UNIVERSITÀ DEGLI STUDI DI ROMA
TOR VERGATA
FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

A.A. 2013/2014

Tesi di Laurea triennale

USO DI TAG NFC E/O QR-CODE IN AMBIENTE MOBILE PER LA GESTIONE DI EVENTI BASATI SU TURNI.

RELATORE

Massimo Regoli

CANDIDATO

Gabriele Belli

Alla mia famiglia per il sostegno

e l'incoraggiamento dimostratomi.

Indice

Capitolo 1

Introduzione

1.1 Panoramica

Rendere tutto più facile e veloce, senza imbattersi nelle interminabili file allo sportello o in lunghe telefonate ai centralini è l'idea cardine che mi ha portato allo sviluppo della piattaforma *Prenota Facile*.
La semplicità di utilizzo dell'applicazione e le svariate modalità di prenotazione degli eventi basati su turni fanno sì che questo sistema possa essere utilizzato da una vasta categoria di persone, che va da coloro che utilizzano uno smartphone abitualmente, a chi è restio nel servirsi di questi.
Mentre i primi apprezzeranno il fatto che grazie a *Prenota Facile* sarà possibile prenotare il proprio evento premendo un pulsante, scansionando un *QR–Code* o avvicinando un *tag NFC* (operazioni con un costo temporale irrisorio), le seconde potranno utilizzare il supporto vocale: basterà semplicemente pronunciare il servizio di cui vogliamo usufruire per visualizzare gli appuntamenti disponibili.
La prenotazione potrà avvenire stando comodamente nella propria abitazione o nel luogo in cui l'ente eroga il servizio, sarà inoltre possibile cancellare l'evento richiesto in modo semplice e veloce.
A tutto questo poi va aggiunto un design molto intuitivo che segue le linee guida di un *elder application* (pulsanti e scritte molto grandi e applicazione simile ad un normale telecomando per TV), così da poter essere utilizzata da persone anziane e/o con problemi di vista.
Proprio a queste tipologie di persone che il *Voice Helper*, funzionalità dell'applicazione, viene incontro, dando loro informazioni leggibili e/o udibili sul servizio che vogliono

prenotare, come ad esempio: via, numero ufficio e personale con il quale andranno ad interagire.

1.2 Obiettivi

Gli obiettivi che il sistema *Prenota Facile* vuole raggiungere sono molto ambiziosi e sopratutto non sono fini all'applicazione stessa.
La piattaforma vuole infatti:

- Avvicinare al mobile e più in generale alle tecnologie digitali la fetta di pubblico ancora dubbiosa sui benefici che questa può portare anche a livello personale.
- Rendere semplice meccanismi quotidiani come la prenotazione di eventi, necessari a svolgere molti adempimenti pubblici o della nostra vita privata.
- Ridurre drasticamente il tempo impiegato per interfacciarsi verso coloro che erogano un servizio: sportelli, centralini, etc.
  Viene così ridotto o quasi annullato il tempo che si perde in inutili file, tempo che può essere utilizzato per altre circostanze.
- Ottimizzare il servizio degli enti che decidono di usufruire di *Prenota Facile* fornendo loro un feedback che gli permetta di venire a conoscenza di quale servizio viene meno o più utilizzato dalla clientela, così da prendere delle decisioni che permetta loro di crescere o investire al meglio le loro risorse.
- Poter essere utilizzato da tutti grazie alla sua semplicità e facilità di utilizzo.
  La semplicità non è sinonimo di superficialità in quanto qualcosa di semplice, se utile, può rendere piacevole l'utilizzo del sistema.

1.3 Contesti di utilizzo

Prenota Facile è un sistema di ampio respiro, che può adeguarsi a molti contesti della realtà che ci circonda.
Attraverso una doverosa personalizzazione del database dei contenuti, l'applicazione potrà essere adattata da qualsiasi servizio basato su turni, come ad esempio: prenotazioni mediche, appuntamenti dal parrucchiere o dal dentista, ritiro di merci, consulenze in uffici di pubblica amministrazione etc, etc.

Prenota Facile potrà entrare a far parte della nostra quotidianità, con vantaggi sia per chi l'utilizza sia per chi lo mette a disposizione come servizio aggiuntivo.
Ritornando al contesto di utilizzo, l'esempio della prenotazione medica è il case-study che ho deciso di adottare per mostrare la semplicità della mia piattaforma, poiché mi è sembrato il più immediato per la presentazione dell'idea e sufficientemente appetibile per un ente pubblico o privato: il suo utilizzo e le sue funzionalità saranno descritte nei prossimi capitoli.

Il lavoro di tesi è articolato come segue: il primo capitolo introduce le tecnologie utilizzate; il progetto dell'applicazione e l'implementazione sono descritte nel secondo capitolo; il case study occuperà tutto il terzo ed infine il porting e le conclusioni del lavoro svolto e degli sviluppi futuri sono riportate rispettivamente nel quarto e quinto capitolo. Il lavoro verrà concluso con la bibliografia.

Capitolo 2

Tecnologie utilizzate

In questo capitolo saranno descritte le principali tecnologie utilizzate per la realizzazione dell'applicazione client/server.

2.1 Android

Android non è un linguaggio di programmazione, ma un vero e proprio insieme di strumenti e librerie per la realizzazione di applicazioni mobili. Ha come obiettivo quello di fornire tutto ciò che un operatore, un vendor di dispositivi o uno sviluppatore ha bisogno per raggiungere i propri obiettivi.
Android ha la fondamentale caratteristica di essere open, dove il termine assume diversi significati. È open in quanto utilizza tecnologie open, prima fra tutte il kernel di Linux, nella versione 2.6; ed in quanto le librerie e le API che sono state utilizzate per la sua realizzazione sono esattamente le stesse che andremo ad usare per creare le nostre applicazioni.
Quasi tutti i componenti di Android potranno essere rimpiazzati dai nostri, così da non avere limiti nella personalizzazione dell'ambiente, se non quelli legati ad aspetti di sicurezza nell'utilizzo, per esempio, delle funzionalità del telefono. Android è open in quanto il suo codice è open–source e consultabile da chiunque intenda contribuire a migliorarlo, voglia creare una parte della documentazione o, semplicemente, voglia scoprirne il funzionamento.
La licenza scelta dalla Open Handset Alliance è la Open Source Apache License 2.0, la quale permette ai diversi vendor di costruire su Android le proprie estensioni anche proprietarie senza legami che ne potrebbero limitare l'utilizzo.

Questo significa che un produttore non deve pagare alcuna royalty per adottare Android sui propri dispositivi.
Dal 2008, data della prima uscita su smartphone, ad oggi, Android è migliorato moltissimo e ha subito grandi variazioni dal punto di vista grafico, funzionale e delle prestazioni. Si sono susseguite 49 versioni (contando anche le release minori) e 21 livelli di API, Application User Interface. Le API sono un insieme di funzioni disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per lo svolgimento di un determinato compito all'interno di un certo programma. Le release maggiori, cioè che portano grandi cambiamenti, passano da un numero ad un altro (es. da 3 a 4) mentre le release minori, correzioni di piccoli bug e piccole novità, mantengono quasi lo stesso numero di versione (es. da 4.0 a 4.1). Le versioni di Android prima della 1.0 non avevano un nome identificativo ma solamente un numero di riconoscimento come m3-rc20a e m5-rc15.
Per poter essere ricordate più facilmente anche dagli utenti, si decise di dare a queste due versioni dei nomi che potessero ricordare un robottino e furono rispettivamente Astro Boy e Bender. Per le versioni seguenti gli sviluppatori decisero che avrebbero invece utilizzato dei nomi di dolcetti, senza un particolare motivo, ma in ordine alfabetico. La prima di queste versioni è Cupcake, la prima che esce anche in Italia, introduce delle novità che con l'evolversi del sistema operativo verranno mantenute, migliorate e considerate sempre più importanti, come i Widget e la rotazione dello schermo. Donut e Eclair non sono particolarmente interessanti dato che ottimizzano solo in parte le prestazioni di Android ed infatti non sono per nulla diffuse.
Froyo introduce importantissime novità nel kernel2 grazie alla Dalvik Virtual Machine (di cui parleremo più avanti) e al JIT (Just in Time compiler). Gingerbread porta solamente delle piccole migliorie, mentre con HoneyComb Google si sposta anche nel settore tablet, introducendo la grafica Holo che sarà alla base delle successive versioni per smartphone come Ice Cream Sandwich e Jelly Bean. Quest'ultima versione è decisamente migliore delle precedenti grazie all'implementazione del Project Butter, un'iniziativa di Google per rendere il sistema più fluido.
KitKat non stravolge il sistema operativo. Ciò nonostante le novità non mancano, a partire da un'ottimizzazione del codice e in particolare dell'utilizzo che fanno le applicazioni della memoria, in modo da incrementare le prestazioni generali e rendere la piattaforma compatibile anche con i dispositivi di fascia medio-bassa.
La "L release" di Android cambia runtime system, sostituendo Dalvik con il nuovo

ART. In termini pratici, questo si traduce in performance migliorate. È forse la novità più attesa dagli sviluppatori, che in questo modo possono sfruttare appieno le potenzialità hardware messe a disposizione da processori e chip grafici, ottimizzando il codice delle applicazioni. A questo scopo Google ha dichiarato di essere al lavoro con chipmaker come NVIDIA, Qualcomm, Arm e Imagination Technologies.
Il sistema operativo Android è basato su kernel Linux e consiste in una struttura formata da vari livelli o layer, ognuno dei quali fornisce al livello superiore un'astrazione del sistema sottostante:

- **Linux Kernel**
- **Libraries**
- **Application Framework**
- **Applications**

2.1.1 Linux Kernel

Il sistema è basato su Kernel Linux, inizialmente 2.6 poi 3.X, che costituisce il livello di astrazione di tutto l'hardware sottostante: include cioè i driver per la gestione del WiFi, Bluetooth, GPS, fotocamera, touchscreen, audio, USB driver, etc. Ogni produttore di smartphone dovrà modificare in primo luogo questa parte per renderlo compatibile con l'hardware scelto.
Anche la creazione di una custom ROM spesso ha bisogno di modifiche a questo livello, in quanto spesso vengono effettuati dei porting delle ultime versioni di Android su dispositivi che non riceverebbero più aggiornamenti ufficiali. L'astrazione hardware permette ai livelli superiori e agli sviluppatori una programmazione ad alto livello senza preoccuparsi del tipo di hardware che monta il telefono.
A differenza di un kernel Linux standard, in quello di Android sono stati aggiunti ulteriori moduli per la miglior adattabilità a dispositivi mobili, come:

- Binder (IPC) Driver: driver dedicato che permette la comunicazione tra processi con un costo computazionale minore e quindi un più basso consumo di batteria.
- Low Memory Killer: modulo che si occupa di terminare i processi in modo da liberare spazio nella memoria centrale per soddisfare le richieste di altri processi. La terminazione avviene sulla base di un sistema di ranking che assegna dei

punteggi in base all'importanza dei processi. Il processo init non può essere terminato.

- Android Debug Bridge: strumento che permette di gestire in maniera versatile un'istanza dell'emulatore o di un dispositivo reale.
- RAM Console e Log devices: modulo per agevolare il debugging. Android fornisce infatti la possibilità di memorizzare messaggi di log generati dal kernel in un buffer RAM, in modo da poter essere osservati dagli sviluppatori per trovare eventuali bug.
- Power Management: sezione progettata per permettere alla CPU di adattare il proprio funzionamento per non consumare energia se nessuna applicazione o servizio ne fa richiesta.
- Ashmem: sistema di memoria condiviso anonimo (Anonimous Shared Memory) che definisce interfacce che permettono ai processi di condividere zone di memoria attraverso un nome.

2.1.2 Libraries

Il livello superiore riguarda le librerie C/C++ che vengono utilizzate da vari componenti del sistema. Tra esse troviamo:

- Il Surface Manager: modulo che gestisce le View, cioè i componenti di un'interfaccia grafica.
- Il Media Framework: si occupa dei Codec per l'acquisizione e riproduzione di contenuti audio e video.
- SQLite: il Database Management System (DBMS) utilizzato da Android. E' un DBMS relazionale piccolo ed efficiente messo a disposizione dello sviluppatore per la memorizzazione dei dati nelle varie applicazioni sviluppate.
- FreeType: un motore di rendering dei font.
- LibWebCore: un browser-engine basato su WebKit, open source, che può essere integrato in qualunque applicazione sotto forma di finestra browser.
- SGL e OpenGL ES: librerie per gestire rispettivamente grafica 2D e 3D.

- SSL: libreria per il Secure Socket Layer.
- LibC: implementazione della libreria di sistema standard C, ottimizzata per dispositivi basati su versioni embedded di Linux.

Di grande rilievo è l'ambiente di Runtime, costituito dalla Core Library e la Dalvik Virtual Machine: insieme costituiscono l'ambiente di sviluppo per Android. Le Core Libraries includono buona parte delle funzionalità fornite dalle librerie standard di Java a cui sono state aggiunte librerie specifiche di Android.
La Dalvik Virtual Machine (DVM) è invece un'evoluzione della Java Virtual Machine sviluppata da Google in cooperazione con altri marchi noti come ad esempio nVidia e Intel, in modo da lavorare su dispositivi poco performanti come i cellulari. Bisogna tuttavia sottolineare che con gli ultimi modelli di smartphone si sono raggiunte delle prestazioni veramente elevate (processori superiori a 2 GHz e memorie RAM da 2 GB), paragonabili a dei notebook di fascia bassa.
Vediamo più nel dettaglio come è organizzata la Dalvik Virtual Machine. Per poterlo capire dobbiamo dapprima chiarire il concetto di macchina virtuale: una VM è un software che può essere un sistema operativo, un emulatore o una virtualizzazione hardware completa. Ci possono essere due tipi di macchine virtuali: System VM e Process VM. La prima supporta l'esecuzione di un sistema operativo completo mentre la seconda supporta solo l'esecuzione di processi singoli. Rappresentandole con dei layer si troverebbe che la System VM è al di sopra della Process VM, è utile inoltre chiarire la differenza tra Stack Based VM e RegisteredBased VM. La prima è orientata, come dice il nome, all'uso dello stack mentre la seconda è orientata all'uso dei registri. Se inizialmente, per facilità di implementazione, si utilizzava la Stack Based, ora invece si preferisce la Registered-Based in quanto riduce di molto il numero di istruzioni.
La DVM è una Process Virtual Machine che adotta proprio l'approccio Registered Based, a differenza della JVM che utilizza l'approccio più vecchio, quello Stack Based. E' quindi veloce in esecuzione ma leggermente più lenta nella creazione del bytecode. Ad ogni applicazione viene associato un processo che gira all'interno di una DVM ad esso dedicata, ed il sistema gestisce più macchine virtuali contemporaneamente per rendere possibile il multitasking. Il bytecode costituisce l'insieme delle operazioni che la macchina dovrà eseguire e viene scritto una sola volta. Quello della DVM deriva da quello della JVM: viene creato da un file .class un file .dex (Dalvik EXecutable), che non è altro che una modifica del bytecode della JVM a cui vengono tolte le parti ripetute più volte. Da alcune versioni di Android inoltre (in particolare dalla 2.2), la

DVM implementa un Just in Time Compiler (JIT), che attraverso il riconoscimento di alcune parti di codice Java, esegue una traduzione in parti di codice nativo C o C++ rendendo l'esecuzione molto più veloce. Riepilogando, i motivi per cui viene adottata la DVM sono: minor spazio occupato dal bytecode, maggior velocità in esecuzione rispetto ad una Stack Based VM.
Sempre relativamente al kernel, quello di Linux di Android è un sistema multiutente nel quale ogni applicazione è un utente differente. Il sistema infatti crea uno user ID distinto per ogni applicazione ed imposta i permessi dei file dell'applicazione stessa in modo che solo quell'ID possa accedervi. Inoltre, ogni applicazione sul telefono viene lanciata in un processo Linux a se stante all'interno della propria istanza della JVM: questa architettura a sandbox garantisce la stabilità del telefono nel caso in cui qualche applicazione crei problemi.
Tutto questo non limita però la possibilità di interazione tra i processi: permette anzi loro di condividere lo stesso user ID e la stessa JVM in modo da preservare la coerenza delle risorse di sistema.

2.1.3 Application Framework

In questo livello è possibile rintracciare i gestori e le applicazioni di base del sistema. In questo modo gli sviluppatori possono concentrarsi nella risoluzione di problemi non ancora affrontati, avendo sempre a propria disposizione il lavoro già svolto da altri. Tra i vari gestori presenti troviamo:

- Activity Manager: gestisce tutto il ciclo di vita delle Activity. Le Activity sono entità associate ad una schermata dell'applicazione. Il compito dell'Activity Manager è quindi quello di gestire le varie Activity sul display del terminale e di organizzarle in uno stack in base all'ordine di visualizzazione sullo schermo.
- Content Providers: gestiscono la condivisione di informazioni tra i vari processi attivi.
- Window Manager: gestisce le finestre relative a differenti applicazioni.
- Il Telephony Manager: gestisce le funzioni base del telefono quali chiamate ed SMS.
- Resource Manager: gestisce tutte le informazioni relative ad una applicazione (file di configurazione, file di definizione dei layout, immagini utilizzate, ecc).

- Package Manager: gestisce i processi di installazione e rimozione delle applicazioni dal sistema.
- Location Managaer: mette a disposizione dello sviluppatore una serie di API che si occupano della localizzazione (tramite GPS, rete cellulare o WiFi).
- System View: gestisce l'insieme degli elementi grafici utilizzati nella costruzione dell'interfaccia verso l'utente (bottoni, griglie, text boxes, ecc. . .).
- Notification Manager: gestisce le informazioni di notifica tra il dispositivo e l'utente. Le notifiche possono consistere in vari eventi: la comparsa di un'icona nella barra di notifica, l'accensione del LED del dispositivo, l'attivazione della retroilluminazione del display, la riproduzione di suoni o vibrazioni.

2.1.4 Applications

Al livello più alto risiedono le applicazioni utente. Le funzionalità base del sistema, come per esempio il telefono, il calendario, la rubrica, non sono altro che applicazioni scritte in Java che girano ognuna nella propria DVM.
E' bene notare che Android non differenzia le applicazioni di terze parti da quelle già incluse di default, infatti garantisce gli stessi privilegi a entrambe le categorie.

2.2 NFC

La Near Field Communication (NFC), nota anche come comunicazione contactless o di prossimità, è una tecnologia grazie alla quale è possibile connettere con onde a corto raggio (ossia in modalità wireless) due dispositivi senza creare alcun contatto fisico fra gli stessi. Il suo funzionamento avverrebbe entro un raggio di circa 10 cm di distanza, ma solitamente è limitato a molto meno per ragioni di sicurezza (4 cm o meno). Questo standard permette una comunicazione bidirezionale, ed entrambi i device coinvolti possono inviare e ricevere informazioni.
Si tratta di una tecnologia relativamente recente, naturale evoluzione della Radio Frequency Identification (RFID), brevetto registrato nel 1983 da Charles Walton.
Il grande impulso verso lo sviluppo della NFC si è avuto nel 2004 quando i tre colossi dell'hi-tech Nokia, Philips e Sony crearono il Near Field Communication Forum, un'associazione di aziende che promuove la standardizzazione e l'implementazione di

questa tecnologia.
L'NFC può essere inserito nei dispositivi tramite un chip integrato oppure tramite l'uso di una speciale scheda esterna che sfrutta le porte microSD. La comunicazione a breve distanza sta avendo un forte impatto sulle abitudini quotidiane e in un brevissimo futuro saranno molte le cose che potremo fare utilizzando un normale smartphone dotato di tecnologia NFC:

- Scaricamento e pagamento su dispositivi portatili NFC di giochi, file audio digitali, video, software.
- Scaricamento da un PC su di un dispositivo portatile, della prenotazione o acquisto di una permanenza in albergo, ingressi a cinema, viaggio in treno o aereo e accesso al servizio comperato mediante il dispositivo stesso avvicinandolo o toccando il chiosco elettronico in albergo, al gate di ingresso o di partenza.
- Scaricamento da un chiosco elettronico mediante scansione o contatto di informazioni addizionali per ingressi a teatri, stadi, titolo di viaggio con mezzi urbani e accesso al servizio mediante il dispositivo stesso anche sui mezzi di trasporto urbano.
- Trasferimento e visualizzazione di fotografie da una macchina fotografica o telefono cellulare NFC a un chiosco elettronico, televisione, computer per la visione o la stampa.
- Trasferimento facilitato di file o messa in rete fra sistemi wireless.
- Uso della tecnologia NFC per i sistemi di bigliettazione elettronica.

2.2.1 Tag NFC

I Tag NFC sono generalmente piccoli adesivi contenenti un chip NFC. Questi vengono programmati tramite un'apposita applicazione per smartphone e la loro capacità è variabile. In genere, un tag NFC è in grado di contenere una stringa, il link ad un indirizzo web o ad un numero di telefono, ma può anche essere impostato per far eseguire determinate azioni allo smartphone (attivare il WiFi, abbassare la suoneria). Nella maggior parte dei casi, i dati presenti sui Tag NFC possono essere riscritti diverse volte, finché il proprietario non decida di "chiuderla", impedendo così la modifica ulteriore dei dati presenti sull'adesivo. A differenza dei codici QR, per leggere un Tag

NFC non è necessario aprire un'app scanner sullo smartphone e le informazioni sono trasmesse istantaneamente senza la necessità di un tempo di interpretazione. Questo li rende adatti anche a circuti di pagamento e transazioni bancarie.

2.3 QR-code

Un codice QR (in inglese QR Code, abbreviazione di Quick Response Code) è un codice a barre bidimensionale, ossia a matrice, composto da moduli neri disposti all'interno di uno schema di forma quadrata. Viene impiegato per memorizzare informazioni generalmente destinate a essere lette tramite un telefono cellulare o uno smartphone. In un solo crittogramma sono contenuti 7.089 caratteri numerici o 4.296 alfanumerici.
Il nome "QR" è l'abbreviazione dell'inglese "Quick Response" ("risposta rapida"), in virtù del fatto che il codice fu sviluppato per permettere una rapida decodifica del suo contenuto. I codici QR possono contenere sia indirizzi internet, che testi, numeri di telefono, o sms. Sono leggibili da qualsiasi telefono cellulare e smartphone munito di un apposito programma di lettura (lettore di codici QR, o in inglese QR reader). Dato che Denso Wave ha reso pubblico l'uso della tecnologia QR con licenza libera, su internet è possibile trovare programmi gratuiti sia per la lettura (decodifica) che per la scrittura (codifica) dei codici QR.
Dalla fine degli anni 2000 i programmi di lettura dei codici QR sono spesso già installati nei telefonini dai relativi produttori. Esistono comunque molti siti web che offrono i lettori per cellulari, generalmente senza costi. Per leggere un codice QR è sufficiente inquadrarlo con la fotocamera del cellulare dopo aver aperto il lettore. Per quel che riguarda la scrittura, esistono diversi siti che consentono la libera produzione di codici QR.

2.3.1 Rilevazione e correzione d'errore

I codici QR possono memorizzare, come detto in precedenza, fino ad un massimo di 4.296 caratteri alfanumerici, 7.089 caratteri numerici. Nei codici QR è utilizzato il codice Reed-Solomon per la rilevazione e correzione d'errore: nel caso in cui il QR fosse in parte danneggiato, per esempio da macchie o graffi sul supporto cartaceo, l'applicazione Reed-Solomon permette di ricostruire i dati persi, ripristinando, durante la decodifica, fino al 30% delle informazioni codificate.

Capacità di memorizzazione dei dati:

- Solo numerico: max 7. 089 caratteri.
- Alfanumerico: max 4. 296 caratteri.
- Binario (8 bit): max 2. 953 byte.
- Kanji/Kana: max 1. 817 caratteri.

Capacità di correzione degli errori:

- Livello L: circa il 7 % delle parole in codice può essere ripristinato.
- Livello M: circa il 15 % può essere ripristinato.
- Livello Q: circa il 25 % può essere ripristinato.
- Livello H: circa il 30 % può essere ripristinato.

2.4 MySql

"MySQL is a very fast, multi-threaded, multi-user and robust SQL (Structured Query Language) database server" (MySQL è un database SQL server molto veloce, multi-processo, multi-utente e robusto). Questa è la definizione di MySQL data dal team di sviluppo nella documentazione.
Analizziamo in dettaglio i termini che la compongono:

- Database: è un archivio per gestire dati, strutturato in modo semplice e regolare. I dati sono organizzati in tabelle. Ogni tabella è organizzata in righe e colonne. Le righe possono contenere diverse parti di informazione.
- SQL (Structured Query Language): è un linguaggio standard che permette di immagazzinare, aggiornare e accedere alle informazioni contenute in un database, di qualunque tipo esse siano (testo, immagini, file in generale). MySQL è basato su SQL.
- Multi processo: significa che può rispondere a diverse richieste contemporaneamente.

- Multi utente: significa che può essere utilizzato contemporaneamente da diversi utenti.
- Robusto e veloce: significa che è in grado di gestire grandi quantità di dati con risorse hardware limitate.

MySql è il database relazionale open source più diffuso al mondo. Originariamente sviluppato dalla società svedese TcX per uso interno, MySql costituisce oggi la soluzione ottimale per le seguenti ragioni:

- **Scalabilità e flessibilità** Mysql offre il massimo in termini di scalabilità, specie nella capacità di gestire le applicazioni embedded. E' caratterizzato da una grande flessibilità di impiego per quello che riguarda la piattaforma sul quale è utilizzato sia questa Linux, UNIX o Windows oltre alla capacità di modificare la configurazione derivante dalle esigenze delle singole applicazioni con le quali interagisce, requisito derivato dalla sua natura open-source.
- **Elevate performance** In base al tipo di applicazione da realizzare è possibile variare la configurazione per ottenere le migliori performance. MySQL è in grado di soddisfare le più esigenti aspettative di prestazioni di qualsiasi sistema con alta velocità di carico per l'utenza e di elaborazione delle transazioni.
- **Alta affidabilità** MySQL offre una varietà di opzioni di configurazioni per garantire alta disponibilità e affidabilità con soluzioni ad alta velocità per garantire la replica delle informazioni e la ridondanza.
- **Svilppo di Applicazioni** Uno dei motivi per cui MySQL è il più popolare database open source, è che fornisce supporto completo per ogni necessità di sviluppo di applicazioni. All'interno del database, è garantito il supporto per stored procedure, trigger, e molto altro ancora. MySQL fornisce anche connettori e driver (ODBC, JDBC, ecc) per sviluppare qualsiasi tipo di applicazione in molti linguaggi di programmazione.
- **Facilità di gestione** MySQL offre una vasta gamma di opzioni di avvio rapido per qualsiasi piattaforma oltre a caratteristiche di auto-gestione modificabili a partire da file configurazione. MySQL fornisce anche una suite completa di gestione grafici e strumenti di per la risoluzione dei problemi e il controllo del funzionamento di molti server MySQL da una singola postazione di lavoro.

- **Open source** MySQL non è un tipico progetto open source, tutto il software è di proprietà ed è supportato da MySQL AB, ma il progetto offre una combinazione unica di libertà legata al mondo open source, unendo il sostegno e il supporto della casa produttrice.

- **Costi** Utilizzare MySQL nella realizzazione di nuovi progetti porta le imprese a risparmiare molti fondi. Attraverso l'utilizzo di server MySQL è possibile effettuare operazioni di scale-out con utilizzo di architetture hardware a basso costo che non vanno ad intaccare le prestazioni. Inoltre, l'affidabilità e la facilità di manutenzione del database MySQL permette agli amministratori di non perdere tempo nella risoluzione dei problemi di prestazioni, lasciando la possibilità di concentrarsi sullo sviluppo del software.

2.5 JSON

JSON (JavaScript Object Notation) è un semplice formato per lo scambio di dati. Per le persone è facile da leggere e scrivere, mentre per le macchine risulta facile da generare e analizzarne la sintassi.
Si basa su un sottoinsieme del Linguaggio di Programmazione JavaScript, Standard ECMA-262 Terza Edizione - Dicembre 1999.
JSON è un formato di testo completamente indipendente dal linguaggio di programmazione, ma utilizza convenzioni conosciute dai programmatori di linguaggi della famiglia del C, come C, C++, Java, JavaScript, Perl, Python, e molti altri. Questa caratteristica fa di JSON un linguaggio ideale per lo scambio di dati. JSON è basato su due strutture:

- Un insieme di coppie nome/valore. In diversi linguaggi, questo è realizzato come un oggetto, un record, uno struct, un dizionario, una tabella hash, un elenco di chiavi o un array associativo.

- Un elenco ordinato di valori. Nella maggior parte dei linguaggi questo si realizza con un array, un vettore, un elenco o una sequenza.

Queste sono strutture di dati universali. Virtualmente tutti i linguaggi di programmazione moderni li supportano in entrambe le forme. E' sensato che un formato di dati che è interscambiabile con linguaggi di programmazione debba essere basato su queste strutture.

In JSON, assumono queste forme:

- Un oggetto è una serie non ordinata di nomi/valori. Un oggetto inizia con (parentesi graffa sinistra) e finisce con (parentesi graffa destra). Ogni nome è seguito da : (due punti) e la coppia di nome/valore sono separata da , (virgola).
- Un array è una raccolta ordinata di valori. Un array comincia con [(parentesi quadra sinistra) e finisce con] (parentesi quadra destra). I valori sono separati da , (virgola).
- Un valore può essere una stringa tra virgolette, o un numero, o vero o falso o nullo, o un oggetto o un array. Queste strutture possono essere annidate.
- Una stringa è una raccolta di zero o più caratteri Unicode, tra virgolette; per le sequenze di escape utilizza la barra rovesciata. Un singolo carattere è rappresentato come una stringa di caratteri di lunghezza uno. Una stringa è molto simile ad una stringa C o Java.
- Un numero è molto simile ad un numero C o Java, a parte il fatto che i formati ottali e esadecimali non sono utilizzati.

2.6 Hibernate

Nell'ambito informatico il termine persistenza viene utilizzato per indicare la possibilità dei dati di sopravvivere anche dopo la cessazione del programma che li ha creati, proprietà non immediata in quanto all'atto dell'esecuzione vengono salvati nella memoria RAM, che per definizione è volatile. Il salvataggio dei dati si può ottenere utilizzando principalmente due metodi, ovvero mediante file system oppure attraverso un DBMS (Database managment System).

2.6.1 Vantaggi

L'utilizzo di un DBMS è considerato migliore per applicazioni che lavorano con grandi moli di dati in quanto fornisce un buon controllo sulla ridondanza, una riduzione di tempi per lo sviluppo di applicazioni, flessibilità d'uso, funzioni di backup, assistenza e ripristino e molti alti vantaggi. Nel linguaggio Java per accedere al database si usa ricorrere alle API JDBC. Per fare ciò il programmatore deve aggiungere alle classi

dei metodi che implementano le operazioni CRUD (Create, Read, Update, Delete). Questa tecnica occupa mediamente però un terzo del tempo totale per lo sviluppo dell'intera applicazione, dovuto alla scrittura del codice che implementa il database ma soprattutto per il mantenimento della connessione. Un'ulteriore punto di debolezza è dato dalla differenza tra la rappresentazione dei dati nel modello object-oriented e quella del mondo relazionale.
Per cercare di ridurre tale inefficienza sono state create delle tecniche di programmazione atte a fornire delle interfacce per implementare i dati dal modello relazionale al modello a oggetti e viceversa, facilitando di fatto notevolmente il lavoro svolto dal programmatore per rendere gli oggetti persistenti. La principale di queste tecniche Object Relation Mapping (ORM). Attraverso questo mezzo ogni oggetto viene reso persistente nel database. Tramite inserimento di nuovi record i cui campi contengono i valori degli attributi dell'oggetto stesso, si viene quindi a creare la relazione tra oggetto Java e tabella SQLException. Una soluzione ORM consiste di quattro componenti caratteristici:

- Una API per eseguire le operazioni CRUD sugli oggetti delle classi del modello.
- Un linguaggio o una API per specificare query che fanno riferimento alle classi e alle proprietà delle classi stesse
- Uno strumento per la specifica del mapping mediante metadata.
- Una tecnica per l'implementazione di ORM per iteragire con oggetti transazionali al fine di eseguire funzioni di ottimizzazione

L'associazione fisica tra la classe e la tabella viene ottenuta mediante l'utilizzo di file di descrizioni detti file di mapping, in cui si specificano le modalità di conversione tra gli attributi e dell'oggetto e i campi della tabella. In definitiva i principali vantaggi derivanti dall'uso di una soluzione ORM sono:

- Snellimento della parte di codice riservata alla persistenza dei dati.
- Maggiore facilità per quanto riguarda la manutenzione del codice grazie alla separazione tra il modello ad oggetti e il modello relazionale.
- Performance più efficenti grazie alle numerose opzioni di ottimizzazione presenti.
- Elevata portabilità rispetto alla tecnologia DBMS utilizzata.

L'esempio più noto di ORM per il linguaggio Java è Hibernate. Hibernate può essere utilizzato principalmente in modalità Native o Provider. Nel primo caso si utilizzano dei file XML di configurazione per realizzare la mappatura tra le classi da rendere persistenti con le tabelle del database: precisamente ad ogni classe Java si associa il suo file di mappatura XML. La seconda modalità prevede l'inserimento diretto all'interno delle classi del codice inerente alla mappatura utilizzando lo stile Java Annotation (JPA). Questo ha il vantaggio di ridurre i file utilizzati nel progetto, ed è proprio questa la modalità utilizzata nella mia tesi.

2.7 JPA

Il processo di trasformazione da oggetto istanza di una classe a dato di un database prende il nome di ORM (Object Relational Mapping). JPA (Java Persistence API) non è un prodotto, ma una specifica Java per mappare un POJO su un database, pertanto richiede un'implementazione: ci sono diversi progetti che ne hanno implementato la specifica, alcuni commerciali e altri open source. Il vantaggio offerto da JPA consiste nell'abilitare un mapping oggetti-relazioni attraverso annotazioni , definendo come avviene il mapping tra classi Java e tabelle di un database relazionale. JPA definisce inoltre le API di un EntityManager, il cui scopo è la gestione a runtime di queries e transazioni su oggetti resi persistenti.
JPA è la più recente tra le specifiche Java volte a fornire la persistenza. Le prime o non hanno avuto seguito o hanno registrato problemi, in particolare per quanto riguardava la complessità e il degrado delle prestazioni come nel caso di EJB CMP 1.0 e 2.0.
Ciò ha portato alla creazione di un'altra specifica Java, lo standard JDO, che ha riscosso attenzione e ha visto lo sviluppo di diverse implementazioni commerciali o open source, ma senza trovare l'appoggio dei principali attori del Java EE.
Nel mezzo del confronto tra gli standard, continuarono a prosperare soluzioni basate su set di API proprietarie. Tra le principali ricordiamo Hibernate e TopLink. In buona parte la diffidenza verso CMP era dovuta alla diffidenza verso Java EE, vista come troppo complicata e troppo rivolta a soluzioni specifiche (e proprietarie). Tutto ciò ha portato a una nuova specifica, EJB 3.0, con l'obiettivo principale di ridurre la complessità e unificare quanto di buono avevano da offrire le soluzioni offerte dai prodotti non conformi agli standard.

Ne è scaturita la specifica JPA. Attualmente la maggior parte degli attori presenti nel campo della persistenza hanno rilasciato implementazioni delle specifiche JPA, inclusi Hibernate, TopLink, Kodo. JPA stesso si è evoluto passando alla specifica 2.0, approvata nel 2009, il cui scopo è offrire caratteristiche già presenti negli ORM più popolari. Le principali caratteristiche aggiunte riguardano:

- L'estensione delle funzionalità di mapping (ad esempio il supporto alla collezione di oggetti embedded e liste ordinate).
- Il supporto alla validazione.
- Query Criteria API.

Capitolo 3

Case Study

La piattaforma Prenota Facile basa il suo funzionamento su un sistema Client-Server:

- Il Server svolge le operazioni necessarie per realizzare il servizio come ad esempio: la ricerca degli appuntamenti, la gestione e l'aggiornamento dei questi e la loro integrità.
- Il Client accede ai servizi ed alle risorse del server, rappresenta l'interfaccia utente della piattaforma, provvedendo ad inviare al server le richieste formulate ed elaborando le risposte provenienti da quest'ultimo.

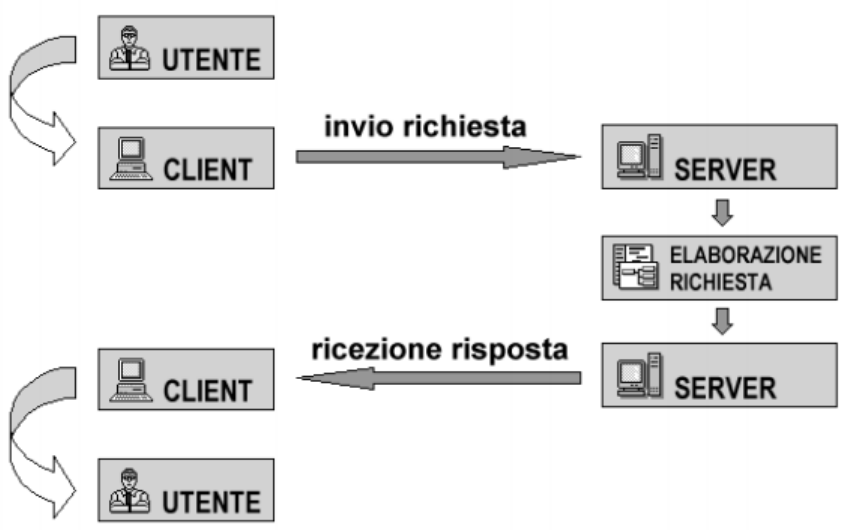


Figura 3.1: Struttura della piattaforma

Passiamo ora al dettaglio l'organizzazione dei dati per poi nelle sezioni successive descrivere l'operato del server e del client.

3.1 Database

Prima di descrivere le funzionalità del Server e del Client ho pensato di presentare il Domain Model così da facilitare la comprensione di questi.

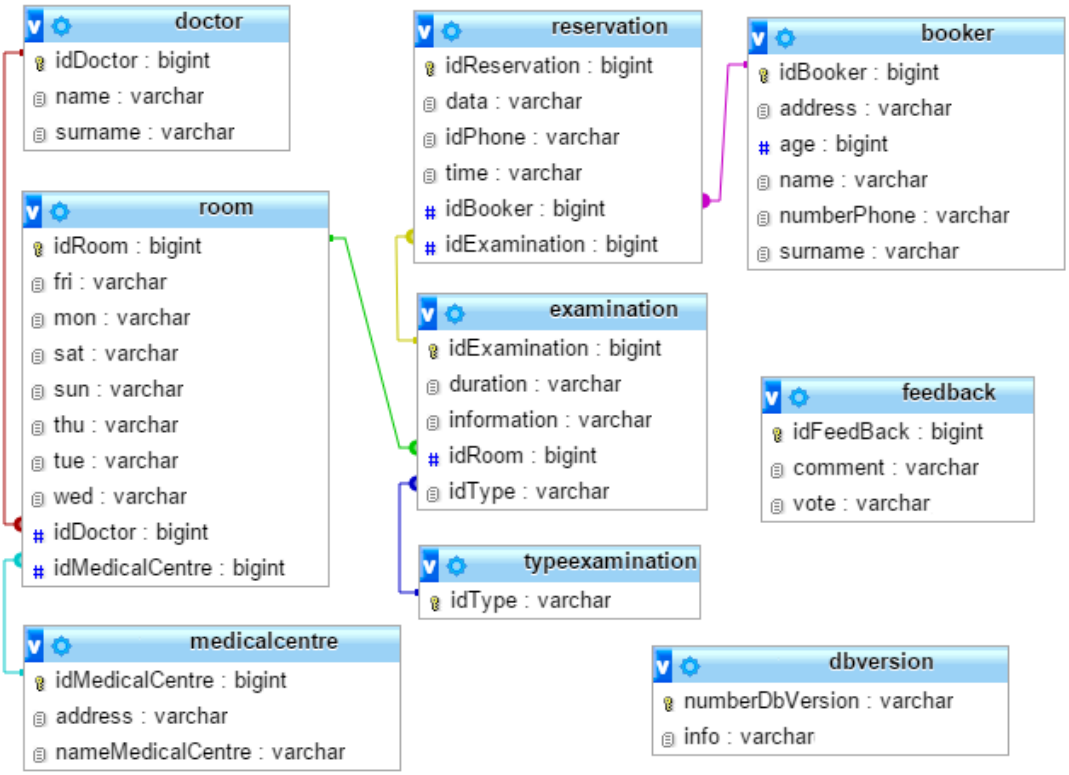


Figura 3.2: Struttura del Domain Model

Guardando la figura troviamo:

- la tabella **MedicalCentre** memorizza il nome e l'indirizzo del centro medico che eroga un determinato servizio, informazioni utile per fornire all'utente qualora non lo conoscesse l'ubicazione di questo.

- la tabella **Room** la quale è in relazione "molti ad uno" con la tabella Medical Centre, infatti questo può contenere varie stanze in ognuna delle quali viene svolta una tipologia di visite. Mi è sembrato opportuno creare dei campi per ogni giorno della settimana, infatti ogni ambulatorio può svolgere orari sempre diversi e può essere aperto o chiuso più volte in un giorno.
  Ho deciso inoltre di mettere in relazione "molti a molti" anche la tabella Room e Doctor poiché ogni dottore è specializzato in alcune determinate visite che avvengono nella stanza dove sono contenuti i macchinari necessari allo svolgimento di queste.

- la tabella **Doctor** contiene il nome ed il cognome dei dottori presenti nella struttura.

- la tabella **TypeExamination** contiene la categoria di visita medica, ho deciso questa suddivisione per permettere una ricerca più facile e veloce. Ogni categoria è in relazione "uno a molti" con le visite, infatti è abbastanza intuitivo capire che una categoria racchiude più visite.

- la tabella **Examination** contiene le visite mediche, all'interno viene memorizzato la durata della visita, la stanza nelle quale questa viene fatta, il nome della visita e giustamente come ho detto in precedenza a quale categoria appartiene.

- la tabella **Booker** memorizza i dati delle persone che usufruiscono del servizio, per in un futuro, poter effettuare indagini statistiche.

- la tabella **DBVersion** memorizza la versione del database e il giorno nel quale è avvenuta la modifica, informazione necessaria per avere un client sempre aggiornato.

- la tabella **Reservation** contiene le prenotazioni effettuate dai clienti, quindi l'ora e il giorno, per distinguere le varie prenotazioni abbiamo inoltre deciso di associare come id univoco l'IMEI dello smartphone.

- la tabella **Feedback** memorizza i commenti che l'utente può decidere o meno di lasciare, questi possono essere anche anonimi.

3.2 Lato Server

L'implementazione del sistema è basata sul pattern architteturale MVC (Model-View-Control) questo ha l'intento di disaccoppiare il più possibile tra loro le parti dell'applicazione adibite al controllo, all'accesso ai dati ed alla presentazione.

- Il core dell'applicazione viene implementato dal Model, che incapsulando lo stato dell'applicazione definisce i dati e le operazioni che possono essere eseguite su questi. Definisce le regole di business per l'interazione con i dati, esponendo alla View ed al Controller rispettivamente le funzionalità per l'accesso e l'aggiornamento.

- Il Controller ha la responsabilità di trasformare le interazioni dell'utente della View in azioni eseguite dal Model.Il Controller non rappresenta un semplice "ponte" tra View e Model. Realizzando la mappatura tra input dell'utente e processi eseguiti dal Model e selezionando la schermate della View richieste, il Controller implementa la logica di controllo dell'applicazione.

- La logica di presentazione dei dati viene gestita solo e solamente dalla View, la quale si occupa dell'interazione con l'utente e quindi della visualizzazione dei dati.

Un altro pattern utilizzato all'interno del sistema è stato il DAO, questo ha l'intento di disaccoppiare la logica di business dalla logica di accesso ai dati.
Il Data Access Object è un oggetto specializzato in operazioni di persistenza per una classe di oggetti del modello ed è in grado di caricare, salvare e modificare oggetti di una classe per un riutilizzo futuro.
I vantaggi tratti dal suo utilizzo sono: aumento della coesione e della modularità del codice ottenuta separando lo strato del modello da quello della persistenza ed una riduzione dell'accoppiamento poiché isola lo strato del controllo dalla tecnologia della persistenza (il controllo interagisce esclusivamente con i DAO).

3.2.1 Funzionalità

Il server esaminando il campo "text" dell'oggetto JSON che riceve dal client decide, a seconda del valore contenuto, quale operazione svolgere.
Mostrerò ora le funzionalità a cui può adempiere il server, che sono:

- **getAllCategory()** restituisce tutte le tipologie di visite che un centro medico mette a disposizione, pertanto viene controllato se nel pacchetto trasmesso sia specificato il centro medico altrimenti viene restituito un messaggio di errore al client. Qualora il pacchetto contenga tutti i campi richiesti per adempiere a tale operazione, si accede al database e nel pacchetto ritrasmesso nel campo "text" vengono inserite tutte le categorie di visite.

- **getAllExamination()** il funzionamento di tale metodo è simile a quello precedentemente elencato con l'unica differenza che il server controlla se nel pacchetto JSON ci sia il campo "type_visit", qualora questo sia presente vengono restituite tutte le visite di quella categoria.

- **getMyVisit()** tramite questa operazione è possibile richiedere al Server tutte le visite prenotate da un determinato utente. Come già specificato in precedenza ad ogni utente è specificata una chiave univoca data dall'IMEI del telefono con il quale decidono di prenotarsi. Nel caso in cui non fosse stata fatta nessuna prenotazione viene inviato all'utente un messaggio che lo porta a conoscenza di tale circostanza.

- **deleteVisit()** è il metodo che si occupa della cancellazione di una determinata visita, nel pacchetto vengono ricercati i campi: "hour", "data", "name_visit" e "id_phone" con il quale viene ricercata e cancellata la visita; ad operazione conclusa viene notificato il nuovo stato all'utente.

- **getInformationAbout()** ha il compito di fornire informazioni su una determinata visita come il dottore che la effettuerà, il numero della stanza dell'ambulatorio, etc.

- **getHours()** questa è una delle funzionalità più importanti del Server. Come scelta implementativa ho deciso di fornire all'utente due modalità di prenotazione, questa che chiamerò *veloce* ed un'altra che presenterò nel prossimo punto.

La modalità *veloce* restituisce una lista che rappresenta l'elenco del *primo appuntamento giornaliero disponibile* in ordine cronologico in dieci giorni successivi alla prenotazione. Questa scelta è stata adottata per velocizzare al massimo la procedura di prenotazione. Il server quindi accede tramite la visita che si vuole prenotare alla stanza dove questa viene svolta, controlla i giorni in cui l'ambulatorio è aperto e in quale ore svolge il servizio per poi controllare se durante queste ore vi sono già prenotazioni in corso.

- **getFreeHoursbyDay()** è la seconda tipologia di ricerca di appuntamenti possibili, infatti qualora l'utente non fosse soddisfatto degli orari proposti dalla modalità *veloce* può decidere di scegliere la data e visualizzare per quel giorno gli appuntamenti disponibili. La ricerca degli orari è simile al metodo getHours() con la differenza che vengono ricercati prima gli appuntamenti prenotati per quel giorno.

- **saveComment()** come è facilmente intuibile dal nome, questo metodo svolge il compito di salvare i commenti, sulla piattaforma "Prenota Facile", nel database. Viene fatta un'analisi sul campo "other" del pacchetto JSON e qualora in questo non fosse presente una stringa contenente il nome di chi ha inviato il commento, questo viene salvato come anonimo.

- **getDbVersion()** questo metodo è nato da una mia scelta implementativa, in quanto ho deciso di salvare nel client le tabelle TypeExamination ed Examination per avere user-experience meno stressante e più veloce possibile. Questa mia scelta però ha portato al problema della sincronizzazione tra database nel client e quello del server, per questo motivo ho deciso di implementare un metodo che restituisca la versione corrente del database.

- **bookReservation()** questo metodo come è facilmente intuibile ha il compito di prenotare una determinata visita medica, affinché la prenotazione possa avvenire in modo corretto sono necessari nel oggetto JSON i campi: "hour", "data", "name_visit" e "id_phone". Una volta avvenuta la corretta prenotazione viene restituito all'utente lo stato della prenotazione che può essere avvenuta con successo o essere fallita in quanto un altro cliente ha prenotato prima la visita.

3.3 Lato Client

L'applicazione mobile della piattaforma "Prenota Facile" è implementata in ambiente Android, di seguito presentiamo questa mostrando prima le scelte implementative e successivamente i relativi screenshot. All'apertura dell'app viene mostrato all'utente un immagine di benvenuto, nascosto agli occhi di questo, invece, vengono eseguiti alcuni controlli ed operazioni importantissime al corretto funzionamento dell'applicazione stessa. In primo luogo viene controllato se il dispositivo abbia un accesso alla rete, in mancanza di ciò viene restituito una AlertDialog dove si invita questo a connettersi.
Una volta effettuato il controllo del networking si passa a verificare lo stato del Server tramite il metodo isHostReachable() qui mostrato.

```
public static boolean isHostReachable(String serverAddress, int
    serverTCPport, int timeoutMS){
    boolean connected = false;
    Socket socket;
    try {
        socket = new Socket();
        SocketAddress socketAddress = new
            InetSocketAddress(serverAddress, serverTCPport);
        socket.connect(socketAddress, timeoutMS);
        if (socket.isConnected()) {
            connected = true;
            socket.close();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return connected;
}
```

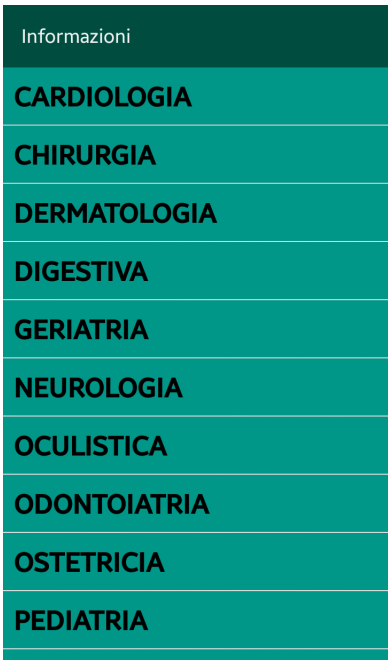
Nel caso il Server non rispondesse entro il tempo indicato in "timeoutMS" si procede a mostrare all'utente una finestra dove lo si invita a provare a contattare in un secondo momento il servizio. Solo dopo che tutti questi controlli hanno restituito un esito positivo viene scaricato sul dispositivo mobile una parte del database contenuto nel Server; questa operazione come specificato nella sezione precedente è stata una mia scelta implementativa per velocizzare l'operazione di ricerca è prenotazione.

(a) Schermata iniziale

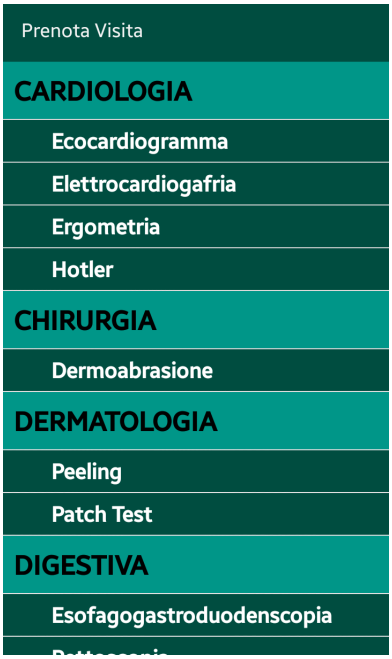
(b) Menù

Qualora il database scaricato non fosse nella stessa versione di quello del Server viene aggiornato e mostrato a schermo un Toast dove si informa l'utente del corretto stato dell'applicazione (a).
Ora l'utente potrà tramite il menù decidere quale operazione effettuare (b).
Premendo il primo pulsante possiamo visualizzare tramite una lista espandibile le visite disponibili suddivise per categorie (c). Per creare questa Activity mi sono servito del widget ExpandableListView. Tramite le linee di codice che mostrerò a breve viene settato l'adapter ovvero il componente che si occupa della rappresentazione grafica dei dati e dell'interazione con essi.

```
private MyExpandableAdapter adapter;
...
adapter = new MyExpandableAdapter(parentItems, childItems);
adapter.setInflater((LayoutInflater)
                    getSystemService(Context.LAYOUT_INFLATER_SERVICE),
                    PrenotationsActivity.this);
expandableList.setAdapter(adapter);
```



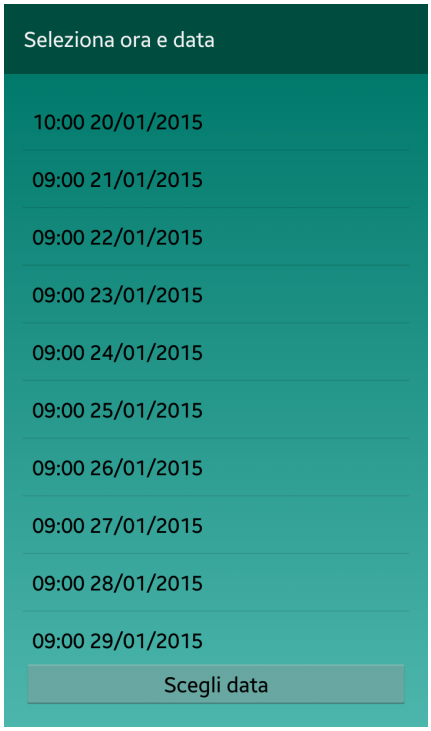
(c) Lista Tipologia visite



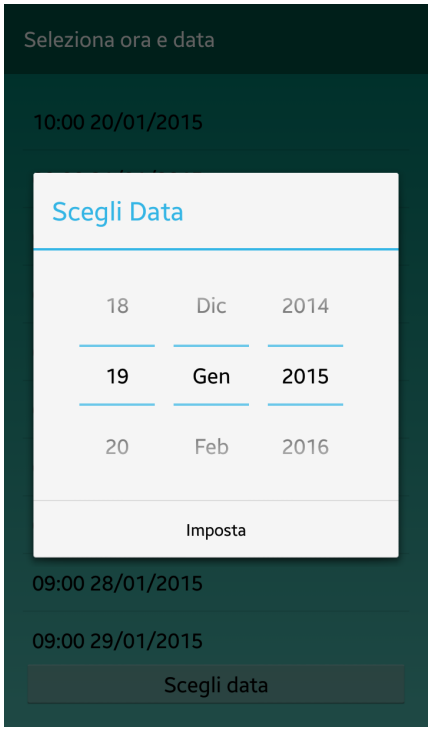
(d) Visite disponibili

Ora sarà possibile scegliere la visita e continuare la prenotazione semplicemente cliccandoci sopra, compariranno così a schermo tutte gli appuntamenti disponibili per quella visita (e).
Qualora non fossimo soddisfatti degli orari proposti, premendo sul pulsante "Scegli data" sarà possibile scegliere di visualizzare gli appuntamenti liberi di un determinato giorno (f). Al server pertanto verrà passata la data scelta, con il DatePicker; di seguito è mostrata una porzione di codice che svolge questo compito.

```
AlertDialog.Builder builderData = new
                                AlertDialog.Builder(VisitActivity.this);
final DatePicker picker = new DatePicker(VisitActivity.this);
picker.setCalendarViewShown(false);
builderData.setView(picker);
builderData.setPositiveButton("Imposta",
                               new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int id) {
    int    day  = picker.getDayOfMonth();
    int    month= picker.getMonth();
    int    year = picker.getYear();
                                );
```



(e) Lista appuntamenti disponibili

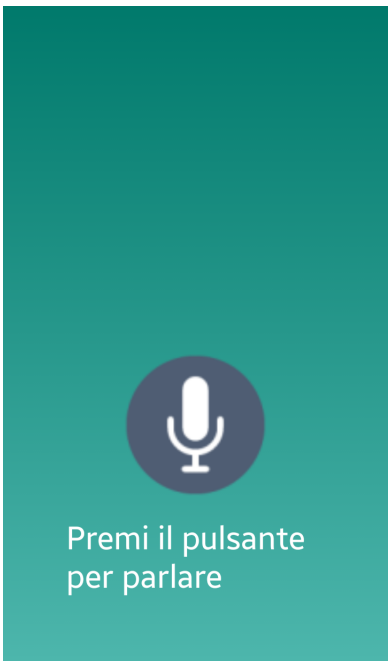


(f) Scelta della data

Premendo a lungo il primo pulsante del menù sarà possibile richiamare, invece, la funzionalità del Voice Helper e prenotare la visita medica pronunciando direttamente il suo nome (g).
Il cuore del servizio che mette a disposizione l'app di "Prenota Facile" è racchiuso nelle linee di codice che mostrerò tra poco.
Alla pressione del pulsante con disegnato il microfono, viene creato l'Intent del riconoscimento vocale, specificato che l'input vocale sarà basato sulla lingua installata dal sistema e avviato il metodo startActivityForResult responsabile dello stato del riconoscitore (h). Qualora vi fossero problemi viene mostrato a schermo un messaggio, una volta pronunciata la visita l'applicazione controllerà che questa sia presente o meno nel database e mostrerà un feedback visivo all'utente che sta utilizzando l'applicazione.

```
private void promptSpeechInput() {
Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
                RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, Locale.getDefault());
```

```
intent.putExtra(RecognizerIntent.EXTRA_PROMPT,
                getString(R.string.speech_prompt));
try{
    startActivityForResult(intent, REQ_CODE_SPEECH_INPUT);
    }
catch (ActivityNotFoundException a) {
    Toast.makeText(getApplicationContext(),
    getString(R.string.speech_not_supported),
    Toast.LENGTH_SHORT).show();
            }
        }
```



(g) Voice Helper

(h) Riconoscimento vocale attivo

Tramite il pulsante "Cerca Visita" figura (b) è possibile invece conoscere le informazioni necessarie su una determinata visita, come ad esempio la stanza dell'ambulatorio, nome e ubicazione del centro medico, da chi verremo visitati.
Anche qui vi è la possibilità di utilizzare il Voice Helper il quale leggerà al posto nostro le informazioni riportate.
L'implementazione di questa porzione di codice è semplicissima ma aggiunge una potenzialità all'applicazione incredibile. Viene prima creato l'oggetto, impostata la lingua e successivamente passata la stringa da leggere.

```
TextToSpeech ttobj;
ttobj=new TextToSpeech(getApplicationContext(),new
    TextToSpeech.OnInitListener() {
       @Override
       public void onInit(int status) {
          if(status != TextToSpeech.ERROR){
              ttobj.setLanguage(Locale.ITALY);
            }
          }
       });
...
ttobj.speak(toSpeak, TextToSpeech.QUEUE_FLUSH, null);
```

Tornando al menù della figura (b) è possibile, premendo l'icona del calendario, visualizzare tutti gli appuntamenti prenotati e cancellarne alcuni, l'app provvederà ad inviare al Server la nostra richiesta e mostrare a schermo la corretta riuscita dell'operazione.
Passiamo ora a descrivere le altre due modalità di prenotazione delle visite mediche cioè quelle tramite QR–code e tramite tag NFC. La prenotazione tramite QR–code avviene cliccando sull'apposito pulsante e avvicinando il codice alla fotocamera.
La lettura avviene utilizzando l'app esterna Barcode Scanner, se questa applicazione non è installata nel dispositivo è il sistema "Prenota Facile" che, dopo previa autorizzazione, la installa sul dispositivo.

```
try {
            Intent intent = new Intent(ACTION_SCAN);
            intent.putExtra("SCAN_MODE", "QR_CODE_MODE");
            startActivityForResult(intent, 0);
    }
catch (ActivityNotFoundException anfe){
        showDialog(QRcodeActivity.this, "No Scanner Found", "Download
            a scanner code activity?", "Yes", "No").show();
        }


...
        downloadDialog.setPositiveButton(buttonYes, new
            DialogInterface.OnClickListener() {
```

```
            public void onClick(DialogInterface dialogInterface, int
                i) {
                 Uri uri = Uri.parse("market://search?q=pname:" +
                    "com.google.zxing.client.android");
                 Intent intent = new Intent(Intent.ACTION_VIEW, uri);
                 try {
                     act.startActivity(intent);
                 } catch (ActivityNotFoundException anfe) {

                 }
            }
        });
...
```

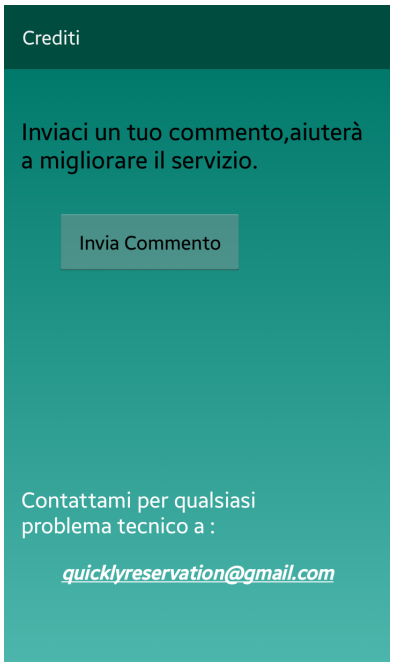
Una volta che "Barcode Scanner" sarà presente sul nostro dispositivo mobile, sarà anche possibile sfruttare tale funzionalità (i). Scansionato il QR–code l'app esterna ritornerà il valore letto e solo dopo aver verificato che questo sia realmente una visita svolta dal centro medico viene contattato il Server che restituirà gli orari disponibili (j); qualora non fossimo soddisfatti di questi sarà possibile tramite il pulsante "Scegli data" descritto in precedenza, scegliere un determinato giorno per verificare la presenza di appuntamenti liberi. Questo codice descrive il funzionamento dell'applicazione dal momento in cui viene ritornato il valore letto a quando l'app contatta il Server.

```
...
if (resultCode == RESULT_OK) {
    String contents = intent.getStringExtra("SCAN_RESULT");
    String format = intent.getStringExtra("SCAN_RESULT_FORMAT");
    Toast toast = Toast.makeText(this, "Content:" + contents + "
       Format:" + format, Toast.LENGTH_LONG);
    textViewQR.setText(contents);
    toast.show();
    ...
    cS = new ConnectServer();
    cS.execute(contents);
    }
    ...
```

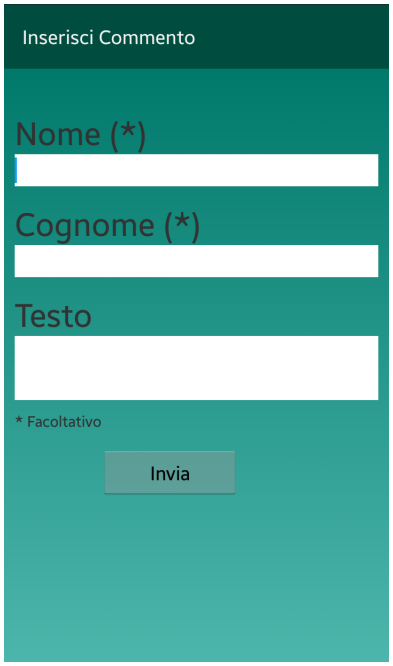
(i) Barcode Scanner

(j) Verifica Qr-Code

Per prenotare la visita medica tramite tag NFC è necessario che questa tecnologia sia supportata dal dispositivo e che sia attiva altrimenti l'applicazione mostrerà all'utente il problema. La prenotazione può avvenire in due modi o tenendo l'app chiusa allora il sistema Android provvederà ad aprire questa e contattare il server, oppure cliccando sull'apposito pulsante del menù di figura (b).
Il codice è abbastanza lungo e tedioso quindi ho preferito non riportarlo sopratutto per non appesantire la lettura. Una volta letto il tag, l'algoritmo è simile a quello del QR–code infatti viene verificata la presenza della visita nel database e chiesto al Server gli appuntamenti disponibili, qualora non fossimo soddisfatti potremmo sempre scegliere manualmente la data figure (e) ed (f). Nell'ultimo pulsante del menù (b) è possibile, invece, richiedere un supporto tecnico qualora vi fossero problemi con l'app oppure tramite il pulsante inviare un commento sul servizio o eventuali mal funzionamenti. La scelta di inviare nome e cognome è facoltativa, completato il campo commento sarà possibile inviare questo e l'app ci mostrerà la corretta esecuzione o meno dell'operazione. Mostriamo i relativi screenshot di questa funzionalità.



(k) Schermata di aiuto



(l) Completamento campi

Mostriamo infine una porzione dell'AndroidManifest questo è il file XML che contiene informazioni dettagliate riguardo l'applicazione, più precisamente definisce il nome del package, il codice della versione, la versione del SDK minima per l'utilizzo dell'applicazione (minSdkVersion), l'immagine che rappresenta l'applicazione sul dispositivo (icon), i permessi di sistema dell'app, il nome dell'applicazione (label) e il nome delle activity presenti nell'applicazione.

```
...

<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
    android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission
    android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.NFC" />

...

    <activity
        android:name=".NFCActivity"
        android:label="@string/menuNFCactivity" >
        <intent-filter>
```

```
            <action android:name="android.nfc.action.TECH_DISCOVERED"
                />
            </intent-filter>
            <meta-data
                android:name="android.nfc.action.TECH_DISCOVERED"
                android:resource="@xml/filter_nfc" />
        </activity>
    ...

</manifest>
```

In questa parte parte di codice vengono mostrati i permessi richiesti dall'applicazione al sistema Android, permessi necessari a controllare la presenza di una connessione internet, la possibilità di utilizzo del NFC e dell'accesso alla rete.
Nella seconda parte del codice, invece, osserviamo gli Intent filters necessari affinché rilevato il tag NFC l'applicazione ottenga i permessi per gestirlo.

Capitolo 4

Portabilità

La Portabilità o porting, è un processo usato per adattare l'esecuzione di un software in un ambiente diverso da quello originariamente pensato, per esempio differente CPU, differente sistema operativo, o differente linguaggio di programmazione.
Un software può essere considerato "portable" se eseguirne il porting è semplice (poco costoso), rispetto ad effetture un porting partendo da zero, cioè riscrivendo tutto, in tal caso la portabilità diventerebbe un'attività complessa e costosa.

4.1 Lato Client

Portare un'applicazione su un'altra piattaforma quindi richiede più sforzo di quello che si crede. La traduzione dell'applicazione in un altro linguaggio, non è l'unico problema, bisogna conoscere anche i comportamenti degli utenti abituati all'utilizzo di ciascun sistema operativo: siamo davvero certi che un utente Android si aspetti le stesse "risposte" del sistema rispetto ad utente iOS o Windows Mobile? Siamo sicuri cioè che le tre piattaforme si usino nello stesso modo? Basta pensare al numero di tasti presenti su iPhone rispetto ai modelli che usano Android per avere già qualche dubbio sulla risposta.
Le interfacce utente di iPhone, Android e Windows Mobile sono molto simili, ma esistono delle differenze peculiari che non dobbiamo trascurare se vogliamo fare il porting di una applicazione.
Ci sono infatti delle gesture, dei comportamenti a cui un utente è abituato che in un sistema fanno una cosa, mentre nell'altro, un'altra.
Dobbiamo tenere estremamente conto di come l'applicazione viene solitamente usata

per non introdurre interazioni inattese dall'utente. In virtù di quanto detto allora andiamo ad analizzare il porting della parte client verso le piattaforme con più mercato come iOS e Windows Mobile.
I linguaggi di programmazione alla base dei tre sistemi mobile sono differenti pertanto dobbiamo immediatamente abbandonare l'idea di un "cut and copy" ed iniziare a tradurre il codice manualmente anche perchè tool come j2Objc forniti da Google ci tradurrebbero si, la parte relativa alle entità, ma questo potrebbe portare solo a rallentamenti nel porting ed anche problemi di compatibilità.
Dal punto di vista dell'user experience invece ci imbattiamo come detto precedentemente della mancanza del tasto "back" sui dispositivi dotati di iOS cosa che però non accade in quelli Windows Phone, pertanto per il sistema Apple dovremmo implementare un tasto che ci fornisca la possibilità di tornare indietro.
Un altro problema che riscontriamo nella portabilità dell'applicazione in ambiente iOS è l'assenza dell'antenna NFC, in virtù di ciò perdiamo purtroppo la comodità della comunicazione di prossimità.
In seguito a quanto detto, il porting del lato client della mia tesi ha un costo abbastanza elevato per l'adattamento del codice per l'ambiente Windows Mobile ma non stravolge il normale funzionamento dell'app, cosa che purtroppo non avviene per il sistema iOS dove al costo del porting del codice va anche aggiunta una revisione al design del prodotto.

4.2 Lato Server

Per quanto riguarda il lato Server invece durante tutto il periodo di realizzazione dell'app e la successiva fase di test questo è rimasto sul mio pc, ma per un utilizzo da parte di terzi con un costo di portabilità non eccessivo è possibile acquistare le prestazioni di Amazon EC2.

4.2.1 Amazon EC2

Il servizio Amazon Elastic Compute Cloud (Amazon EC2) è un servizio web con capacità di calcolo nel cloud scalabile. E' progettato per gli sviluppatori al fine di semplificare il web computing.
Il servizio Amazon EC2, rivolto a tutti gli sviluppatori di piccole dimensioni che non hanno il capitale per acquistare nuove macchine o che non possono gestire improvvisi

picchi di carico computazionale, permette il calcolo nel cloud.
L'interfaccia del servizio web (la AWS Management Console) permette all'utente di ottenere e configurare senza sforzo le funzionalità del servizio.
Il servizio riduce a pochi minuti il tempo richiesto per ottenere e caricare nuovi server virtuali (conosciuti anche come server instances), permettendo all'utente di aumentare o ridurre le prestazione del server virtuale a seconda delle necessità. Infatti il termine Elastic (elastico) indica che il servizio permette agli sviluppatori di scalare immediatamente i requisiti al fine di soddisfare immediatamente questi picchi, cosa che altri servizi di cloud computing non permettono.
Spesso, infatti, questi ultimi forniscono un numero fisso di risorse per un determinato periodo di tempo, limitando le capacità dell'utente. Inoltre, usando un metodo di pagamento pay-per-use, offre la possibilità di pagare solo per le risorse effettivamente usate.
Le principali caratteristiche del servizio sono:

- **Elasticità** Amazon EC2 concede all'utente di aumentare e diminuire le prestazioni hardware e software in pochi minuti. Si possono commissionare una, centinaia o persino migliaia di istanze simultaneamente. Dato che tutto questo è controllato dalle API del servizio web, le applicazioni eseguite possono persino automaticamente aumentare o diminuire le prestazioni a seconda delle loro esigenze.

- **Controllo Completo** Dato che si ha accesso come root, l'utente ha il pieno controllo delle proprie istanze, come se stesse operando su una macchina in locale. Grazie all'interfaccia web, si può tranquillamente fermare, riavviare e spegnere una qualsiasi istanza

- **Flessibilità** L'utente ha la possibilità di scegliere tra diversi tipi di istanze, sistemi operativi e pacchetti software.
  Amazon EC2 consente di selezionare le dimensioni di storage, memoria e CPU, e il software (sistemi operativi ed applicazioni) più consoni ai propri requisiti

- **Compatibilità con altri servizi Amazon** Amazon EC2 funziona in congiunzione con Amazon Simple Storage Service (Amazon S3), Amazon Relational Database Service (Amazon RDS), Amazon SimpleDB e Amazon Simple Queue Service (Amazon SQS) per fornire una soluzione completa per il calcolo, l'elaborazione delle query e lo storage in una vasta gamma di applicazioni.

- **Sicurezza** Per rendere sicure le proprie istanze Amazon EC2 prevede l'utilizzo di un firewall, che controlla l'accesso a internet delle istanze e la comunicazione tra gruppi di istanze ed è configurabile via interfaccia Web, e Amazon Virtual Private Cloud (Amazon VPC), con cui è possibile definire una rete virtuale che simula una normale rete aziendale, permettendo quindi l'interazione tra diverse istanze e la creazione di sottoreti e tabelle di routing.
  Amazon EC2 offre la possibilità di collocare le istanze in posti diversi. I posti sono composti da regioni e Availability Zones.
  Le Availability Zone sono posizioni distinte che sono state progettate per essere isolate da errori di altre Availability Zones e per ridurre al minimo la latenza di connettività di rete con le Availability Zones nella stessa regione. Con il lancio di istanze in Availability Zones separate, è possibile proteggere le applicazioni dai guasti di una singola posizione. Le regioni posseggono da una o più Availability Zones, sono geograficamente disperse, e sono in particolari aree geografiche o paesi.

- **Economicità** Tramite un negozio online (AWS Marketplace) c'è la possibilità di trovare, acquistare e implementare rapidamente il software che viene eseguito su AWS.
  E' possibile utilizzare AWS Marketplace per avviare rapidamente il software ' preconfigurato e esso sarà addebitato in base all'utilizzo, o a ore o a mesi. AWS si occupa di fatturazione e pagamenti, e gli oneri software appaiono sulla bolletta AWS.

Capitolo 5

Conclusioni

Prenota Facile è un sistema che semplifica e velocizza operazioni quotidiane.
Può essere usato, come abbiamo visto e approfondito nel capitolo 3, per prenotazione mediche, ma anche per qualsiasi altro tipo di evento basato su turni come: prenotazioni di esami, di appuntamenti in ufficio, ritiro di merci, etc.
Peculiarità fondamentale della piattaforma è l'essere fruibile da tutti, grazie agli accorgimenti tecnologici in termini di *easy access*, senza distinzioni di età, predisposizione verso la tecnologia e, cosa molto importante a mio parere, può essere utilizzata anche da persone ipovedenti, poiché possiede una *user-experience* consona al loro stato e fornisce, inoltre, la possibilità di prenotare gli appuntamenti usando comandi vocali grazie al *Voice Helper*.
Ho fatto della facilità di utilizzo la base per la mia app, avendo pensato che i maggiori utilizzatori di questa potrebbero essere persone che, anche per l'età, non sono invogliate ad utilizzare un dispositivo mobile per prenotare visite, appuntamenti, controlli, etc.
Oltre alla funzionalità del Voice Helper su citata, la piattaforma *Prenota Facile* mette a disposizione la prenotazione guidata o semplificata con *tag NFC* o *QR–code*.
L'utilizzo del QR–code ha molti vantaggi: può essere creato con un costo quasi nullo e distribuito ai clienti sotto forma di calendari, agende, etc con lo scopo di pubblicizzare e attirare più clientela possibile, inoltre questa è una funzionalità che ritroviamo anche con il porting verso i sistemi iOS e Windows Mobile.
Il tag NFC ha un costo superiore rispetto ad un QR–code, ma ha il vantaggio di essere programmabile e questo può essere più funzionale quando situato nel luogo in cui il servizio stesso è erogato.

In conclusione sono molto soddisfatto del lavoro svolto, soprattutto in virtù dei progressi e dei miglioramenti apportati all'applicazione durante tutta la sua fase di analisi e sviluppo. La release attuale ha già tutti i requisiti per essere distribuibile per una prova sul "campo"; nonostante ciò sono molte gli *improvement* che sono sopraggiunti durante questi mesi e che a causa del poco tempo non ho potuto realizzare.

5.1 Sviluppi futuri

I possibili miglioramenti che potrebbero essere apportati alla piattaforma *Prenota Facile* sono:

- Creazione di una "Web Application" per l'inserimento, la modifica, e la cancellazione di prenotazione utilizzabile dal personale dell'ente che decide di usufruire della piattaforma; questa modifica permetterebbe così, ad esempio, in un ambiente medico di avere un'agenda di appuntamenti consultabile dai dottori.
- Creazione di una "Web Application" che supporti la prenotazione di appuntamenti ed utilizzabile dai clienti direttamente da pc, tale modifica avrebbe come fine quello di fornire all'utente una nuova modalità di gestione di eventi.
- Implementare all'interno dell'applicazione mobile una funzionalità di ingrandimento per permettere una facile lettura e consultazione di questa.
- Possibilità di decidere font e skin dell'intera applicazione, tale miglioramento è necessario per gli utenti che hanno problemi con determinati tipi di colore oppure che hanno difficoltà nel decifrare scritte diverse dal grassetto.
- Permettere all'utente di decidere ad esempio da quale dottore vuole essere visitato, qualora vi fosse la possibilità di scegliere.
- Aggiungere la possibilità di prenotazione di appuntamenti in ambiente medico, direttamente dalla ricetta.
- Possibilità di spostare, se possibile, il proprio appuntamento.
- Utilizzare le Google Cloud Messaging per notificare, qualora prenotazioni precedenti alla nostra vengano cancellate, l'eventuale possibilità di modificare il nostro appuntamento.

Capitolo 6

Ringraziamenti

Sento il dovere di esprimere profonda gratitudine al Professore Massimo Regoli, relatore, il quale durante tutto il periodo di svolgimento della tesi mi ha aiutato con suggerimenti, critiche ed osservazioni.
Ha dimostrato interesse fin da subito verso questa mia idea.
La sua esperienza e conoscenza dell'ambiente mobile inoltre mi ha permesso di realizzare una tesi moderna ma già proiettata verso il futuro.
A lui vanno i miei più sentiti ringraziamenti per questi mesi.
Vorrei ringraziare infine tutte quelle persone che hanno condiviso insieme a me questo percorso.

Ed impari a costruire tutte le strade oggi
perché il terreno di domani
è troppo incerto per fare piani.
E impari che puoi davvero sopportare,
che sei davvero forte, e che vali davvero.
E impari e impari e impari.

V. A. Shoffstall.

Bibliografia

Cay Horstmann, Gary Cornell *"Core Java Volume I"* PEARSON Prentice-Hall.

Cay Horstmann, Gary Cornell *"Core Java Volume II"* PEARSON Prentice-Hall.

"Android Programming" Edizioni Master, on line:
http://punto-informatico.it/PILibri/Dettaglio.aspx?id=238
ultimo accesso 11/02/2015

Massimo Carli *"Android Guida per lo sviluppatore"* APOGEO, 2010

Rossi Pietro Alberto *"Android by Example"*, on line:
http://www.sprik.it/guida/Android4_2.pdf
ultimo accesso 11/02/2015

Tutorial Android, on line:
http://android.devapp.it/
ultimo accesso 11/02/2015

Annotazioni JPA e operazioni CRUD in Hibernate, on line:
http://www.html.it/articoli/jpa-e-la-persistenza-in-java/
ultimo accesso 11/02/2015

Cinzia Bocchi *"Guida Hibernate"*, on line:
https://www.youtube.com/user/63E62L219
ultimo accesso 11/02/2015

Saverio Rubini, *"MySQL. Mettersi in tasca il database in open source"* Apogeo

Guida MySQL, on line:
http://www.html.it/guide/guida-mysql/
ultimo accesso 11/02/2015

Introduzione a JSON, on line:
http://www.json.org/json-it.html
ultimo accesso 11/02/2015

Java Persistence API (JPA), on line:
http://www.giuseppesicari.it/articoli/jpa-java-persistence-api/
ultimo accesso 11/02/2015

Il pattern Data Access Object, on line:
http://www.mokabyte.it/2002/04/pattern-dao.htm
ultimo accesso 11/02/2015

Cos'è la tecnologia NFC, on line:
http://www.tulipmobile.it/approfondimenti/35-cos-e-la-tecnologia-nfc.html
ultimo accesso 11/02/2015

Che cos'è l'NFC?, on line:
http://www.tulipmobile.it/approfondimenti/35-cos-e-la-tecnologia-nfc.html
ultimo accesso 11/02/2015

TAG NFC approfondimento e idee di utilizzo, on line:
http://www.massarielectronics.it/informatica/12-guide/35-approfondimento-e-idee-di-utilizzo-dei-tag-nfc.html
ultimo accesso 11/02/2015

Cos'è un codice QR?, on line:
http://www.qrmode.it/it/
ultimo accesso 11/02/2015