosa di più articolato. In particolare, il license manager si aspetta una richiesta con le seguenti informazioni:

- identificativo dell'applicazione;
- versione dell'applicazione;
- identificativo del device;

- username;

- password.

Ricevuta la richiesta, il license manager deve verificare le informazioni sulla base di quelle contenute nel proprio database e rispondere fornendo le seguenti indicazioni:

CODICE DI RISPOSTA indica se l'autenticazione dell'utente è andata a buon fine oppure si è verificato un errore come nel caso di licenze scadute o credenziali non corrette;

CONFIGURAZIONE TABLET indica un insieme di informazioni che stabiliscono alcune autorizzazioni dell'utente, come la possibilità di visualizzare o meno certi moduli dell'app;

LISTA PROGETTI indica tutte le possibili aziende per cui l'utente presta servizio; è necessario sin da subito selezionare il progetto corrispondete all'azienda per cui lavora attualmente e in ogni momento, come già detto, è possibile cambiarlo;

INDIRIZZO APP MANAGER indica l'indirizzo web al quale si trovano i dati del progetto selezionato.

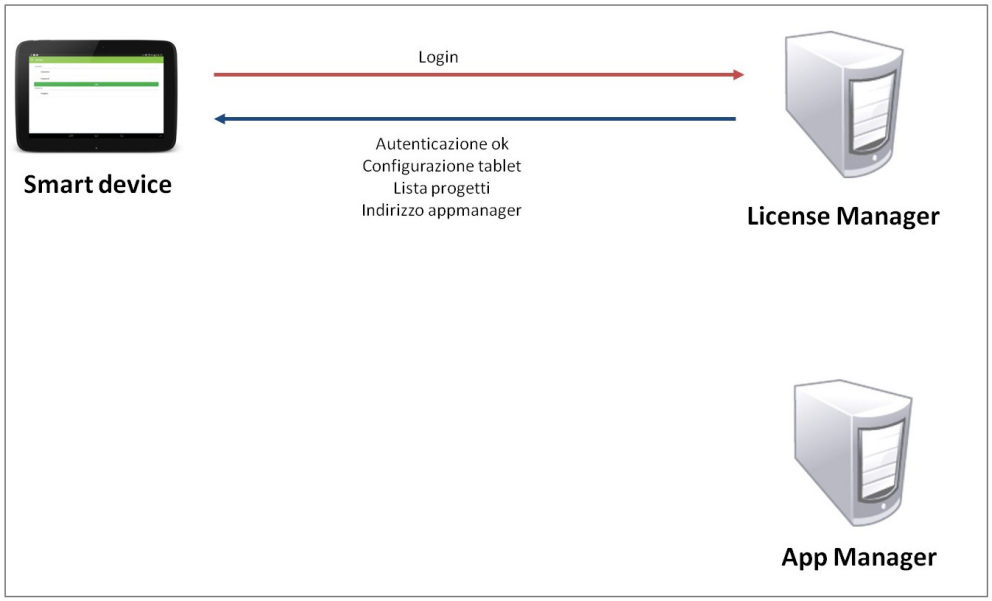

Figura 8: Funzionamento del License Manager

2.7.2 App Manager

L'App Manager è la componente server che interviene nella gestione dei dati veri e propri, sia in fase di sincronizzazione dai sistemi aziendali ai device che in quella di invio degli ordini dai device ai database delle aziende. Può essere vista come una sorta di interfaccia tra la diversa natura dei molteplici database aziendali e l'applicazione client eseguita sugli smart device.

In fase di sincronizzazione, la richiesta comprende l'indirizzo associato al progetto selezionato e la tipologia di informazioni richieste come clienti, prodotti e molto altro.

L'app manager, ricevuta la richiesta, rileva i dati richiesti nel proprio database inviandogli, poi, nella risposta allo smart device. In realtà, al fine di ottimizzare il processo di sincronizzazione, il reperimento di una certa categoria di informazioni si realizza con una richiesta di tipo incrementale: le informazioni non si ottengono in un'unica connessione al server, ma si suddividono su molteplici connessioni client-server.

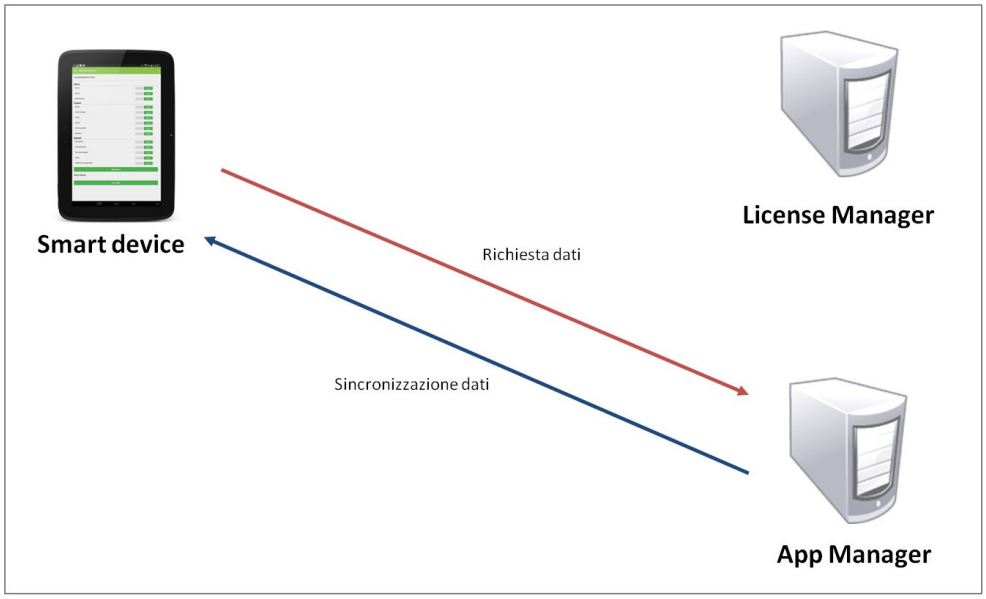

Figura 9: Sincronizzazione dati

Per poter disporre sempre di dati aggiornati, l'app manager deve importare i dati dai gestionali sulla propria base dati:

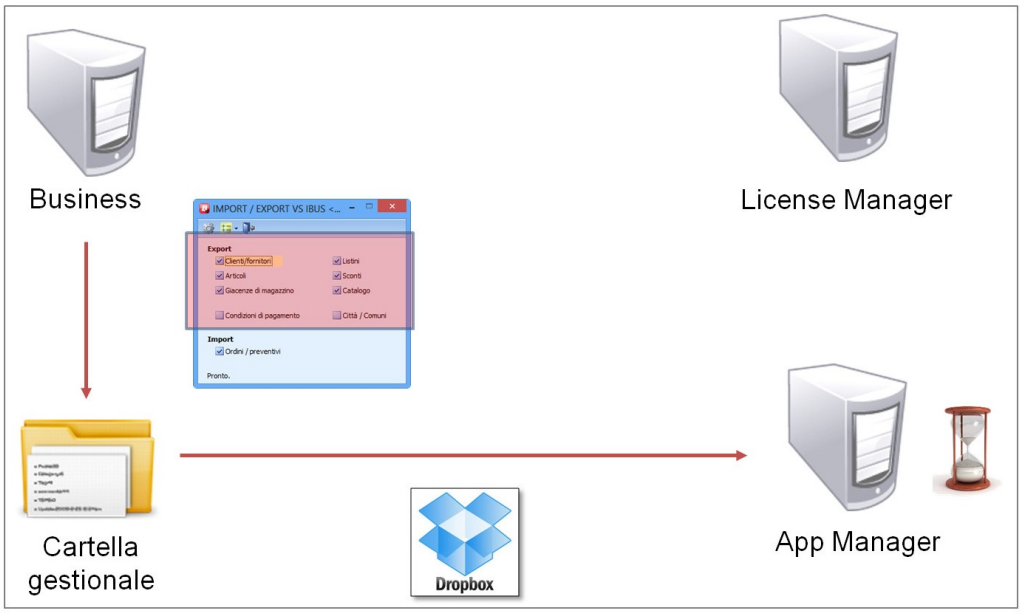

Figura 10: Importazione dei dati da parte dell'app manager

Per quel che riguarda l'invio degli ordini, questi vengono ricevuti dall'app manager all'interno di un'apposita richiesta dello smart device. L'app manager quindi procede con il salvataggio degli ordini nel proprio database e, dopo un intervallo di tempo di cinque minuti, si archiviano nella cartella dropbox associata all'azienda. A questo punto, dal gestionale aziendale è possibile effettuare l'import degli ordini dalla cartella dropbox.

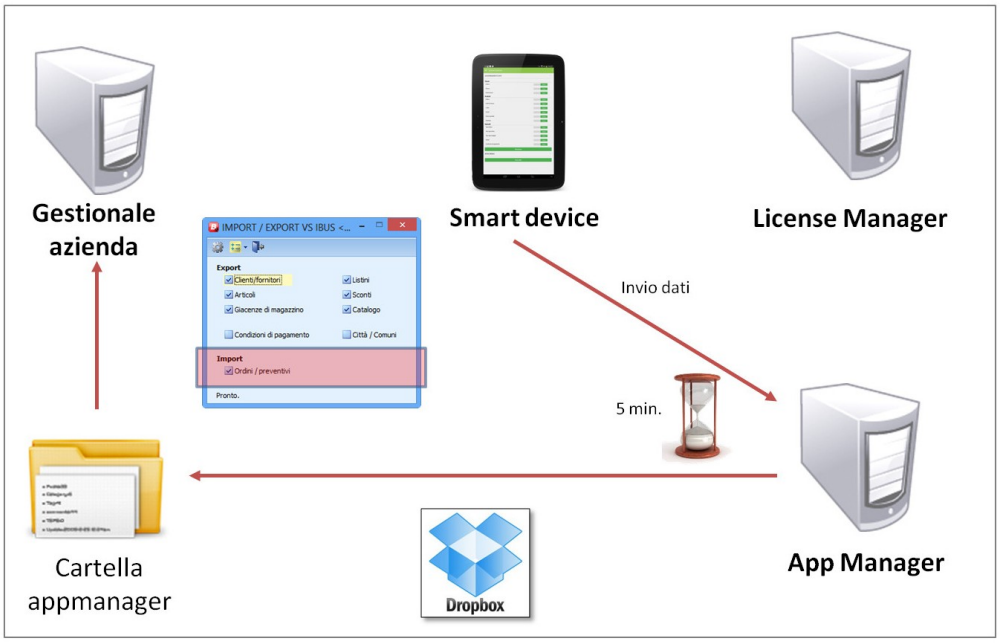

Figura 11: Invio degli ordini ai gestionali

3 IMPLEMENTAZIONE

3.1 PIATTAFORMA ANDROID

Come già detto, il fattore chiave del progetto è quello della portabilità.
In questo senso, è necessario effettuare una scelta riguardo allo sviluppo dell'applicazione che potrebbe essere realizzata per qualsiasi sistema mobile presente sul mercato. In questo caso, si è scelto il sistema di Google: Android. Il principale motivo della scelta è la sua grande diffusione su molteplici dispositivi come rivelano le recenti statistiche del mercato mobile. In questo modo, si fornisce all'utente più flessibilità nella scelta dello smart device sul quale utilizzare l'applicazione.
Una delle principali difficoltà nello sviluppo di applicazioni Android è la gestione di più versioni del sistema e di varie dimensioni relative ai dispositivi. In questo contesto, trattandosi di un applicativo vasto e modulare, si attribuisce una maggior priorità ai tablets, in quanto la grandezza dello schermo permette una miglior usabilità.

3.2 AMBIENTE DI SVILUPPO

Google, per riuscire a concorrere con il colosso Apple, ha apportato notevoli migliorie riguardo al proprio sistema mobile. La più recente è relativa all'introduzione di un nuovo ambiente di sviluppo per le applicazioni in sostituzione di Eclipse [1].

1 Ambiente di sviluppo integrato multi-linguaggio e multipiattaforma, ideato da un consorzio di grandi società quali Ericsson, HP, IBM, Intel, MontaVista Software, QNX, SAP e Serena Software, chiamato Eclipse Foundation sullo stile dell'open source.

3.2.1 Android Studio

L'introduzione di Android Studio rappresenta un grande passo in avanti per il mondo Android, in quanto fino ad oggi si appoggiava su Eclipse, ovvero un ambiente di sviluppo esterno, non di proprietà di Google Inc., e dedicato allo sviluppo di diverse tipologie di applicazioni. Con Android Studio, quindi, Google Inc. vanta di un proprio ambiente di sviluppo dedicato esclusivamente allo sviluppo di applicativi Android.
In questo senso, lo sviluppo di questo progetto sfrutta proprio le potenzialità di questo nuovo ambiente di sviluppo.

3.2.2 Gradle

Integrato al nuovo ambiente Android Studio, vi è la presenza di Gradle, un avanzato sistema di build che consente di definire una propria logica di compilazione con numerosissime personalizzazioni.
Questo nuovo sistema facilita, anche per questo progetto, la realizzazione di più versioni specifiche a partire dall'applicazione standard.
In questo modo, all'interno di ogni versione è possibile definire le personalizzazioni richieste dal committente.

3.3 APPLICAZIONE

Sul Play Store di Google è possibile trovare un'infinità di applicazioni, ma quella realizzata per questo progetto risulta unica nel suo genere e, al momento, risulta difficile trovarne una simile.
L'app si presenta con la seguente interfaccia:

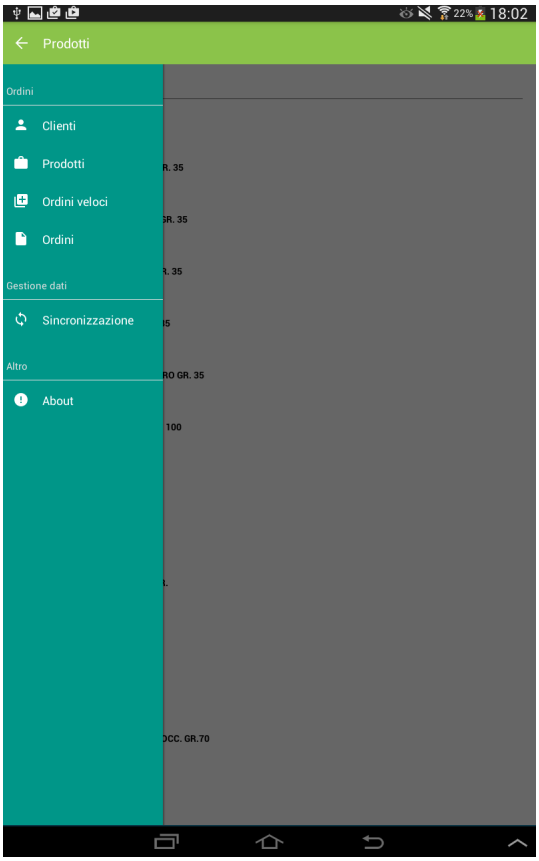

Figura 12: Struttura applicazione

Sono presenti tre sezioni principali:

- raccolta degli ordini;
- gestione dei dati;
- informazioni sull'app.

La prima sezione contiene le sottosezioni riguardanti le informazioni di tutti i clienti disponibili, la lista di tutti i prodotti che possono essere ordinati, la visualizzazione di tutti gli ordini (sia quelli inviati ai gestionali che quelli salvati solo sul device) e la funzionalità dell'ordine veloce.
La seconda sezione, invece, contiene le funzionalità di sincronizzazione dei dati e di salvataggio degli ordini sui gestionali aziendali.
In fine, l'ultima sezione permette di verificare la versione corrente dell'applicativo e la possibilità di inviare eventuali segnalazioni.

3.3.1 Definizione dello schema del database

Il primo punto cruciale nello sviluppo dell'applicazione riguarda la creazione e la gestione di un database sul dispositivo.
Per la creazione del db si utilizza la tecnologia SQLite, una libreria software multi-piattaforma scritta in linguaggio C che permette di creare basi di dati in un unico file.
SQLite rappresenta la libreria standard utilizzata per la gestione dei dati in Android per diverse ragioni:

- è molto veloce;
- è in grado di interpretare stringhe SQL;
- ha transazioni che soddisfano le proprietà ACID;
- supporta database che possono essere anche molto grandi;
- presenta una database di un unico file con formato indipendente dalla piattaforma.

Tuttavia sviluppare utilizzando esclusivamente questa libreria per la gestione dei dati comporta del lavoro aggiuntivo e l'aumento di errori dovuti alla scrittura manuale di query SQL e alla manipolazione complessa dei risultati delle varie interrogazioni al database.
Quindi, considerata anche la grande quantità di dati da gestire, in supporto al database SQLite si utilizza la tecnica ORM (Object-Relational Mapping). Quest'ultima favorisce l'integrazione di sistemi software aderenti al paradigma della programmazione orientata agli oggetti con sistemi RDBMS[2] (Relational database management system). Un prodotto ORM fornisce, mediante un'interfaccia orientata agli oggetti, tutti i servizi inerenti alla persistenza dei dati, astraendo nel contempo le caratteristiche implementative dello specifico RDBMS utilizzato.
I principali vantaggi dati dal suo utilizzo sono i seguenti:

2 Introdotto da Edgar F. Codd, indica un sistema per la gestione di basi di dati relazionali.

- il superamento (più o meno completo) dell'incompatibilità di fondo tra il progetto orientato agli oggetti ed il modello relazionale sul quale è basata la maggior parte degli attuali RDBMS utilizzati;

- un'elevata portabilità rispetto alla tecnologia DBMS[3] (Database Management System) utilizzata: cambiando DBMS non devono essere riscritte le routine che implementano lo strato di persistenza; generalmente basta cambiare poche righe nella configurazione del prodotto per l'ORM utilizzato;

- drastica riduzione della quantità di codice sorgente da redigere; l'ORM maschera dietro semplici comandi le complesse attività di creazione, prelievo, aggiornamento ed eliminazione dei dati. Tali attività occupano di solito una buona percentuale del tempo di stesura, testing e manutenzione complessivo. Inoltre, sono per loro natura molto ripetitive e, dunque, favoriscono la possibilità che vengano commessi errori durante la stesura del codice che le implementa;

- suggerisce la realizzazione dell'architettura di un sistema software mediante approccio stratificato, tendendo pertanto ad isolare in un solo livello la logica di persistenza dei dati, a vantaggio della modularità complessiva del sistema;

Tra i numerosi progetti ORM presenti relativamente alla piattaforma Android, la scelta ricade su quello che garantisce stabilità, ma soprattutto ottime performance per quel che riguarda la persistenza dei dati sul database: tutto questo viene dato dal progetto greenDAO.
GreenDao è un ORM open source ottimizzato per Android con altissime performance, basso consumo di memoria, APIs semplici da utilizzare e una libreria da piccole dimensioni.
Il suo obiettivo è quindi quello di effettuare il mapping tra oggetti e database sqlite:

3 Sistema software, ospitato su architettura hardware dedicata o su semplice computer, progettato per consentire la creazione, la manipolazione e l'interrogazione efficiente di database.

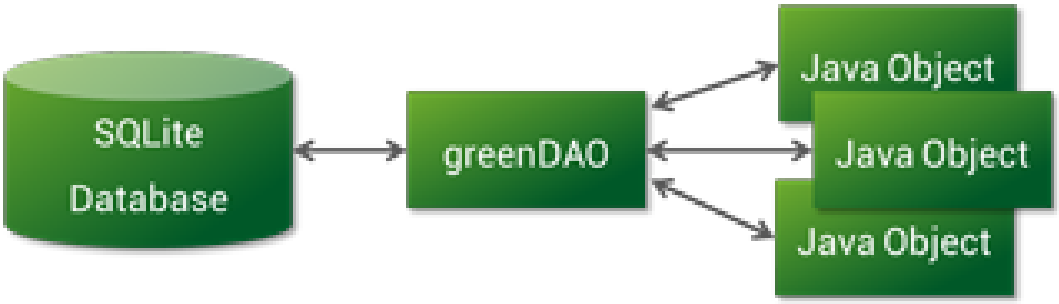

Figura 13: greenDao - Android ORM per SQLite

Per raggiungere tale obiettivo, greenDao necessità della definizione dello schema del database tramite un applicativo java, dove si andranno a definire entità, proprietà, indici, relazioni e molto altro.
Questo progetto java può essere considerato come un generatore la cui esecuzione produce le apposite entità e gli oggetti Dao, utilizzati per l'esecuzione di operazioni sul db. Il risultato dell'esecuzione dev'essere direttamente collocato all'interno del progetto Android costituente l'applicazione realizzata.
La sua struttura può essere riassunta con la seguente figura:

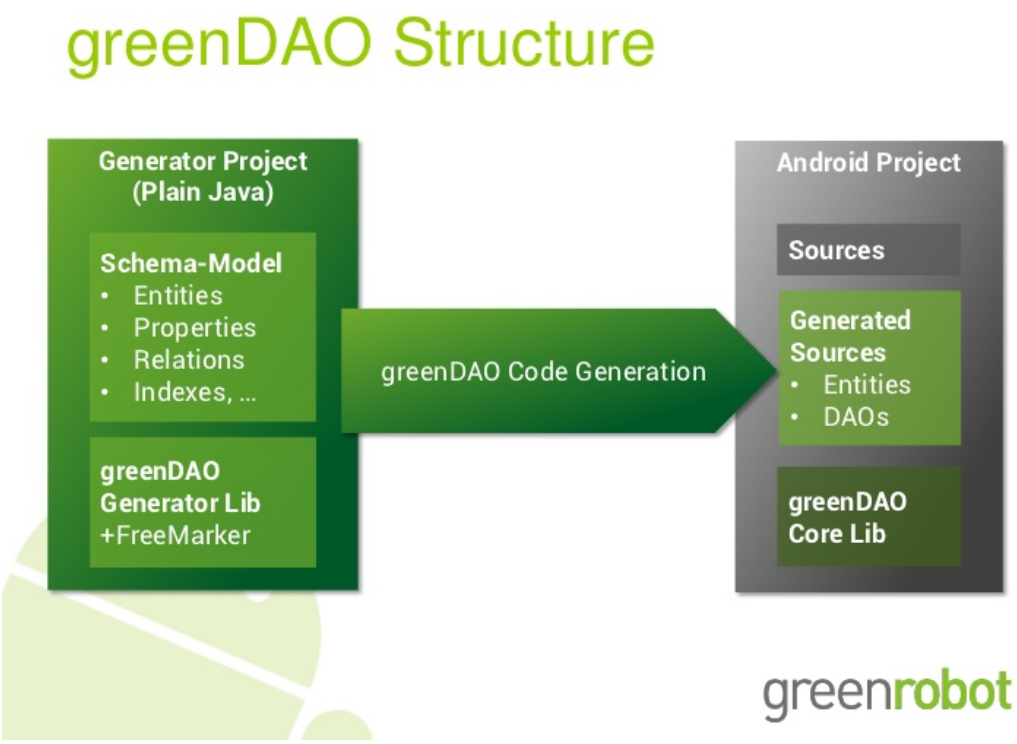

Figura 14: greenDao - Struttura e integrazione con progetto Android

Per effettuare il mapping oggetti-modello relazionale è necessario, quindi, definire lo schema del db tramite un'apposita applicazione java:

```
public static void main(String[] args) throws Exception {

                Schema scheme = new Schema(1, "it.apexnet.
                    orderandroid.data");

                addCliente(scheme);
                addClienteBlocco(scheme);
                addClienteDestinazione(scheme);
                addCondizionePagamento(scheme);
                addProdotto(scheme);
                addProdottoUM(scheme);
                addTipoOrdine(scheme);
                addTipoRigaOmaggio(scheme);
                addTipoRigaOrdine(scheme);
                addValuta(scheme);
                addOrdineTestata(scheme);
                addOrdineTemp(scheme);
                addProgetto(scheme);
                addProdottoListino(scheme);
                addProdottoSconto(scheme);
                addProdottoStorico(scheme);
                addGiacenza(scheme);
                scheme.enableKeepSectionsByDefault();

                new DaoGenerator().generateAll(scheme, "../
                    OrderAndroid/app/src/main/java");
            }
```

L'applicazione sfrutta la libreria e il generatore greenDao.
In particolare, è richiesta la definizione di un oggetto Schema indicando la versione del database e il package del progetto Android nel quale collocare tutte le classi generate da greenDao. Ad ogni esecuzione dell'applicazione Java, lo schema viene rigenerato e, di conseguenza, tutte le classi relative allo schema, presenti nel progetto Android, vengono sovrascritte. Per evitare la perdita di librerie, proprietà e funzioni aggiunte manualmente alle classi rappresentanti le entità del sistema, greenDao offre la possibilità di conservarle inserendole in apposite sezioni generate dal metodo 'enableKeepSectionsByDefault'. Tutta l'implementazione introdotta in queste sezioni non viene mai modificata dal generatore greenDao.
Per la definizione dello schema è necessario creare le diverse entità definite in fase

di progettazione e corrispondenti alla tabelle che andranno a popolare la base di dati presente sul device.

Per ogni entità aggiunta allo schema si definisce l'insieme delle proprietà, come nel caso della riga ordine, qui chiamata OrdineTemp:

```
private static void addOrdineTemp(Schema scheme){

                Entity ordine=scheme.addEntity("OrdineTemp");
                ordine.setTableName("OrdineTemp");
                ordine.implementsInterface("Parcelable");

                ordine.addStringProperty("uuid").primaryKey().
                    columnName("uuid");
                ordine.addIntProperty("acquisito").columnName("
                    acquisito");
                ordine.addStringProperty("codArt").columnName("
                    codArt");
                ordine.addStringProperty("codAgente").columnName("
                    codAgente");
                ordine.addDateProperty("dataInvio").columnName("
                    dataInvio").columnType("TEXT");
                ordine.addDateProperty("dataIns").columnName("
                    dataIns").columnType("TEXT");
                ordine.addDateProperty("dataConsegna").columnName(
                    "dataConsegna").columnType("TEXT");
                ordine.addStringProperty("codTipoOrdine").
                    columnName("codTipoOrdine");
                ordine.addStringProperty("codTipoOmaggio").
                    columnName("codTipoOmaggio");
                ordine.addStringProperty("codTipoRiga").columnName
                    ("codTipoRiga");
                ordine.addStringProperty("codDestinazione").
                    columnName("codDestinazione");
                ordine.addStringProperty("descrizione").columnName
                    ("descrizione");
                ordine.addStringProperty("noteRiga").columnName("
                    noteRiga");
                ordine.addDoubleProperty("sconto1Riga").columnName
                    ("sconto1Riga");
                ordine.addDoubleProperty("sconto2Riga").columnName
                    ("sconto2Riga");
                ordine.addDoubleProperty("sconto3Riga").columnName
                    ("sconto3Riga");
                ordine.addDoubleProperty("sconto4Riga").columnName
                    ("sconto4Riga");
                ordine.addDoubleProperty("sconto5Riga").columnName
                    ("sconto5Riga");
                ordine.addDoubleProperty("sconto6Riga").columnName
                    ("sconto6Riga");
                ordine.addIntProperty("daStoricoProdotto").
                    columnName("daStoricoProdotto");
```

```
        ordine.addStringProperty("uuidPadre").columnName("
            uuidPadre");
        ordine.addStringProperty("uuidTestata").columnName
            ("uuidTestata");
        ordine.addStringProperty("codDeperibilitaArt").
            columnName("codDeperibilitaArt");
        ordine.addIntProperty("flagDeperibilitaCli").
            columnName("flagDeperibilitaCli");
        ordine.addStringProperty("codCliforKey").
            columnName("codCliforKey");
        ordine.addStringProperty("user").columnName("user"
            );
        ordine.addStringProperty("codPag").columnName("
            codPag");
        ordine.addStringProperty("codValuta").columnName("
            codValuta");
        ordine.addStringProperty("simboloValuta").
            columnName("simboloValuta");
        ordine.addDoubleProperty("przCalcolatoListiniFull"
            ).columnName("przCalcolatoListiniFull");
        ordine.addStringProperty("umRiga").columnName("
            umRiga");
        ordine.addIntProperty("tipoUmRiga").columnName("
            tipoUmRiga");
        ordine.addDoubleProperty("fattoreConversioneRiga")
            .columnName("fattoreConversioneRiga");
        ordine.addDoubleProperty("qtaRiga").columnName("
            qtaRiga");
        ordine.addStringProperty("um1").columnName("um1");
        ordine.addDoubleProperty("qta1").columnName("qta1"
            );
        ordine.addDoubleProperty("prz1").columnName("prz1"
            );
        ordine.addIntProperty("indexUm").columnName("
            indexUm");
        ordine.addDoubleProperty("importoRiga").columnName
            ("importoRiga");
        ordine.addDoubleProperty("scontoTotaleRiga").
            columnName("scontoTotaleRiga");
        ordine.addDoubleProperty("scontoTotaleDaList").
            columnName("scontoTotaleDaList");

    }
```

La chiave primaria della tabella OrdineTemp, come si può vedere, è impostata sull'attributo uuid che rappresenta quindi il codice identificativo della riga dell'ordine.

Per ogni attributo si definiscono la tipologia e il nome che deve assumere la corrispondente colonna nella tabella. Si noti che SQLite presenta lo svantaggio di non

supportare direttamente la gestione del tipo data, ma è necessario utilizzare una rappresentazione testuale o numerica ed effettuare le opportune operazioni di parsing.
Ciascuna entità utilizza l'interfaccia Parcelable, un'interfaccia specifica di Android che permette il passaggio di oggetti specifici, ad esempio un cliente, da una sezione all'altra dell'applicazione.
La vera mappatura di entità in tabelle è resa possibile dagli oggetti Dao; il generatore greenDao, oltre alla classe Java rappresentante l'entità specificata nello schema, crea la classe Dao corrispondente che viene utilizza da greenDao come una sorta di interfaccia tra oggetto e tabella. All'interno di quest'ultima viene generato anche il codice sql necessario alla creazione della tabella corrispondente; attraverso questo meccanismo si evita la scrittura di procedure sql con conseguente minimizzazione degli errori.

Per quel che riguarda le entità coinvolte nella sincronizzazione risulta fondamentale l'aggiunta di un'annotazione della libreria Gson utilizzata per le operazioni di parsing dei dati; l'annotazione, sostanzialmente, consente di indicare il nome dell'attributo ricavato dalla risposta data dall'app manager. In questo modo il valore del dato ottenuto durante la sincronizzazione viene correttamente mappato nella proprietà dell'oggetto.

```
valuta.addStringProperty("codValuta").primaryKey().
    annotationGson("cod_valuta").columnName("codValuta");
```

Si prenda l'esempio della valuta; il dato cod_valuta, corrispondente al codice identificativo della valuta, tramite l'utilizzo della libreria Gson viene mappato nella proprietà codValuta dell'oggetto 'Valuta' e attraverso un salvataggio sulla base di dati viene salvato nell'apposita colonna codValuta della tabella 'Valuta'.

3.3.2 Creazione e migrazione del database

Una volta definito lo schema, nel solito package del progetto Android vengono create altre due classi fondamentali: DaoMaster e DaoSession.
In particolare, la prima contiene le funzionalità di 'createTable' che saranno ese-

guite alla creazione del database sul dispositivo e la classe OpenHelper di SQLite. Quest'ultima si occupa di controllare l'effettiva presenza del db sul dispositivo al momento dell'esecuzione dell'applicazione Android e in caso contrario provvede alla sua creazione. In caso di esistenza del database, invece, attraverso la classe 'DaoSession' viene stabilita la sessione per l'interazione con la base dati.

Considerato che il database locale rappresenti il cuore dell'app e che questo possa continuamente variare la sua struttura in conseguenza a richieste di aggiunta, modifica o rimozione di tabelle e attributi, si richiede l'implementazione di un'attenta migrazione dei dati. Questo significa che dev'essere concessa la possibilità di effettuare aggiornamenti sul db attraverso l'incremento della sua versione corrente, senza causare la perdita dei dati salvati attualmente. A tal fine è necessaria la specializzazione della classe OpenHelper che oltre alla funzionalità di creazione del database offre quella di aggiornamento. Si definisce cosi il comportamento desiderato per la funzionalità di aggiornamento: ogni volta che vengono apportate modifiche al db e che viene, di conseguenza, incrementata la versione corrente, l'Open Helper alla successiva apertura del database rileva il cambiamento di versione e si preoccupa dell'esecuzione degli script sql necessari all'aggiornamento, senza causare perdite di dati:

```
    @Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int
    newVersion) {

     /* i represent the version where the user is now and the
        class named with this number implies that is upgrading
         from i to i++                          schema */
    for (int i = oldVersion; i < newVersion; i++) {

        /* New instance of the class that migrates from i
            version to i++ version named DBMigratorHelper{
            version that the db has on
                                                this moment} */
        AbstractMigratorHelper migratorHelper = null;
        try {
            migratorHelper = (AbstractMigratorHelper) Class.
                forName("it.apexnet.orderandroid.migration.
                DBMigrationHelper" + i).newInstance();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
```

```
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }

            if (migratorHelper != null) {
                    /* Upgrade db */
                    migratorHelper.onUpgrade(db,context);
            }


        }
```

3.3.3 Interrogazione del database

Tutte le query effettuate sulla base dati vengono eseguite tramite gli oggetti Dao. Come già indicato, l'utilizzo del progetto greenDao permette l'esecuzione di query in maniera object-oriented evitando la scrittura di codice sql.
Un semplice esempio è dato dalla rilevazione dei clienti:

```
    public List<Cliente> getCustomers() {
        return customerDao.queryBuilder().orderAsc(ClienteDao.
            Properties.RagSoc).list();
    }
```

In questo caso, sull'oggetto Dao relativo ai clienti viene eseguita la query tramite il metodo queryBuilder(), ordinando la lista ritornata in base alla ragione sociale del cliente.

3.3.4 Accesso all'applicazione

Per poter sfruttare a pieno le potenzialità dell'app, l'utente deve effettuare l'accesso autenticandosi al license manager.
Questo si realizza tramite una chiamata di tipo Http Post al license manager con l'inserimento nel body della chiamata di un apposito json contenente informazioni relative a id e nome dell'applicazione android, identificativo del device, nome e password dell'utente.

```
                HttpClient httpclient;
```

```
        if (param[o].type == CallWsParameters.CallWsType.POST) {
            HttpPost post = new HttpPost(param[o].urlWs);
            try {
                if (param[o].hasHeaders()) {
                    for(String key : param[o].getHeaders().keySet
                        ())
                        post.addHeader(key, param[o].getHeaders().
                            get(key));
                }
                else
                    post.addHeader("Content-Type", "application/
                        json");

                if (!TextUtils.isEmpty(param[o].body))
                {
                    StringEntity input = new StringEntity(param
                        [o].body);
                    post.setEntity(input);
                }

                response = httpclient.execute(post);

            } catch (Exception e) {
                Log.c(TAG, "call post exception: " + e.getMessage
                    (),
                        LogType.ERROR);
            }
        }
```

La chiamata si effettuata in un task asincrono grazie all'oggetto HttpClient al quale viene passato un apposito oggetto HttpPost contenente il body con il json da inviare al server, l'header e l'url che identifica il servizio.

In caso di autenticazione avvenuta con successo, viene ritornata dal license manager la lista dei progetti, associati all'utente, che vengono salvati sul database locale. Inoltre, al completamento del salvataggio viene mostrata un'apposita dialog android nel quale selezionare il progetto desiderato: questo equivale a scegliere l'azienda per la quale si presta attività in quel preciso momento. Tra le informazioni relative al progetto, quelle più importanti sono le seguenti:

```
                progetto.addIntProperty("bloccaPrezzoMinVen").
                    columnName("bloccaPrezzoMinVen");
                progetto.addIntProperty("bloccaQtaMinOrd").
                    columnName("bloccaQtaMinOrd");
                progetto.addIntProperty("bloccaScontoMaxVen").
                    columnName("bloccaScontoMaxVen");
                progetto.addIntProperty("numSconti").columnName("
                    numSconti");
```

```
progetto.addStringProperty("codAgente").columnName
    ("codAgente");
progetto.addStringProperty("url_ws").columnName("
    url_ws");
progetto.addIntProperty("visDisponib").columnName(
    "visDisponib");
progetto.addIntProperty("visGiacenza").columnName(
    "visGiacenza");
```

Gli attributi bloccaPrezzoMinVen e bloccaQtaMinOrd sono dei flag che permettono di stabilire, in fase di creazione della riga ordine, se è possibile specificare un prezzo e una quantità inferiori ai valori minimi indicati per il prodotto selezionato. Allo stesso modo bloccaScontoMaxVen permette di autorizzare o meno l'agente nella determinazione di un sconto superiore al massimo consentito per il prodotto.

Altro dato molto importante è numSconti che indica il numero di sconti da poter inserire in una riga ordine: il massimo è sei, ma in base all'azienda selezionata alcuni di questi sono disabilitati.

L'attributo codAgente è diverso dal valore nullo solo se l'utente che ha effettuato il login è un agente e il suo codice viene assegnato ad ogni ordine da lui creato.

La proprietà url_ws è fondamentale per l'avvio della sincronizzazione, in quanto indica l'url verso il quale inoltrare la richiesta per il rilevamento dei dati e per l'invio degli ordini.

In fine, i flags visDisponib e visGiacenza indicano se mostrare o meno la disponibilità e la giacenza del prodotto.

Concludendo per quel che riguarda la gestione di username e password si utilizzano le shared preferences di android, visto che non è richiesta la presenza contemporanea di più account sul singolo dispositivo.

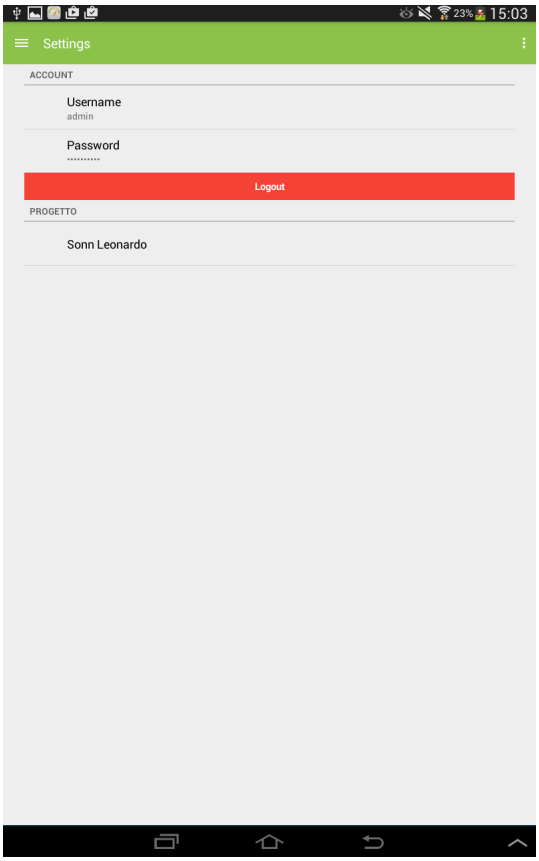

Figura 15: Settings - sezione dell'applicazione

3.3.5 Sincronizzazione dei dati

La sincronizzazione richiede un'azione preliminare di verifica dei permessi per l'esecuzione di tale operazione. La verifica dei permessi consiste in una richiesta uguale a quella di login con l'aggiunta dell'identificativo del progetto corrente selezionato. In questo modo, nel caso in cui sia scaduta la licenza di utilizzo dell'applicazione, l'agente non può procedere alla rilevazione dei dati aziendali. Altrimenti, in caso di risposta positiva da parte del license manager, l'agente o l'utente corrente può avviare la sincronizzazione.
L'aggiornamento dei dati, come già detto in fase di progettazione, può riguardare solo una certa tipologia di informazioni in base alla scelta dell'utente.
Per la sincronizzazione dei dati si utilizza una libreria Http per Android, chiamata Volley. Si tratta di una libreria molto utilizzata che comporta i seguenti benefici:

- schedulazione automatica delle richieste di rete;
- gestione concorrente di connessioni di rete;
- ampia possibilità di personalizzazione;
- possibilità di ordinamento per le richieste.

Sfruttando le possibilità di personalizzazione, si estende la classe generica per le richieste Volley, ovvero Request:

```
public class VolleyRequest<T> extends Request<T> {

    private Context mContext;
    private CallConfiguration configuration;
    private Class<T> classToParse;
    private OnResponseListener<T> respListener;
    PersistenceParameters parameters;

    public VolleyRequest(Context mContext,String url, Class
       classToParse,
                         OnResponseListener respListener ,
                            Response.ErrorListener errorListener,
                            CallConfiguration configuration,
                            PersistenceParameters parameters) {
        super(Method.GET, url, errorListener);
        this.mContext=mContext;
        this.classToParse=classToParse;
        this.respListener=respListener;
        this.configuration=configuration;
        this.parameters=parameters;
    }

    public Map<String, String> getHeaders() {
        if(configuration.getHeaders()!=null)
            return configuration.getHeaders();
        else
            return new HashMap<String, String>();
    }

    @Override
    public String getBodyContentType()
    {
        return configuration.getContentType();
    }

    @Override
    public byte[] getBody() throws AuthFailureError {
        if(configuration.getBody()!=null)
            return configuration.getBody().getBytes();
```

```
        else
            return null;
    }

    @Override
    protected void deliverResponse(T response) {

        it.apexnet.orderandroid.parsing.Response data=null;
        ArrayList dataList=null;

        if(response instanceof it.apexnet.orderandroid.parsing.
            Response) {
            data = (it.apexnet.orderandroid.parsing.Response)
                response;
            if (parameters.total_count == 0) {
                parameters.dao.deleteAll();
                parameters.total_count = data.meta.total_count;
            }
            parameters.dao.insertInTx(data.object);
        }else{
            dataList = (ArrayList) response;
            parameters.dao.deleteAll();
            parameters.dao.insertInTx(dataList);
        }

        //Verify if there are other objects to download or stop
            sync(setting total_count=-1)
        if (parameters.offset >= parameters.total_count)
            parameters.total_count=-1;

        if (respListener != null) respListener.onSuccessResponse(
            response,parameters);
    }


    @Override
    protected Response<T> parseNetworkResponse(NetworkResponse
        response) {
        try {
            Gson gson = GsonUtils.getGson();
            String json = new String(response.data,
                HttpHeaderParser.parseCharset(response.headers));

            if(classToParse==null) {
                ArrayList dataList = gson.fromJson(json, new
                    ResponseType<T>(parameters.tClass,ArrayList.
                    class));

                return (Response<T>) Response.success(dataList,
                    HttpHeaderParser.parseCacheHeaders(response,
                    false));
            }else if(classToParse!=null){
```

```
                it.apexnet.orderandroid.parsing.Response data =
                    gson.fromJson(json, new ResponseType<T>(
                    parameters.tClass, it.apexnet.orderandroid.
                    parsing.Response.class));

                return (Response<T>) Response.success(data,
                    HttpHeaderParser.parseCacheHeaders(response,
                    false));
            }

            return Response.success(null,HttpHeaderParser.
                parseCacheHeaders(response,false));
        } catch (UnsupportedEncodingException e) {
            return Response.error(new ParseError(e));
        } catch (JsonSyntaxException e) {
            return Response.error(new ParseError(e));
        }catch (Exception e){
            String m=e.getMessage();
            Log.d("ERROR_PARSING", m);
            return Response.error(new ParseError(e));
        }
    }

}
```

La classe VolleyRequest richiede i seguenti parametri:

MCONTEXT oggetto di tipo Context che individua l'activity dal quale si effettua la chiamata;

CONFIGURATION oggetto di tipo CallConfiguration che racchiude al suo interno header, body e content type della richiesta http;

CLASSTOPARSE riferimento alla classe rappresentante il json di risposta sul quale effettuare il parsing;

RESPLISTENER interfaccia OnResponseListener che gestisce le risposte alla chiamata Volley in caso di successo;

PARAMETERS oggetto di tipo PersistenceParameters contenente varie informazioni necessarie alla sincronizzazione, come la tipologia di informazioni da sincronizzare, il codice agente, l'url base, numero totale di oggetti presenti e offset di riferimento.

In particolare, la classe eredita i tre metodi principali getHeaders(), getBodyContentType() e getBody() per fornire la possibilità di personalizzare rispettivamente header, content type e body della richiesta Http.
I due metodi principali, invece, sono parseNetworkResponse(NetworkResponse response) e deliverResponse(T response). Il primo consente di effettuare il parsing della risposta ottenuta dal server ritornando l'apposito oggetto all'interno della risposta volley, Response, in caso di successo. La risposta data dal server, inserita in un json, può essere presentata in due formati diversi: il primo è dato da una semplice lista di items, mentre il secondo, oltre alla lista, presenta un oggetto di tipo Meta:

```
public class Meta {
        public int limit;
        public int maxID;
        public int offset;
        public int total_count;
}
```

L'oggetto Meta ritornato è fondamentale, soprattutto, per la prima chiamata, in quanto consente di individuare il numero totale di elementi da dover sincronizzare.
In base alla risposta ottenuta all'interno del metodo parseNetworkResponse si esegue il parsing appropriato, utilizzando la libreria Gson che permette di mappare una determinata stringa nell'oggetto Java corrispondente. In caso di una semplice lista di elementi, il parsing avviene in base a 'new ResponseType (parameters.tClass,ArrayList.class)'; la classe ResponseType permette di specificare che la risposta è caratterizzata da un arraylist di oggetti tipo parameters.tClass.
Nell'altro caso, invece, il parsing si basa su 'new ResponseType(parameters.tClass, it.apexnet.orderandroid.parsing.Response.class)'; in questo caso si specifica che la risposta da mappare è caratterizzata dall'oggetto di tipo Response che, oltre all'oggetto Meta descritto in precedenza, contiene anche la lista di oggetti di tipo parameters.tClass.
Una volta effettuato il parsing, viene eseguito il metodo deliverResponse. Al suo interno si verifica se la risposta Volley contiene un oggetto di tipo Response.class oppure ArrayList.class. Se la risposta è di tipo Response, ci si aspetta sempre

una prima chiamata per verificare il totale degli oggetti da sincronizzare e quindi, in questa situazione, si aggiorna semplicemente il numero totale di elementi eliminando quelli precedenti, altrimenti significa che è stata effettuata una delle n chiamate e quindi si procede al salvataggio sul db degli oggetti ritornati nella lista. Se invece la risposta è di tipo ArrayList, significa che non ci si trova nella situazione di chiamate incrementali, ma si tratta di una singola richiesta per un piccolo insieme di elementi (come nel caso delle valute) e quindi si sostituiscono i dati precedentemente salvati con quelli nuovi. In fine, sempre all'interno di questa funzione, si controlla se l'offset ha raggiunto il numero totale di elementi e in tal caso significa che la sincronizzazione per quella tipologia di dati è terminata: si ritorna, cosi, all'activity principale, passandogli il parametro total_count con valore -1. Il comportamento da adottare in caso di completamento di una richiesta viene integrato nel metodo onSuccessResponse dell'interfaccia OnResponseListener:

```
    @Override
    public void onSuccessResponse(Object data,
        PersistenceParameters parameters) {
         if(parameters.total_count==-1){
             parameters.progressBar.setVisibility(View.INVISIBLE);
             parameters.syncFinishImage.setVisibility(View.VISIBLE)
                 ;
             parameters.syncFinishImage.setImageDrawable(
                 getResources().getDrawable(R.drawable.sync_ok));
         }else{
             CallConfiguration configuration= new CallConfiguration
                 (SyncActivity.this);
             MyVolleyErrorListener errorListener=new
                 MyVolleyErrorListener(SyncActivity.this,parameters)
                 ;
             int tempOffset=parameters.offset;
             parameters.offset = parameters.offset + parameters.
                 limit;
             VolleyRequest request= new VolleyRequest(SyncActivity.
                 this,getUrl(parameters.baseUrl,tempOffset,
                 parameters.limit),Response.class,SyncActivity.this,
                 errorListener,configuration,parameters);
             request.setRetryPolicy(policy);
             queue.add(request);
         }

    }
```

In particolare, si verifica se il parametro total_count ha valore -1 e, in tal caso, significa che è stata completata l'ultima richiesta per la sincronizzazione di una certa tipologia di dati; altrimenti la sincronizzazione non è terminata e dev'essere avviata un ulteriore richiesta Volley per l'aggiornamento dei successivi n dati.
Come mostrato, ogni richiesta viene aggiunta ad una coda, RequestQueue di Volley, che permette di gestire i vari threads al quale assegnare le richieste in coda da eseguire. In questo modo, si sfruttano le risorse attualmente disponibili senza rischiare di eseguire troppe richieste in contemporanea.

Nella seguente figura viene mostrata la sezione di sincronizzazione con la selezione dei dati da richiedere al server o meno:

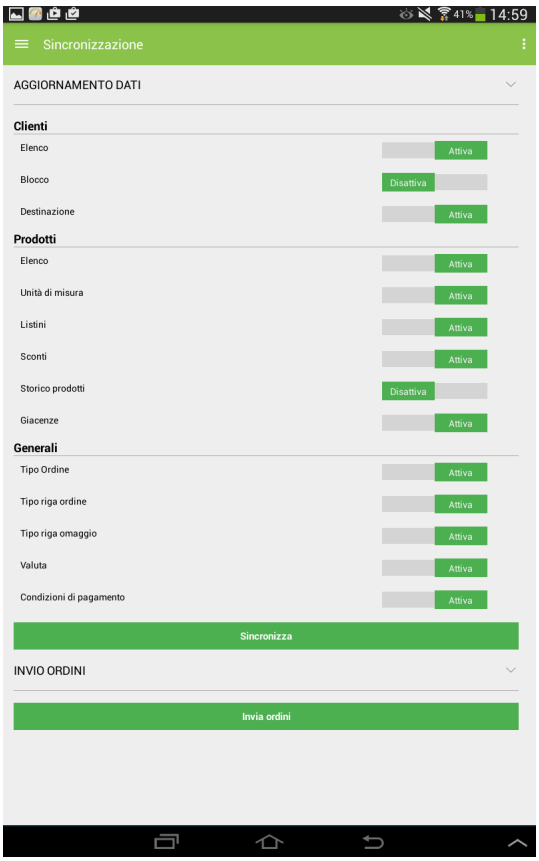

Figura 16: Sincronizzazione - sezione dell'applicazione

3.3.6 Creazione dell'ordine

CALCOLO DEL PREZZO DA LISTINO L'algoritmo del calcolo del prezzo da listino si basa su una serie di listini ricavati dall'apposita query:

```
/*Query for product lists*/
public List<ProdottoListino> getProductLists(String
    codDeposito,String codCli,String codDest,String codArt
    ,String codListino,String codFam,String codSFam,String
     codGrup,String codSGrup,String macrocat,String cat,
    String sottocat,String macroarea,String area,String
    zona,Double qta,Date data,String codValuta,int
    enabled_valute){
     QueryBuilder<ProdottoListino> builer=productlistDao.
         queryBuilder();
     List<ProdottoListino> lists=null;

     /*Equal Predicates*/
     List<Predicate> equalPredicates=loadEqualPredicates(
         codDeposito,codCli,codDest,codArt,codListino,
             macrocat,cat,sottocat,macroarea,area,zona);
     /*If value=null insert only the null predicate*/
     for(int k=0;k<equalPredicates.size();k++){
          Predicate predicate=equalPredicates.get(k);
          if(predicate.value!=null)
              builer.whereOr(isEqual(predicate.property,
                  predicate.value),predicate.property.isNull
                  ());
          else
              builer.where(predicate.property.isNull());
     }

     /*Less or Equal Predicates*/
     List<Predicate> lessOrEqualPredicates=
         loadLessOrEqualPredicates(codFam,codSFam,codGrup,
         codSGrup,qta,data);
     /*If value=null insert only the null predicate*/
     for(int k=0;k<lessOrEqualPredicates.size();k++){
          Predicate predicate=lessOrEqualPredicates.get(k);
          if(predicate.value!=null)
              builer.whereOr(isLessOrEqual(predicate.
                  property,predicate.value),predicate.
                  property.isNull());
          else
              builer.where(predicate.property.isNull());
     }

     /*Greater or Equal Predicates*/
     List<Predicate> greaterOrEqualPredicates=
         loadGreaterOrEqualPredicates(codFam,codSFam,
         codGrup,codSGrup,qta,data);
```

```
        /*If value=null insert only the null predicate*/
        for(int k=0;k<greaterOrEqualPredicates.size();k++){
            Predicate predicate=greaterOrEqualPredicates.get(k
                );
            if(predicate.value!=null)
                builer.whereOr(isGreaterOrEqual(predicate.
                    property,predicate.value),predicate.
                    property.isNull());
            else
                builer.where(predicate.property.isNull());
        }

        /*Add valute predicate if the valute management is
            active*/
        if(enabled_valute==-1)
            builer.where(isEqual(ProdottoListinoDao.Properties
                .CodValuta,codValuta));
        /*Order by priorita desc*/
        builer.orderDesc(ProdottoListinoDao.Properties.
            Priorita);
        builer.LOG_SQL=true;
        /*Execute query*/
        lists=builer.list();
        return lists;
}
```

La query, come definito in fase di progettazione, è costituita da molteplici condizioni che possono essere di uguaglianza, come nel caso del codice cliente associato all'ordine, e di appartenenza ad un range di valori, come nel caso del codice famiglia associato al prodotto. In particolare, si costruisce la query tramite l'oggetto QueryBuilder di greenDao al quale vengono aggiunti i vari predicati. Si noti che i listini vengono ritornati in ordine decrescente di priorità.

Sui listini ottenuti, si esegue successivamente l'algoritmo:

```
public ProductList calculateNetPrice(){
        for(int i=0;i<lists.size();i++)
        {
            currentIndex=i;

            /*Init new values*/
            newPrice=oldPrice;
            newDiscountImp=oldDiscountImp;
            newIncreaseAmount=oldIncreaseAmount;
            newDiscounts=oldDiscounts;
            newIncrease=oldIncrease;
            newFlagExcludeDiscount=oldFlagExcludeDiscount;
```

```
            /*Check price*/
            checkPrice(lists.get(i));

            /*Check discounts*/
            checkDiscount(lists.get(i),0);
            checkDiscount(lists.get(i),1);
            checkDiscount(lists.get(i),2);
            checkDiscount(lists.get(i),3);
            checkDiscount(lists.get(i),4);
            checkDiscount(lists.get(i),5);

            /*Check amount discount*/
            checkAmountDiscount(lists.get(i));

            /*Check increases*/
            checkIncrease(lists.get(i),0);
            checkIncrease(lists.get(i),1);

            /*Check amount increase*/
            checkAmountIncrease(lists.get(i));

            /*Check flag exclude discount*/
            checkFlagExcludeDiscounts(lists.get(i));

            oldDiscountImp=newDiscountImp;
            oldIncreaseAmount=newIncreaseAmount;
            oldPrice=newPrice;

            oldDiscounts=newDiscounts;
            oldIncrease=newIncrease;
            oldFlagExcludeDiscount=newFlagExcludeDiscount;
        }

        /*Calculate net price*/
        double netPrice=(((oldPrice * (100 - (oldDiscounts[0])
            ) / 100  * (100 - (oldDiscounts[1])) / 100 * (100
            - (oldDiscounts[2])) / 100 * (100 - (oldDiscounts
            [3])) / 100 * (100 - (oldDiscounts[4])) / 100 *
            (100 - (oldDiscounts[5])) / 100) - oldDiscountImp)
             * (100 + (oldIncrease[0])) / 100 * (100 + (
            oldIncrease[1])) / 100) + oldIncreaseAmount;
        ProductList list=new ProductList();
        list.price=oldPrice;
        list.netPrice=netPrice;
        list.discounts=oldDiscounts;
        list.amountDiscount=oldDiscountImp;
        list.increases=oldIncrease;
        list.amountIncrease=oldIncreaseAmount;
        list.flagExcludeDiscounts=oldFlagExcludeDiscount;
        return list;
    }
```

L'algoritmo consiste nell'interazione dei vari listini partendo da quelli con maggiore priorità. Ad ogni iterazione vengono aggiornate le informazioni di prezzo, dei sei sconti possibili, dello sconto totale sull'importo, di due possibili maggiorazioni sul prezzo, della maggiorazione sull'importo e di un particolare flag che indica se escludere o meno il calcolo di sconti di default per l'ordine in corso. Questo insieme di dati viene determinato in base a specifici indici impostati sul listino: ciascun valore può essere sostituito o sommato con quello corrispondente nel listino corrente. Al termine dell'iterazione si calcola il prezzo netto tenendo conto degli sconti e delle maggiorazioni determinate precedentemente.

Concludendo, viene ritornato una sorta di 'listino finale' di tipo ProductLists con informazioni sul prezzo, prezzo netto, sconti iniziali, sconto totale, maggiorazioni, maggiorazione totale e flag per il calcolo o meno di ulteriori sconti che sostituiranno gli sconti iniziali appena determinati.

CALCOLO DEGLI SCONTI In base al flag relativo agli sconti ricavato in precedenza si avvia la procedura di calcolo per proporre all'utilizzatore dell'app degli sconti di default che, in caso di autorizzazione, potranno essere modificati. Come nel caso precedente, si esegue un'apposita query che permette di ottenere una lista di righe di tipo ProdottoSconto sulla quale viene eseguito l'algoritmo.

```
/*Query for product discounts*/
public List<ProdottoSconto> getProductDiscounts(String
    codClasseArticolo,String codClasseCliente,String
    codCli,String codArt,Double qta,Date data,String
    codValuta,int enabled_valute){
     QueryBuilder<ProdottoSconto> builer=productDiscountDao
         .queryBuilder();
     List<ProdottoSconto> lists=null;

     /*Equal Predicates*/
     List<Predicate> equalPredicates=loadEqualPredicates(
         codCli,codArt,codClasseArticolo,codClasseCliente);
     /*If value=null insert only the null predicate*/
     for(int k=o;k<equalPredicates.size();k++){
          Predicate predicate=equalPredicates.get(k);
          if(predicate.value!=null)
```

```
                builer.whereOr(isEqual(predicate.property,
                    predicate.value),predicate.property.isNull
                    ());
            else
                builer.where(predicate.property.isNull());
        }

        /*Less or Equal Predicates*/
        List<Predicate> lessOrEqualPredicates=
            loadLessOrEqualPredicates(qta,data);
        /*If value=null insert only the null predicate*/
        for(int k=0;k<lessOrEqualPredicates.size();k++){
            Predicate predicate=lessOrEqualPredicates.get(k);
            if(predicate.value!=null)
                builer.whereOr(isLessOrEqual(predicate.
                    property,predicate.value),predicate.
                    property.isNull());
            else
                builer.where(predicate.property.isNull());
        }

        /*Greater or Equal Predicates*/
        List<Predicate> greaterOrEqualPredicates=
            loadGreaterOrEqualPredicates(qta,data);
        /*If value=null insert only the null predicate*/
        for(int k=0;k<greaterOrEqualPredicates.size();k++){
            Predicate predicate=greaterOrEqualPredicates.get(k
                );
            if(predicate.value!=null)
                builer.whereOr(isGreaterOrEqual(predicate.
                    property,predicate.value),predicate.
                    property.isNull());
            else
                builer.where(predicate.property.isNull());
        }

        /*Add valute predicate if the valute management is
            active*/
        if(enabled_valute==-1)
            builer.where(isEqual(ProdottoScontoDao.Properties.
                CodValuta,codValuta));
        /*Order by priorita desc*/
        builer.orderDesc(ProdottoScontoDao.Properties.Priorita
            );
        builer.LOG_SQL=true;
        /*Execute query*/
        lists=builer.list();
        return lists;
    }
```

La struttura della query è simile a quella precedente: vi sono tanti predicati

di uguaglianza e di appartenenza ad un range di valori e per ogni proprietà viene considerato anche il valore nullo. Una volta ritornata la lista di oggetti di tipo ProdottoSconto, si avvia l'algoritmo che permette di ricavare i sei sconti finali. L'algoritmo si basa sull'iterazione degli oggetti ProdottoSconto, partendo da quelli con priorità maggiore. Ognuno di questi contiene i sei sconti e degli indici associati a quest'ultimi che indicano come trattarli: ad ogni iterazione i sei possibili sconti vengono sostituiti con i valori contenuti nel ProdottoSconto corrente oppure vengono sommati i valori attuali con quelli del ProdottoSconto corrente. L' algoritmo implementato è il seguente:

```
public double[] calculateProductDiscounts(){
    if(discounts!=null)
    {
        for(int i=0;i<discounts.size();i++) {
            newFullDiscounts = oldFullDiscounts;
            for(int k=0;k<enabled_discounts;k++)
            {
                checkFullDiscounts(discounts.get(i),k);
            }
            if(newFullDiscounts!=null)
                oldFullDiscounts=newFullDiscounts;
        }
    }
    return oldFullDiscounts;
}
```

Si ricava cosi dall'algoritmo un array contenente i sei sconti.
Una volta determinati i sei sconti di default, in base al numero di sconti abilitati per il progetto, si calcolano lo sconto totale dato da quest'ultimi, lo sconto totale comprendente anche ulteriori sconti indicati dall'utente, il prezzo netto dato dal prezzo calcolato da listino al quale viene applicato lo sconto totale e l'importo dato dal prezzo netto per la quantità inserita.
Questo insieme rappresenta la configurazione di default per l'ordine corrente; ogni volta che l'utente modifica una di queste informazioni, vengono effettuati in tempo reale tutti i calcoli necessari al fine di avere dati corretti e coerenti.

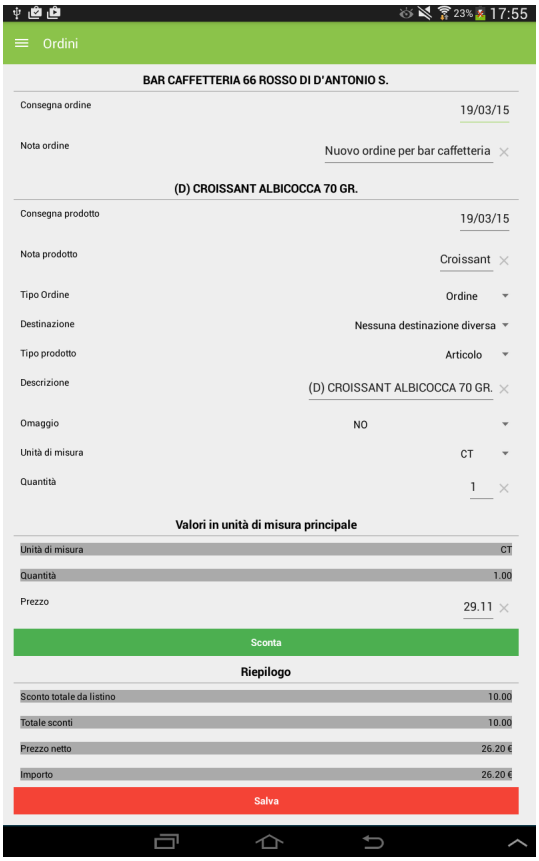

Figura 17: Creazione della riga ordine - sezione dell'applicazione

3.3.7 Invio degli ordini

L'invio degli ordini, non ancora spediti, avviene tramite una richiesta di tipo Http Put, contenente un apposito json, verso il solito app manager.

In particolare, il json è caratterizzato da una lista di testate aperte e per ognuna di esse la lista di righe ordine.

```
    public String createJsonOrder(){
        List<OrdineTemp> orderRows=null;
        List<OrderRow> orderRowsToPost=null;
        orderHeaders=query.getOpenHeaders();
        List<OrderHeader> headersToPost=new ArrayList<OrderHeader
            >();
        /*Foreach open order header, get the open order rows*/
        for(OrdineTestata header : orderHeaders) {
            orderRowsToPost=new ArrayList<OrderRow>();
            orderRows=query.getOpenRows(header);
            for(OrdineTemp row : orderRows )
            {
```

```
                OrderRow orderRow=createRowToPost(row);
                orderRowsToPost.add(orderRow);
            }
            OrderHeader orderHeader=createHeaderToPost(header,
                orderRowsToPost);
            headersToPost.add(orderHeader);
        }

        Order order=createOrderToPost(headersToPost);
        Gson gson = GsonUtils.getGson();
        /*From object,get the json string*/
        JsonElement gsonToPost=gson.toJsonTree(order);
        return gsonToPost.toString();
    }
```

La funzione sopra indicata si occupa di ricavare le testate di tutti i clienti non ancora inviate al server e tutte le righe ordine corrispondenti. La stringa inviata al server e rappresentante il json è data dall'oggetto di tipo Order:

```
public class Order {

    public int Acquisito;
    public String AppID;
    public String AppVersion;
    public String Utente;
    public String CodiceAgente;
    public String DeviceToken;
    public long IDProgetto;

    public List<OrderHeader> Testate;

}
```

All'interno di questo oggetto è contenuta una lista di oggetti OrderHeader rappresentanti le testate degli ordini:

```
public class OrderHeader {

    @SerializedName("Guid")
    public String uuid;
    @SerializedName("RagSocCliFor")
    public String ragSocClienteFittizio;
    @SerializedName("NoteOrdine")
    public String noteTestata;
    @SerializedName("CodiceCliFor")
    public String codCliForKey;
    @SerializedName("CapClifor")
    public String cap;
    @SerializedName("CellulareClifor")
    public String cell;
```

```
    @SerializedName("CittaClifor")
    public String citta;
    @SerializedName("CodCondPagClifor")
    public String codCondPag;
    @SerializedName("CodiceFiscaleClifor")
    public String codFisc;
    @SerializedName("EmailClifor")
    public String email;
    @SerializedName("FaxClifor")
    public String fax;
    @SerializedName("IndirizzoClifor")
    public String indirizzo;
    @SerializedName("NazioneClifor")
    public String nazione;
    @SerializedName("PartitaIVAClifor")
    public String partitaIva;
    @SerializedName("ProvinciaClifor")
    public String prov;
    @SerializedName("Telefono1Clifor")
    public String tel1;
    @SerializedName("Telefono2Clifor")
    public String tel2;
    @SerializedName("CodClasseScontoClifor")
    public String codClasseSconto;

    @SerializedName("Righe")
    public List<OrderRow> Righe;
}
```

Per ogni testata viene indicata una lista di oggetti di tipo OrderRow rappresentanti le righe ordine:

```
public class OrderRow {
    @SerializedName("Guid")
    public String uuid;
    @SerializedName("GuidTestata")
    public String uuidTestata;
    @SerializedName("CodiceArticolo")
    public String codArt;
    @SerializedName("CodiceCondizionePagamento")
    public String codPag;
    @SerializedName("CodiceDeperibilita")
    public String codDeperibilitaArt;
    @SerializedName("CodiceDestinazione")
    public String codDestinazione;
    @SerializedName("CodiceTipoOmaggio")
    public String codTipoOmaggio;
    @SerializedName("CodiceTipoOrdine")
    public String codTipoOrdine;
    @SerializedName("CodiceTipoRiga")
    public String codTipoRiga;
    @SerializedName("DataOrdine")
```

```
    public Date dataIns;
    @SerializedName("Descrizione")
    public String descrizione;
    @SerializedName("FattoreConversione")
    public Double fattoreConversioneRiga;
    @SerializedName("FlagDeperibilita")
    public int flagDeperibilitaCli;
    @SerializedName("NoteRigaOrdine")
    public String noteRiga;
    @SerializedName("Prezzo")
    public Double prz1;
    @SerializedName("Quantita")
    public Double qta1;
    @SerializedName("Quantita2")
    public Double qtaRiga;
    @SerializedName("Sconto1")
    public Double sconto1Riga;
    @SerializedName("Sconto2")
    public Double sconto2Riga;
    @SerializedName("Sconto3")
    public Double sconto3Riga;
    @SerializedName("Sconto4")
    public Double sconto4Riga;
    @SerializedName("Sconto5")
    public Double sconto5Riga;
    @SerializedName("Sconto6")
    public Double sconto6Riga;
    @SerializedName("TipoUM")
    public int tipoUmRiga;
    @SerializedName("UnitaMisura1")
    public String um1;
    @SerializedName("UnitaMisura2")
    public String umRiga;
    @SerializedName("CodiceValuta")
    public String codValuta;
}
```

Grazie alla libreria Gson è possibile comporre il corretto json da inserire nel body della richiesta http verso l'app manager. Se la pubblicazione degli ordini termina con successo, le testate corrispondenti vengono contrassegnate come inviate nell'apposita tabella OrdineTestata attraverso il flag 'acquisito'. Una volta pubblicati, gli ordini non possono più essere modificati, ma solo visualizzati nell'apposita sezione.

3.3.8 Ordine veloce

Per la funzionalità dell'ordine veloce, è richiesta, innanzitutto, la selezione del cliente per il quale si vogliono prendere ordini.
Una volta selezionato, si effettua una query attraverso la quale si ricava lo storico degli ordini relativo al cliente:

```
public List<ProdottoStorico> getHistorical(String customer,
    Property sortingProperty,Constants.SortingType sortingType,
    ClienteDestinazione destination){
        QueryBuilder<ProdottoStorico> builder=historicalProductDao
            .queryBuilder();
        builder.where(ProdottoStoricoDao.Properties.CodCliforKey.
            eq(customer));
        if(!destination.getCodDest().equals(Constants.
            ALL_DEFAULT_VALUE))
             builder.where(ProdottoStoricoDao.Properties.UltCodDest
                 .eq(destination.getCodDest()));

        if(sortingType==Constants.SortingType.ASC)
             builder.orderAsc(sortingProperty);
        else
             builder.orderDesc(sortingProperty);
        return builder.list();
    }
```

In particolare, la query rileva tutti gli ordini effettuati dal cliente per una specifica destinazione passata come parametro in input all'interrogazione oppure per tutte le destinazioni disponibili per il cliente corrente. Inoltre, la lista degli ordini viene ordinata in base ad una proprietà passata in input alla query: data dell'ordine, quantità dell'ordine, descrizione dell'articolo o codice dell'articolo.
Avendo la lista degli ordini effettuati in passato per il cliente, è possibile raccogliere una o più righe dello stesso ordine utilizzando quindi gli stessi dati oppure è possibile raccoglierli variando solo la quantità del prodotto desiderato. Al verificarsi di uno dei due eventi si avvia un task in background che rileva le corrette informazioni riguardanti la riga ordine selezionata ed effettua il salvataggio nell'apposita tabella del database. Particolarità di questa funzionalità è la gestione del prezzo, il quale in base ad un flag (use_historical_values) settato sul progetto corrente viene stabilito se il prezzo del nuovo ordine dev'essere lo stesso dell'ordine passato oppure se dev'essere ricalcolato dai listini dei prezzi. In quest'ultimo caso, quindi,

l'ordine è lo stesso di quello preso in passato con un prezzo aggiornato.
Anche in questo caso, come nella creazione di un nuovo ordine, alla selezione di una riga ordine dello storico si controlla se esiste una testata ordine già aperta per il cliente e, in caso positivo, la riga ordine viene aggiunta a questa testata altrimenti viene aperta una nuova testata per il cliente corrente.

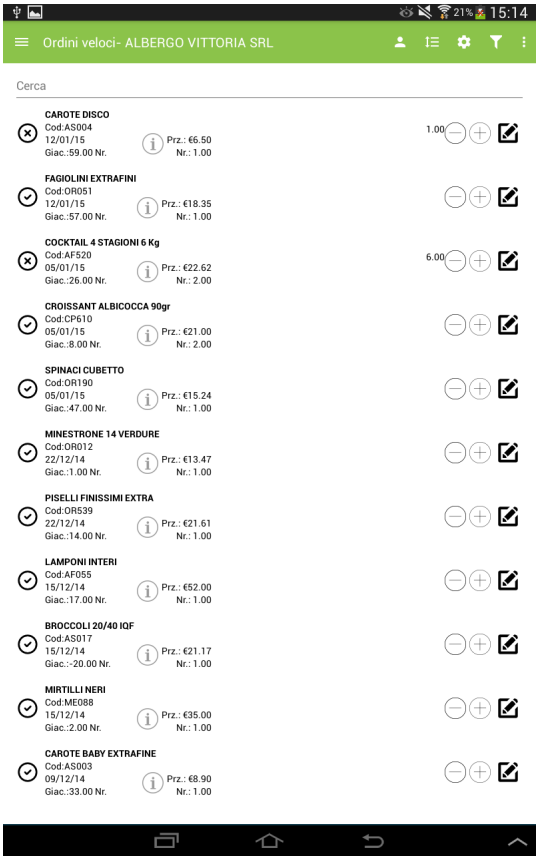

Figura 18: Ordine veloce - sezione dell'applicazione

3.3.9 Visualizzazione degli ordini

La sezione relativa agli ordini permette di mostrare sia gli ordini salvati in locale e non ancora inviati ai gestionali aziendali, che quelli già inviati. Nel primo caso, gli ordini vengono raggruppati per cliente e quindi, alla selezione di uno di questi, vengono mostrati tutti i dettagli dell'ordine corrente. Visto che l'ordine non è stato ancora inviato al gestionale, è anche possibile aprire l'ordine in modalità modifica o cancellarlo del tutto.

Nel secondo caso, invece, è possibile vedere gli ordini, raggruppandoli in diversi modi, senza poter apportare modifiche.

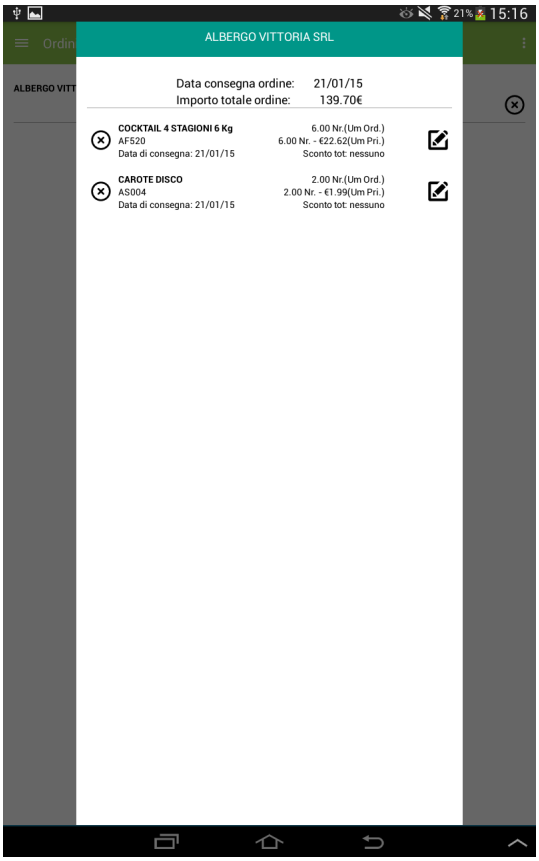

Figura 19: Dettaglio ordine - sezione dell'applicazione

3.4 SERVER

3.4.1 License Manager

Il login dell'utente, come già detto, avviene tramite una richiesta di tipo Http Post.

In particolare, il license manager si aspetta un json opportunamente formattato nel body della richiesta:

```
{
        "AppID":"String content",
        "AppVersion":"String content",
        "CodiceProgetto":"String content",
```

```
        "DeviceToken":"String content",
        "Password":"String content",
        "Utente":"String content"
}
```

La risposta, in caso di autenticazione avvenuta con successo, è data dalla lista dei progetti disponibili per l'utente:

```
[{
        "project1key1":"value1",
        "project1key2":"value2",
        "project1key3":"value3"
},
{
        "project2key1":"value1",
        "project2key2":"value2",
        "project2key3":"value3"
}]
```

3.4.2 App Manager

SINCRONIZZAZIONE DEI DATI Come già detto in fase di progettazione, l'app manager contiene i dati e i servizi necessari da interrogare durante la sincronizzazione. In particolare, si possono individuare le richieste relative a dati opzionali da sincronizzare, come nel caso dei clienti, e quelle relative a informazioni obbligatorie ai fini della presa dell'ordine, come nel caso delle valute.

Un esempio di URI corrispondente alle informazioni sui clienti è il seguente:

```
progetti/{codProgetto}/clientiPaginazione/clienti
```

In fase di richiesta è, quindi, necessario sostituire il parametro codProgetto con l'apposito valore corrispondente al progetto corrente selezionato dall'utente. Inoltre, nella richiesta possono essere definiti i seguenti parametri:

COUNT può assumere solo due valori, 0 e 1; quando ha valore 1 significa che si sta richiedendo all'app manager quanti elementi da sincronizzare ci sono, mentre in caso di valore uguale a 0, il servizio ritorna la lista di dati della tipologia richiesta;

OFFSET determina la posizione all'interno della lista dal quale deve iniziare la sincronizzazione;

LIMIT determina il numero massimo di elementi che possono essere inseriti nella risposta da parte del server;

CODAGENTE definisce il codice dell'agente che ha effettuato la richiesta.

Quindi, riassumendo, in caso di prima chiamata al servizio con count uguale a 1 viene ritornato il numero totale degli oggetti da rilevare, mentre nelle successive n chiamate, con count avente valore 0, vengono ritornati tutti gli oggetti compresi nel range [offset;offset+limit].
La risposta data dal server, nel caso di richieste associate a dati opzionali, è caratterizzata da un json col seguente formato:

```
{
        "meta":{
                "limit":12678967.543233,
                "maxID":12678967.543233,
                "offset":12678967.543233,
                "total_count":12678967.543233
        },
        "object":[{
                "item1key1":"value1",
                "item1key2":"value2",
                "item1key3":"value3"
        },
        {
                "item2key1":"value1",
                "item2key2":"value2",
                "item2key3":"value3"
        }]
}
```

Invece, nel caso di dati obbligatori per la raccolta degli ordini, si effettua un'unica richiesta per il reperimento dei dati e non è quindi necessario utilizzare i parametri definiti in precedenza. Ad esempio, per le valute:

```
progetti/{codProgetto}/moduli/valute
```

In questo caso la risposta differisce leggermente da quella definita in precedenza:

```
[    {
                "item1key1":"value1",
```

```
                "item1key2":"value2",
                "item1key3":"value3"
        },
        {
                "item2key1":"value1",
                "item2key2":"value2",
                "item2key3":"value3"
    }
]
```

INVIO DEGLI ORDINI Per quel riguarda la pubblicazione di ordini, la richiesta al servizio dev'essere nel seguente formato:

```
progetti/{codProgetto}/import_ordine
```

L'app manager, inoltre, si aspetta il json correttamente formattato nel body della richiesta:

```
{
        "Acquisito":12678967.543233,
        "AppID":"String content",
        "AppVersion":"String content",
        "Utente":"String content",
        "CodiceAgente":"String content",
        "DeviceToken":"String content",
        "IDProgetto":12678967.543233,
        "Testate":[{
                "Guid":"String content",
                "key1":"value1",
                "key2":"value2",
                "key3":"value3",
                "Righe":[{
                        "Guid":"String content",
                        "GuidTestata":"String content",
                        "key1":"value1",
                        "key2":"value2",
                        "key3":"value3",
                }]
        }]
}
```

La risposta dell'app manager, in caso di pubblicazione dell'ordine avvenuta con successo, è costituita dal semplice attributo numerico 'Acquisito' con valore -1:

```
{
        "Acquisito":12678967.543233
}
```

SVILUPPI FUTURI

Il progetto si poneva come primo obiettivo quello di realizzare una completa raccolta dell'ordine con tutte le funzionalità che ne derivano.
Come già detto, in fase di progettazione, l'obiettivo più ambizioso è quello di realizzare un vero e proprio gestionale. In parte, questo obiettivo è già stato raggiunto, ma considerata la consistenza di gestionali aziendali risulta molto semplice individuare continuamente nuovi moduli da integrare nell'app.
Un primo esempio è quello dell'inserimento di nuovi clienti: infatti l'agente, partecipando a fiere ed eventi di vario genere, ha la necessità di salvare informazioni riguardo a nuovi clienti. In questo caso, la logica è molto simile a quella utilizzata nella raccolta dell'ordine, ovvero è sempre necessario salvare dati, anche in caso di connessione di rete non disponibile, e inviare quest'ultimi ai gestionali.
Sempre per quel che riguarda i clienti, sono già state predisposte delle informazioni sulle località nelle quali sono collocati: in questo modo si può implementare una mappa all'interno di una nuova sezione dell'applicazione che dia all'utente la possibilità di sapere facilmente dove raggiungere il cliente desiderato.
Quindi, come visto, ci si può divertire nell'aggiungere sezioni o moduli facendo sempre attenzione a mantenere un'applicazione user-friendly: fattore determinante al fine di garantire un semplice utilizzo dell'applicazione durante lo svolgimento della propria attività.
Inoltre, spesso, sono direttamente i vari utenti, che utilizzano l'applicazione, a richiedere delle proprie specializzazioni che possono essere di tipo funzionale, informativo e grafico.
In fine, il compito fondamentale è quello di diffondere l'applicazione presentandola a molteplici ditte e agenti, mostrandone i vantaggi e la veloce integrazione con qualsiasi tipo di gestionale.

CONCLUSIONI

Il progetto realizzato, come detto, si colloca all'interno dell'attività di Business Process Reengineering.
Per un'azienda risulta fondamentale discutere continuamente sul modo di operare e di interagire con la società. A tal proposito viene adottato un ciclo di pianificazione e controllo che permette di definire delle strategie in termini di servizio e tecnologie, degli obiettivi e degli interventi da attuare secondo una determinata priorità.
Uno dei fattori portanti all'interno di questi interventi è sicuramente l'innovazione tecnologica. Negli ultimi anni, in particolare, questa ha raggiunto risultati eccellenti portando all'affermazione di reti di connessione ad alta velocità, del cloud e, soprattutto, del mobile. Quest'ultimo rappresenta una vera e propria rivoluzione che ha cambiato i comportamenti dell'uomo, influenzando qualsiasi sfera della quotidianità tra cui il lavoro.
Infatti, gli smart devices, insieme a cloud e social, grazie alle loro caratteristiche di portabilità, accessibilità e adattabilità hanno permesso di attivare nuovi modelli di lavoro orientati allo Smart Working. In particolare, questi contribuiscono alla flessibilità e all'accessibilità nell'utilizzo di applicazioni e servizi, permettendo di accedere anche ad applicazioni professionali in qualunque momento, luogo e attraverso qualsiasi dispositivo, liberando le persone dalla necessità di una postazione fissa.
Concludendo, il Mobile ha permesso di raggiungere traguardi impensabili fino a poco tempo fa e, negli anni avvenire, potrebbe introdurre nuove possibilità: spetta quindi all'uomo capire come fruttarle al meglio.

RINGRAZIAMENTI

Desidero ringraziare tutti quelli che, in qualche modo, mi hanno supportato nella realizzazione del progetto e nella stesura della tesi. In particolare, ringrazio il Dott. Mirko Ravaioli, la Prof.ssa Antonella Carbonaro e l'azienda Apex-net Srl per avermi dato la possibilità di svolgere l'attività presso la loro struttura.

Inoltre, ringrazio i miei amici per essere stati al mio fianco durante gli ultimi anni di studio e per avermi fatto trovare il sorriso anche in momenti più delicati.

Ringrazio tutti i miei parenti per avermi sempre sostenuto e, soprattutto, per aver creduto in me più di quanto lo facessi io. Un pensiero speciale va anche a miei cari che non ci son più, ma che porto sempre con me, mio nonno Oreste e mia nonna Rosa.

In fine, il ringraziamento più importante va alla mia famiglia : Massimo, Rosalba e mia sorella Ylenia. Li ringrazio per il supporto economico, ma, soprattutto, perché in loro ho trovato la forza per arrivare fino in fondo al mio percorso di studi.

BIBLIOGRAFIA

Android 5 vs. iOS 8 vs. Windows Phone 8.1: Which smartphone OS is best? 2014 , http://www.digitaltrends.com/.

Android and iOS Continue to Dominate the Worldwide Smartphone Market with Android Shipments Just Shy of 800 Million in 2013, According to IDC 2014 , http://www.idc.com/.

Android Developers 2015 , http://developer.android.com/index.html.

Balducci, Daniele
2007 *Agenti e rappresentanti di commercio*, a cura di FAG Milano.

Brognara, Roberto
2014 *Rivoluzione mobile. I cambiamenti sociali e di marketing introdotti dalle tecnologie mobili*, a cura di Franco Angeli.

Business process re-engineering 2009 , http://www.economist.com/.

Cos'è lo smartworking 2014 , http://www.wired.it/.

Gartner Says Smartphone Sales Grew 46.5 Percent in Second Quarter of 2013 and Exceeded Feature Phone Sales for First Time 2013 , http://www.gartner.com/.

Giampio Bracchi Chiara Francalanci, Gianmario Motta
2010 *Sistemi informativi d'impresa*, a cura di McGraw-Hill Companies.

greenDAO - Android ORM for SQLite 2015 , http://greendao-orm.com/.

Jakob Iversen, Michael Eierman
2014 *Learning Mobile App Development*, Addison-Wesley.

La mobilità della comunicazione 2010 , http://www.technologyreview.it/.

Shelton, Ted
2013 *Business Models for the Social Mobile Cloud*, John Wiley Sons, Inc.

SQLite 2015 , http://www.sqlite.org/.

The rebirth of re-engineering 2012 , http://www.computerworld.com/.