ALMA MATER STUDIORUM – UNIVERSITA' DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI SCIENZE

CORSO DI LAUREA IN SCIENZE DELL'INFORMAZIONE

WHAT'S THIS SONG? CONDIVIDERE REGISTRAZIONI SU PIATTAFORMA ANDROID

Relazione finale in

MOBILE WEB DESIGN

Relatore

Dott. Mirko Ravaioli

Presentata da

Davide Canducci

Sessione III
Anno Accademico 2013-2014

INDICE

Introduzione 3

Capitolo 1: INTRODUZIONE AL MONDO MOBILE
E FONDAMENTA DELL'APPLICAZIONE 5
1.1 PICCOLI CENNI DI STORIA 5
1.2 MINI FOCUS SUGLI STORES 7
1.3 EXCURSUS SU ANDROID 8
1.4 YAHOO ANSWERS: UNA COMUNITA' CON
DOMANDE E RISPOSTE 9
1.5 WHAT'S THIS SONG: CHE MOTIVETTO
HO IN TESTA? 10
1.6 DISPOSITIVI SUPPORTATI: PRO E CONTRO 11

Capitolo 2: PROGETTAZIONE DELL'APPLICAZIONE 15
2.1 ANALISI DEI REQUISITI 15
2.2 SCHERMATA DI REGISTRAZIONE E INVIO AUDIO . . 19
2.3 AIUTA LA COMUNITA': ASCOLTA
GLI AUDIO E RISPONDI 20
2.4 GESTISCI LE TUE REGISTRAZIONI 22
2.5 COMUNICAZIONI COL SERVER 22
2.6 DATABASE WHAT'S THIS SONG 25
2.6.1 Utente 26
2.6.2 Registrazione audio 27
2.6.3 Risposta 29

Capitolo 3: IMPLEMENTAZIONE DELL'APPLICAZIONE 31

3.1 SCHERMATA DI SPLASH ED INIZIO 31

3.2 IMPLEMENTAZIONE DELL'ACCOUNT UTENTE 33

3.2.1 Implementazione del login 33

3.2.2 Excursus su Asynctask 35

3.2.3 Implementazione della registrazione . 38

3.3 CUORE DELL'APPLICAZIONE 39

3.3.1 Registra – Ascolta – Invia al server . 39

3.3.2 Download lista registrazioni 44

3.3.3 Ascolta e rispondi 48

3.3.4 Gestione delle proprie registrazioni . 50

Capitolo 4: PROGETTI E PROSPETTIVE FUTURE 53

4.1 PICCOLE MIGLIORIE 53

4.2 GRANDI MIGLIORIE / EVOLUZIONI 56

Conclusioni 61

Bibliografia 63

INTRODUZIONE

Non credo sia errato definire il momento della laurea un crocevia che la società moderna, quantomeno quella in cui viviamo, ha esatto dalla generazione degli anni ottanta, ergo quella con la quale il sottoscritto si è trovato a condividere l'esistenza.

E se una minuscola minoranza ha talvolta la fortuna e talvolta un vero dono naturale da ottenere una laurea *honoris causa*, milioni di persone si confrontano ogni anno con il percorso universitario che, basandomi sulla mia esperienza personale, è pieno di ostacoli e difficoltà, dunque estremamente formativo. E dopo varie peripezie è arrivato pure per il sottoscritto il momento di scrivere il punto esclamativo in quello che è stato un lungo cammino.

L'idea di questa tesi, che intenderà descrivere la nascita di un'applicazione per *smartphones* prima e *tablets* poi, nasce in verità da un'amicizia, da una condivisione di idee generatasi proprio durante il lungo cammino di cui sopra. Mi sento infatti di affermare che, sebbene la formazione scientifica sia un elemento di grande importanza di questo mio percorso, la vera ricchezza è arrivata dalla maturazione del desiderio collaborativo con alcuni miei colleghi universitari, coi quali è nata una vera e propria amicizia.

Esistono in verità già altri progetti simili all'applicazione che andrò a descrivere nelle prossime pagine, eppure questi stessi, sviluppati da aziende che oggi hanno anche dimensioni molto elevate, non centrano proprio il punto che invece proverà a colpire il servizio generato dall'applicazione da me elaborata.

In breve, ognuno di noi molto probabilmente nella vita si è ritrovato con un motivetto musicale in testa senza saperne con esattezza titolo od autore. Cercherò dunque con l'aiuto più delle persone che della pura tecnologia di risolvere gran parte di questi "dilemmi", che sovente occupano parte dei pensieri di giornate intere (nel mio caso un quesito è durato più di cinque anni).

Il vero ostacolo di questo progetto è l'ostacolo che trovano tanti progetti più o meno validi nell'era internet. Un mio ottimo professore, sapiente in materia, ha infatti spiegato a me e a tutti i miei colleghi presenti al corso da lui tenuto che, semplificando, su internet non "vince" sempre il migliore, ma sovente chi arriva prima e riesce a crearsi un proprio spazio all'interno dell'immenso mercato virtuale. Per fare un esempio banale, se oggi stesso immettessi sul *web* un browser completamente gratuito, sicuro ed efficiente, molto probabilmente (salvo investimenti mastodontici in pubblicità) il mio software sarebbe completamente schiacciato dai vari *Microsoft Internet Explorer*, *Google Chrome* o *Mozilla Firefox* (per citare i tre più famosi) anche se (ipoteticamente parlando) il mio prodotto vincesse ogni possibile confronto nei campi della prestazione, della sicurezza e dell'efficienza. E questo perché l'utente medio non è disposto a cambiare servizio in continuazione ma, una volta trovato ciò che più o meno lo soddisfa, ne usufruisce a tempo indeterminato, salvo eventi particolari e talvolta catastrofici (sempre virtualmente parlando).

Cercherò quindi di descrivere al meglio l'applicazione che ho pensato, progettato ed implementato, cercando di essere il più cristallino possibile, consapevole che il mio lavoro abbia già portato una crescita personale e speranzoso che il futuro di questo *software*, uno dei tanti, sia fiorente

CAPITOLO 1

Nel primo capitolo il filo del discorso graviterà attorno all'introduzione al mondo *mobile*, vale a dire al mondo del quale fanno parte tutti i dispositivi mobili capaci di interconnettersi tra loro ed aventi quindi la possibilità di essere trasportabili (aggiungerei facilmente). Caratteristica peculiare di questi dispositivi è in genere la minor capacità di elaborazione (*processing*) e memorizzazione attraverso processori meno veloci e memorie meno capienti rispetto ai dispositivi fissi (es. *personal computer*) per via delle loro dimensioni ridotte e della conseguente semplificazione dell'*hardware*.

1.1 PICCOLI CENNI DI STORIA

Se cambiamenti ed evoluzioni naturali richiedono ere e millenni, e se la stessa tecnologia umana ha conosciuto decenni e talvolta secoli di situazioni pressoché immobili, lo stesso non si può certo dire dell'evoluzione dell'*hardware*, vale a dire la parte fisica di un calcolatore (o più semplicemente *computer*), del quale fanno parte tutte quelle componenti elettroniche, elettriche, meccaniche, magnetiche, ottiche che ne consentono il funzionamento. Si parla infatti di una storia "breve" (metà del secolo scorso) ma intensa, dove il termine "obsoleto" talvolta veniva usato per definire componenti usciti 4-5 anni prima. E sebbene fossero già in commercio diversi tipi di *computers* (fissi) da diverso tempo, per il primo dispositivo mobile bisogna attendere fino al 1973, quando *Martin Cooper*, direttore della sezione Ricerca e sviluppo della **Motorola**, fece la sua prima telefonata da un cellulare, che pesava 1,3 kg e aveva una batteria che durava 30 minuti, ma che impiegava 10 ore a ricaricarsi. Mentre bisogna aspettare altri 10 anni per ottenere un modello che

potesse essere commercializzato (al prezzo di 4000 dollari). Nei successivi 20 anni grazie ad aziende quali *IBM* o Research In Motion (RIM) ed a grandi investimenti di tutte le aziende del settore informatico con una visione futuristica, si è arrivati ai primi *Smartphones*, vale a dire telefoni con capacità di calcolo non indifferente, ed un sistema operativo integrato; questi erano già piuttosto simili (per funzionalità fondamentali) a quelli moderni. Da metà degli anni '90 il *BlackBerry*, prodotto di grande successo della *RIM*, era già in grado di connettersi ad internet, di leggere la posta e navigare grazie ad un apposito *Browser* pensato appositamente per i *devices mobile*. Venne dunque *Nokia*, che fino al 2007 fu leader mondiale nella produzione di cellulari (ancora definiti con questo nome, poiché di *smart* probabilmente avevano ancora troppo poco, sebbene le dimensioni fossero migliorate considerevolmente negli anni). Per la vera "*smart revolution*" bisogna aspettare giugno 2007, con la discesa in campo telecomunicativo di quella che ad oggi è una delle più grandi aziende (del settore e non) di maggior valore al mondo.

Quando **Steve Jobs** infatti, *CEO* della **Apple Inc.**, presentò il primo ***iPhone***, si intuì sin da subito che qualcosa era cambiato rispetto a quanto si era visto negli anni, seppur pochi (storicamente parlando), precedenti. Il dispositivo lanciato sul mercato dall'azienda americana infatti si presentava tecnicamente come un cellulare con un sistema operativo (*iOS*) capace di fare praticamente tutto quello che era in grado di fare la concorrenza, ma di farlo con uno schermo capacitivo "enorme" per l'epoca (3,5'') e di farlo più velocemente ed intuitivamente. Il sistema operativo era progettato inoltre per aggiornarsi in continuazione, ma la cosa che realmente "vinse" la concorrenza, con tutta probabilità, fu il successivo inserimento di un "negozio" denominato ***App Store***, un vero e proprio grande magazzino, capace di contenere migliaia di queste "app" (applicazioni), presentate con un

layout grafico non indifferente, capace in breve tempo di creare un *business* immenso.

1.2 MINI-FOCUS SUGLI STORES

Avendo parlato di *application stores*, quindi di negozi virtuali di applicazioni (per quanto riguarda l'ambito che sto descrivendo), è d'obbligo descrivere in qualche riga ciò che s'intende esattamente con il termine appunto "applicazione", abbreviato, sempre in ambito *mobile*, in App. Le sopracitate sono dei veri e propri software atti a funzionare su dispositivi mobili come *tablets* e *smartphones*, ma ormai, da non tanto tempo a questa parte, in grado di essere riprodotti anche sui sistemi operativi dei veri e propri pc. Interessante il grafico in figura 1.3 che dimostra come i *downloads* delle *apps* siano cresciuti in maniera esponenziale negli anni e siano ancora destinati a farlo.

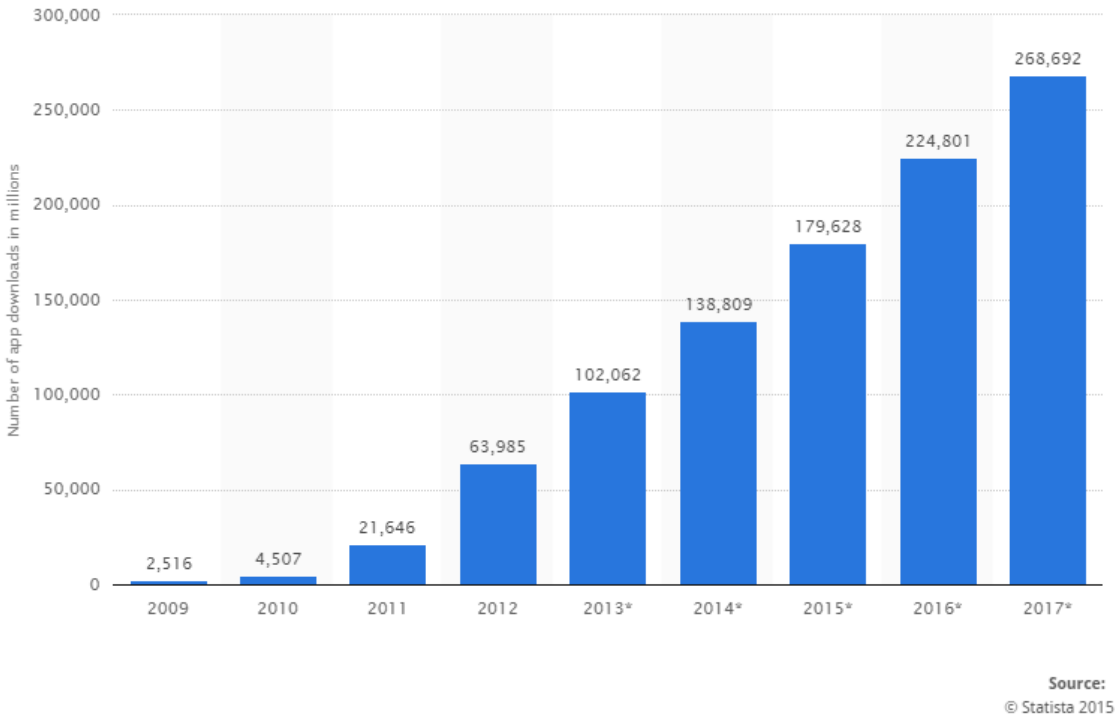

Figura 1.0

1.3 EXCURSUS SU ANDROID

Finora la descrizione è stata limitata al mondo *mobile*, ma chiunque abbia una vaga conoscenza dell'universo informatico è a conoscenza che esso è davvero paragonabile al nostro cosmo fisico, tante sono le discipline e le sottocategorie. E se prima è stata utilizzata la parola "mondo", in verità potrei parlare di un vero sistema solare, per fare un paragone con la fisica. Sistema che ha come necessità praticamente assoluta internet, e che comprende tanti mondi tra loro simili eppure differenti, tra i quali ho poco fa menzionato iOS, vale a dire il sistema operativo proprietario della *Apple*. In piena concorrenza con il sopracitato iOS, poco tempo dopo venne presentato **Android**, un sistema operativo nato per essere utilizzato in ambito *mobile*, quindi per poter essere utilizzato alla perfezione anzitutto sugli *smartphones*, ergo presenta caratteristiche peculiari che favoriscono l'uso appunto *smart* di un telefono. Non è intenzione di questa tesi entrare troppo nel dettaglio di aspetti e caratteristiche prettamente tecnici di un sistema operativo che, per quanto più "leggero" dei sistemi installati sui pc, fornisce oggi praticamente tutti i servizi dei più sofisticati fratelli maggiori. Una cosa però che non può essere omessa è che nasce sulla base di un *kernel* **Linux**. E' importante ciò sottolineare perché lo sviluppo di quest'ultimo è nelle mani della *Linux Foundation*, un'associazione senza fini di lucro nata nel 2007, che garantisce numerose distribuzioni gratuitamente. *Android* stesso è per la quasi totalità ***Free and Open Source Software***, e questo in soldoni significa che un qualsiasi programmatore può modificarne il codice a suo piacimento (sempre nei limiti consentiti dalle leggi). Per il resto, come i suoi concorrenti iOS e WP (*Windows Phone*), è caratterizzato da facilità di utilizzo, intuitività e come già detto leggerezza. Essendo ormai gli *smartphones*, nei paesi industrializzati, praticamente più numerosi degli abitanti stessi, è necessario

fornire ad ogni tipo di utente la possibilità di utilizzare il proprio telefono in maniera più semplice e veloce possibile. E le *App* che sono parte integrante dei telefoni non possono esimersi dall'esserlo a loro volta. Una buona *App* deve assolutamente garantire intuitività e velocità di utilizzo, anche considerato il fatto che generalmente si usa il telefono molto spesso ma per secondi o minuti, non ore (salvo giochi). Essendo poco il tempo a disposizione, esso deve essere ottimizzato al meglio.

1.4 YAHOO ANSWERS: UNA COMUNITA' CON DOMANDE E RISPOSTE

Per poter dare un'idea precisa di quello che farà l'applicazione che nei prossimi capitoli sarà descritta il più dettagliatamente possibile è sicuramente molto utile parlare di ciò che ha ispirato il progetto, vale a dire ***Yahoo! Answers***. L'idea di *Yahoo! Answers* è incredibilmente banale quanto geniale: ogni utente registrato su Yahoo! può inserire una o più domande che vengono state pubblicate nell'apposita sezione del sito. Tutti gli altri utenti provvedono, eventualmente, a rispondere. L'utente "creatore" della domanda sceglie poi la risposta più consona alla sua domanda premiando il rispondente con crediti virtuali. Si è creata in breve tempo (dal 2006 ad oggi) una comunità immensa che ad oggi conta 200 milioni di utenti in tutto il mondo e 15 milioni di visitatori quotidiani, di cui 2 milioni di visitatori unici giornalieri solo dagli USA. E' dunque una comunità viva, nata per rispondere probabilmente a quesiti più o meno difficili ed espansasi fino a assumere talvolta l'aspetto di un forum, ove qualunque tipo di domanda, da "mi aiutate a fare i compiti?" a "il mio ragazzo mi ha lasciata, cosa faccio?" fino a "CLO3- è polare o apolare?" è ben accetta ed ottiene una risposta. Ho parlato

prima dei crediti (punti) virtuali, ma di fatto come funziona il tutto? Si evince facilmente da questa tabella:

Azione	Punti
Attivazione del profilo su Yahoo! Answers	+100 (una tantum)
Invia una domanda	-5
Scegli la miglior risposta alla tua domanda	+3
I votanti hanno scelto "Nessuna miglior risposta" alla tua domanda	+5
Rispondi a una domanda	+2
Cancella una risposta	-2
Accedi a Yahoo! Answers	+1 (tutti i giorni)
Vota una risposta	+1
Vota "Nessuna migliore risposta"	0
La tua risposta viene scelta come migliore	+10
Ricevi un "pollice su" in una delle migliori risposte che hai fornito	+1 per ogni pollice in su
Un tuo contenuto (domanda, risposta, commento) viene eliminato per una violazione	-10

Figura 1.1

Ovviamente più punti si hanno maggiore è l'importanza che si ha all'interno della comunità e la libertà di operare.

1.5 WHAT'S THIS SONG: CHE MOTIVETTO HO IN TESTA?

Ciò che neanche l'attuale Yahoo! Answers tuttavia riesce a risolvere è la fastidiosa situazione nella quale, probabilmente, tantissime persone si sono trovate nella vita almeno una volta. Si tratta di quando si ha in testa una parte di una canzone (magari straniera), un motivetto, qualche nota che si impossessa della mente senza che questa sia a conoscenza della provenienza di ciò che la sta tenendo occupata. L'applicazione **What's This Song** ha l'ambizioso obiettivo di risolvere una volta per tutte queste situazioni scomode. E non si basa su software di riconoscimento vocale perché sarebbero in gran parte inefficaci.

Senza stare a scendere troppo nel dettaglio musicale infatti, è sufficiente pensare che le scale musicali, le pause, la stessa intonazione forniscono un'unicità nella registrazione

vocale che è pressoché indiscutibile. Uno stesso utente infatti registrando più volte non riuscirebbe a riprodurre perfettamente un audio magari registrato da egli stesso. Per questi motivi diverse applicazioni già esistenti hanno margini di successo realmente esigui, poiché si basano su fattori difficilmente confondibili come la comparazione di frequenze sonore. Per questo motivo l'applicazione What's This Song si concentrerà più sull'aspetto comunitario rispetto a quello puramente artificiale, dando dunque la possibilità ad ogni utente di registrare a voce (o con l'aiuto di uno strumento, ma sempre tramite *input* microfonico) e permettendo a tutti coloro che fanno parte di questa comunità di rispondere, aiutando l'utente "domandante". Dettagli più significativi verranno affrontati nei capitoli successivi.

1.6 DISPOSITIVI SUPPORTATI, PRO E CONTRO

Questa applicazione è stata pensata e sviluppata anzitutto per *smartphones*. Questo per diverse ragioni: la prima è proprio relativa al fatto che oramai ognuno di noi ha il proprio *smartphone* sempre a portata di mano, pertanto è realmente semplice accedere all'applicazione, registrare un audio ed in un attimo inviare la "domanda" tramite internet. Più complicato è sicuramente il metodo delle risposte, in quanto, a meno di essere soli, difficilmente si arriverà a riprodurre un audio in pubblico senza l'uso di auricolari o cuffie che garantiscano la privacy necessaria all'uso dell'applicazione. Un altro motivo per cui ho ritenuto proprio i "telefonini" il *target* ideale dell'applicazione sta nelle dimensioni abbastanza contenute di questi ultimi. Registrare infatti da *tablet*, per quanto fattibile (e nei progetti in un futuro prossimo) rimane più scomodo soprattutto per le dimensioni non ideali, mentre il *layout* stesso andrebbe riprogettato per rendere l'applicazione egualmente semplice e funzionale.

L'applicazione sfrutta la neonata versione 5.0 di *Android*, conosciuta con in nome *Lollipop*. Sarà ovviamente garantita la retro-compatibilità anche considerato il fatto che, come si può evincere dalla tabella e dal grafico che segue, l'ultima versione del robottino verde è ancora utilizzata da pochi utenti.

Version	Codename	API	Distribution
2.2	Froyo	8	0.4%
2.3.3 - 2.3.7	Gingerbread	10	7.4%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	6.4%
4.1.x	Jelly Bean	16	18.4%
4.2.x		17	19.8%
4.3		18	6.3%
4.4	KitKat	19	39.7%
5.0	Lollipop	21	1.6%

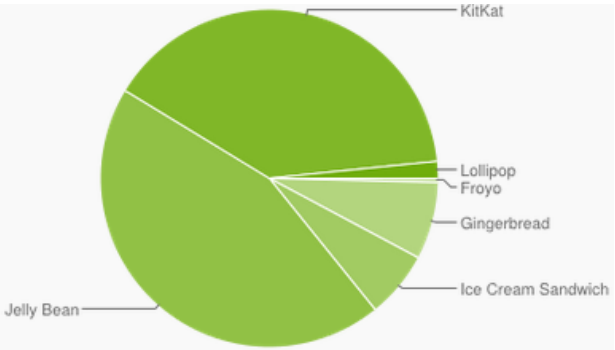

Figura 1.2

Come si può notare dalla figura 1.2, ad ogni versione di Android, corrisponde una nuova versione delle API. Le *Application Programming* Interface, sono librerie messe a disposizione dello sviluppatore generalmente all'interno di un SDK (*Software development kit*) cioè un pacchetto utile per lo sviluppo di software (applicazioni in questo caso). Se uno sviluppatore sceglie di sviluppare software, utilizzando *features* introdotte in API recenti, andrà incontro a un lavoro oneroso in termini di tempo nel caso in cui voglia garantire la retro-compatibilità (non sempre possibile però) con dispositivi che si basano su API Level inferiori. Di contro se uno sviluppatore non vorrà garantire retro-compatibilità sarà costretto a rinunciare a un determinato bacino d'utenza. Io ho scelto di sviluppare utilizzando le API Level 15, rivolgendomi quindi a quei dispositivi sul quale è installata

una versione uguale o superiore alla 4.0 (*Ice Cream Sandwich*). Questo vuol dire che *What's This Song* non potrà essere installata e utilizzata da dispositivi con una versione del Sistema Operativo antecedente alla 4.0. Infatti così facendo si andrà a coprire un bacino d'utenza molto elevato, superando l'80% dei dispositivi *Android*.

CAPITOLO 2

In questo secondo capitolo della tesi saranno affrontate tutte le problematiche affini alla progettazione dell'applicazione. Con progettazione è da intendersi l'attività che è alla base della costruzione/realizzazione di questo programma che dovrà essere semplice all'uso e mantenere la complessità alla base, laddove l'utente non può arrivare.

2.1 ANALISI DEI REQUISITI

Il capitolo sulla progettazione inizierà con la presentazione di quello che in gergo tecnico è chiamato documento dei requisiti. L'utilità di questo paragrafo è spiegare lo scopo dell'applicazione dalla prospettiva dell'utente, indicando quindi cosa questi possa fare, in maniera più dettagliata e ad ampio raggio rispetto al capitolo precedente. Saranno evidenziati in grassetto tutti quei concetti che costituiscono la struttura fondamentale dell'applicazione e che verranno approfonditi nel corso della stesura della tesi.
L'applicazione *What's This Song* si presenta con la necessità immediata di autenticazione, vale a dire ***Login*** obbligatorio. Questa scelta è dettata dal fatto che sarebbe inutile ragionevolmente far registrare un audio ad un utente non registrato. Come potrebbe riconoscere la propria registrazione dalle altre, successivamente? E come poter rispondere senza un'identità?
Pertanto è indubbio che l'autenticazione iniziale sia obbligatoria. Nel caso in cui non si abbiano ancora le credenziali per poter utilizzare l'applicazione, sarà data ovviamente la possibilità all'utente di procedere con la **registrazione**. Essa è comprensiva di uno ***username***, una ***password*** e un indirizzo ***email***. Lo *username* è stato pensato univoco (ognuno ha il proprio), per evitare doppioni che creerebbero grande confusione; la *password*, per rispettare un

minimo di standard di sicurezza, è richiesta di almeno 8 caratteri. Se in futuro l'applicazione avrà grande successo sicuramente i criteri della *password* saranno cambiati, magari obbligando gli utenti ad inserire lettere maiuscole, minuscole, numeri e caratteri speciali (come fanno attualmente i siti che minimamente tengono alla sicurezza degli *account*s dei propri iscritti). Un discorso a parte lo merita l'indirizzo *email*: se "imporre" all'utente di inserire anche quest'ultimo è stato oggetto di una lunga riflessione, presa anche coscienza del fatto che esistono sul web servizi come *10minutemail.com* che permettono la creazione di un indirizzo temporaneo da utilizzare laddove ci si vuole iscrivere a siti che si pensa di consultare solo una volta. La preoccupazione principale stava nel rendere la registrazione il meno invasiva possibile, per cui chiedere all'utente qualcosa di così privato (almeno teoricamente, salvo casi appena descritti) era cosa di non poco conto. L'importanza tuttavia di avere un indirizzo *email* da poter eventualmente utilizzare per contattare in casi particolari (ad esempio segnalazione dell'utente come dannoso, utente particolarmente bravo, eventuale verifica dell'*account*, …) ha fatto pendere la bilancia dalla parte del richiedere anche un indirizzo *email* agli utenti. Inoltre, in prospettiva futura, sarebbe interessante evitare di avere il server intasato da *bot* che, automaticamente, si iscrivono con utenze fasulle solo per rallentare e sabotare il servizio. Esiste chiaramente il metodo della validazione dell'indirizzo email, ed è qualcosa che tutt'ora può essere implementato. Tutti i dati di tutti gli utenti verranno immagazzinati in un *database* situato in un *server* dedicato, per cui eventuali correzioni ed aggiornamenti con tutta probabilità non saranno un problema. E' altresì ovvio che vi sarà totale rispetto della *privacy* per quanto riguarda soprattutto le *passwords* (che gli utenti digitali tendono a ripetere per ogni servizio). Una volta

autenticato l'utente avrà dunque la possibilità di fare diverse cose, prima tra tutte la registrazione di un audio. Descriverò brevemente in cosa consistono le scelte disponibili all'utente.

- **Registra un audio:** è una delle opzioni principali. L'idea di base consiste, all'apertura dell'applicazione e sempre che non sia la prima volta o non ci sia ancora stata autenticazione (cosa già affrontata in precedenza), nel permettere all'utente di registrare subito (all'opportuna opzione scelta) un proprio audio ed inviarlo ad un *server* dedicato, che lo trasformerà in vera e propria "domanda". Dell'audio verranno memorizzati il creatore, un titolo, la data ed il genere. L'audio verrà registrato fisicamente nella memoria dello *smartphone* dell'utente registrante.
- **Rispondi a qualche domanda:** per guadagnare crediti e mantenere viva la comunità ogni utente è "chiamato" a rispondere alle domande poste dagli altri utenti, anch'essi alle prese con motivetti in testa che non riescono a decifrare. All'utente verrà chiesto di scegliere un genere, quindi gli verrà data una lista coi titoli delle registrazioni degli altri utenti della comunità e la possibilità di scaricarne gli audio, per poi rispondere via semplice commento. Tramite la lista dei commenti è possibile segnalare eventuali utenti fastidiosi o tossici per la comunità.
- **Gestisci le mie registrazioni:** questa schermata sarà pressoché identica a quella descritta in precedenza, con la differenza che le registrazioni mostrate saranno esclusivamente quelle create dall'utente proprietario dell'account. L'utente in questione avrà quindi la possibilità di inserirsi egli stesso nella discussione riguardante la sua registrazione e scegliere quale commento "risponde" alla sua domanda (o semplicemente

quale risposta lo abbia aiutato maggiormente), premiando "l'aiutante".

Sarà disponibile infine un ***logout*** che permette all'utente di cancellare ciò che è stato precedentemente memorizzato (all'interno del telefono) riguardo ai suoi dati.
Per dare un aiuto visivo rispetto a quanto descritto il diagramma in figura 2.1 di seguito mostra nel modo più semplice possibile ciascuna opzione possibile.

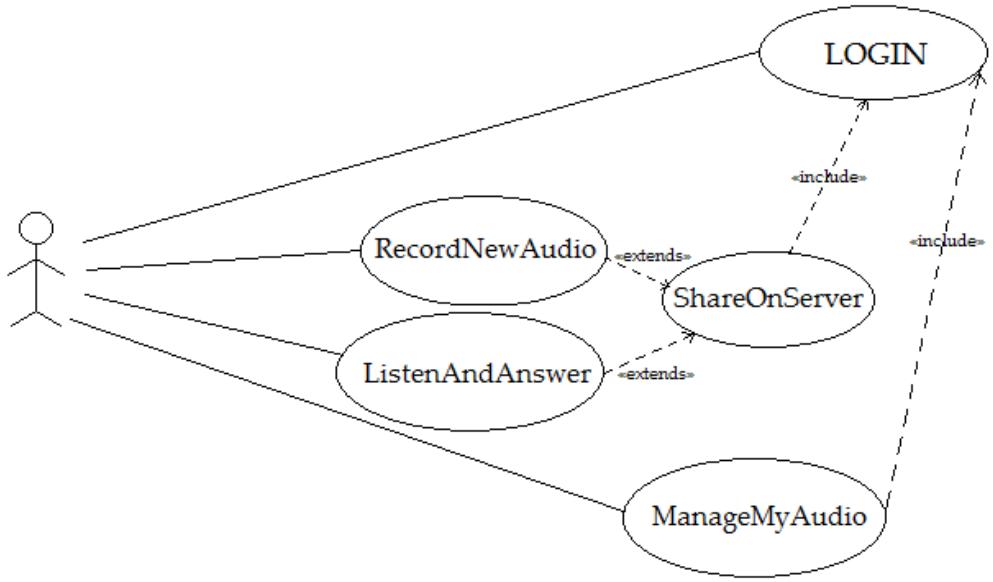

Figura 2.1

Come già detto dunque, l'utente ha in sostanza le seguenti scelte (che comportano un login obbligatorio): registrare audio inviarli al server, ascoltare gli audio altrui, partecipare a "discussioni" in generale. Esiste per tutti un'opzione aggiuntiva che permette, come già detto, di segnalare eventuali utenti scorretti, ma non è il vero cuore dell'applicazione, pertanto non è stata inserita (anche se ai fini della stabilità della comunità è importante).

2.2 SCHERMATA DI REGISTRAZIONE ED INVIO AUDIO

Senza la possibilità di **registrare** gli audio è banale che il software non abbia senso di essere. È fondamentale dunque partire da questa schermata, ricordando il fatto che l'utente dovrà senza meno essere autenticato per vederla. La schermata altro non è che una vera e propria classe atta a contenere tutti i componenti che serviranno alla registrazione dell'audio ed alla condivisione di esso. All'utente saranno visibili tre pulsanti, un menu a tendina dal quale scegliere il genere cui appartiene l'audio ed una casella di testo nel quale inserire un titolo relativo all'audio che si vuole registrare. Il vero cuore però della classe risiede comprensibilmente nei suddetti pulsanti, dei quali il primo ha il compito di far partire e bloccare la registrazione da microfono, il secondo permettere di riascoltare ciò che è stato appena registrato ed il terzo permette di inviare al server la registrazione. Focalizzando i particolari, è utile aggiungere il fatto che durante la registrazione o l'ascolto o l'invio al server dell'audio è impossibile premere gli altri pulsanti. La registrazione inoltre comporta la creazione fisica di un file audio (è stata scelta l'estensione .mp4) che andrà a crearsi nella cartella dedicata all'applicazione. Se la registrazione non è ritenuta consona dall'utente, nel momento in cui si decide di registrare un nuovo audio il precedente *file* verrà eliminato dalla memoria del telefono e non sarà pertanto più disponibile né ascoltabile. Altrimenti l'audio rimarrà come già detto nella memoria del telefono. L'invio al *server* terrà inoltre considerazione di tutte gli oggetti presenti nella schermata, quelli "visibili" e quelli "nascosti". Preleverà infatti l'audio, il genere, il testo scelto come titolo e il codice dell'utente e spedirà tutto al *server* dedicato, che si occuperà di creare nel ***database*** un *record* riguardante tutto ciò che gli è appena pervenuto.

Per fare un piccolo riassunto riguardo quanto appena descritto, propongo un semplice diagramma relativo all'attività.

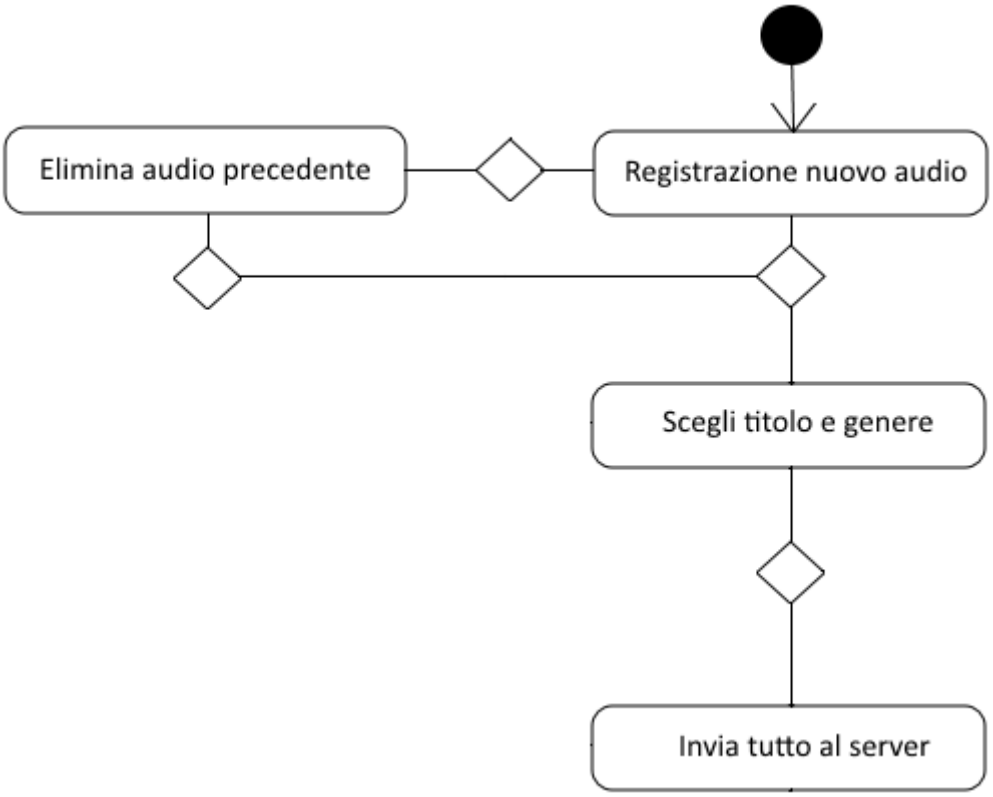

Figura 2.2

2.3 AIUTA LA COMUNITA': ASCOLTA GLI AUDIO E RISPONDI

Altro punto cardine dell'applicazione è la possibilità di **ascoltare** le registrazioni fatte dagli altri utenti della comunità e provare ad aiutare questi ultimi. Come già accennato viene dunque data la possibilità agli utenti, sempre previa registrazione ed autenticazione, di "scaricare" sul proprio *smartphone* la lista delle registrazioni per le quali gli utenti proprietari non hanno ancora scelto la risposta più consona. Anzitutto viene chiesto all'utente di scegliere il genere delle registrazioni che vuole aiutare a "risolvere"; una volta selezionato da una lista di generi preesistente la schermata mostra una lista comprendente titolo della

registrazione (scelto dall'utente registrante), *username* del creatore e la data di creazione della registrazione nel *server*. Aprendo una registrazione alla volta dunque l'utente avrà la possibilità di riprodurre ciò che è stato precedentemente registrato dagli altri utenti ed eventualmente di partecipare alla discussione relativa. Una volta data una risposta il *software* provvederà a tornare alla lista precedentemente descritta. All'interno della discussione ogni utente avrà il potere di **segnalare** eventuali risposte sgradite, con contenuti non appropriati. Spetterà poi alla moderazione dedicarsi alle segnalazioni con eventuali "punizioni" per i trasgressori. È considerevole sottolineare che senza l'opportuno ascolto della registrazione non è possibile partecipare alla discussione.

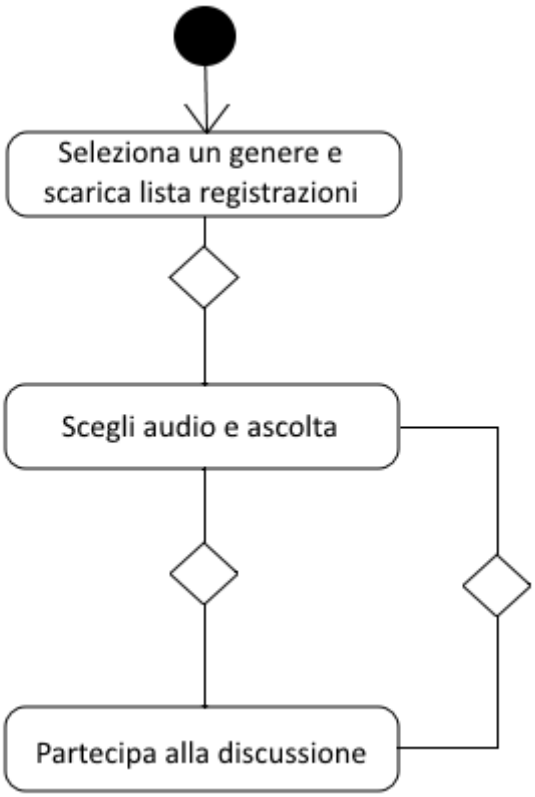

Figura 2.3

Il diagramma in figura 2.3 ancora una volta rappresenta in breve ciò che l'utente può fare.

2.4 GESTISCI LE TUE REGISTRAZIONI

La sezione riguardante la gestione delle proprie registrazioni in realtà altro non è che una perfetta copia della sezione appena descritta nel paragrafo 2.3. L'unica differenza sta nel fatto che la ricerca sul *server* si baserà sull'utente loggato e basta. Ciò che può fare in più il creatore degli audio oltre a segnalare eventuali risposte non consone, come già detto in precedenza, è **marcare** la risposta **giusta** e premiare (il *server* si occupa di tutto) l'utente che ha risposto, accreditandogli punti. Per il resto, il "creatore" ha gli stessi poteri di tutti gli altri, vale a dire può partecipare alla discussione come un qualsiasi utente.

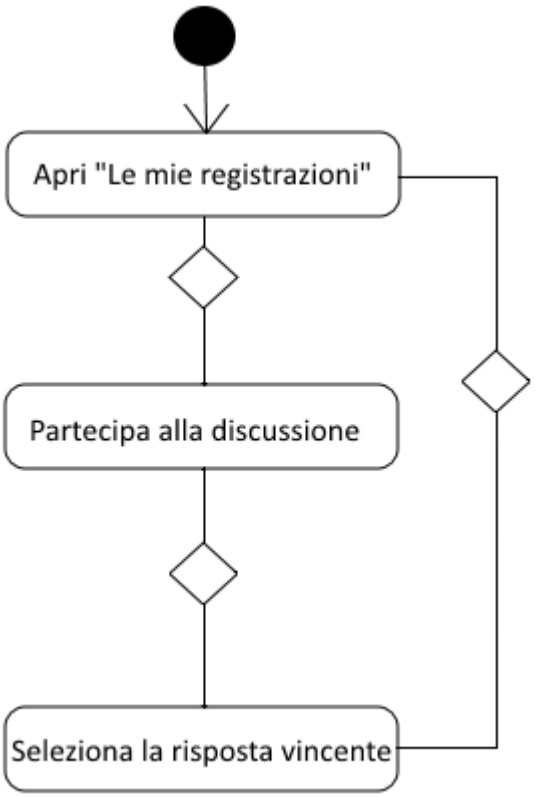

Figura 2.4

2.5 COMUNICAZIONI CON IL SERVER

Più volte all'interno delle pagine precedenti è stata menzionata la parola *server*, e sebbene non è di fondamentale importanza spendere un capitolo intero per descrivere quali

siano le caratteristiche generali dei server in generale, è comunque utile discuterne almeno le proprietà che sono indispensabili per il funzionamento del *software* che ho sviluppato. È indubbio infatti che l'app sia totalmente basata sulla connessione ad internet, e quando essa è assente la suddetta applicazione non funziona pressoché in alcuno dei suoi servizi. Dal momento che l'intero progetto è basato su una struttura ***Client-Server***, La rete è fondamentale appunto per la comunicazione tra i vari *Clients* (che sono i terminali sui quali l'applicazione è installata) ed il *Server* (luogo "fisico" nel quale risiede il database e sono situati gli audio registrati). In che occasioni e come quindi l'applicazione andrà a "dialogare" col server? Nelle prossime righe vi sarà una descrizione di ogni comunicazione col *server*.

- **Registrazione**: è già stato scritto a più riprese che all'utente vengono richiesti determinati parametri. Questi vengono inseriti all'interno del database nella tabella proprio dedicata agli utenti, denominata Utente. Essa, oltre ai campi richiesti *Username*, *Password* ed *Email*, memorizza anche i Crediti (tutti partono da un minimo altrimenti non possono fare nulla), le segnalazioni ricevute per cattiva condotta e soprattutto un codice identificativo generato automaticamente, utile ai fini delle ricerche (ricercare tra interi è più facile che tra stringhe).
- **Login**: l'app comunica sempre con la tabella Utente e rileva la validità dei dati che la persona che sta tendando di "loggarsi" ha inserito. Login si occupa inoltre di controllare se l'indirizzo *email* è già presente nel *database*, informando l'utente in caso affermativo. A *login* effettuato sul dispositivo verrà salvato un cookie che ha durata 2 anni.

- **Registrazione audio**: come già detto in precedenza, una volta che l'utente decide di inviare al *server* l'audio che ha registrato, esso viene registrato nel *database* assieme al codice dell'utente creatore, al genere, alla data di registrazione e ad un campo denominato "Risolta" che conterrà 0 se la domanda ancora non ha ricevuto risposte segnalate come corrette, 1 se invece è stata trovata una risposta appagante. In verità ciò che viene registrato nel *database*, riguardo al file vero è proprio, è solo il suo *link*. L'audio viene memorizzato in una cartella a parte, dalla quale verrà successivamente "estratto" nelle ricerche.
- **Scegli genere** e scarica lista: Riguardo proprio alle ricerche, la ricerca "madre" viene da questa classe (già descritta). L'app infatti invia al *server* il testo del genere scelto ed il *server* restituisce una lista di *link* (non visibili), con titolo della registrazione, utente che ha registrato e data di registrazione.
- **Ascolta** e **commenta** una registrazione: sempre nell'ambito delle registrazioni audio, quando l'utente cerca di ascoltare la registrazione fatta da altri automaticamente l'app si occupa di fare una richiesta per prelevare dal *server* l'audio richiesto dall'utente e di scaricarlo sul device, per poi riprodurlo. Assieme all'audio sarà scaricata pure la lista con i commenti già presenti relativi alla registrazione selezionata. Aggiungendo un commento si dialoga nuovamente col *server* inviandogli il testo relativo a ciò che si è scritto assieme ovviamente al codice dell'utente. Le risposte saranno dunque contenute nel database all'interno della tabella "Risposta" con campi ID_Risposta, che è l'identificatore, il codice dell'utente che ha risposto, l'identificatore della registrazione, il valore 0 nel campo "Vincente" nel caso in cui non sia stata selezionata come risposta

"vincente", 1 altrimenti. Assieme ovviamente al testo vero e proprio della risposta.

- **Segnala risposta** / **seleziona vincente**: il *server* viene contattato solamente per cambiare i campi (già descritti) relativi alla registrazione (Risolta), alla risposta (Vincente/Segnalazioni) e all'utente (aggiunta di Crediti/Segnalazioni).

Viene proposto infine uno schema ER, alias *Entity-Relationship* che riassume tutto quanto scritto finora.

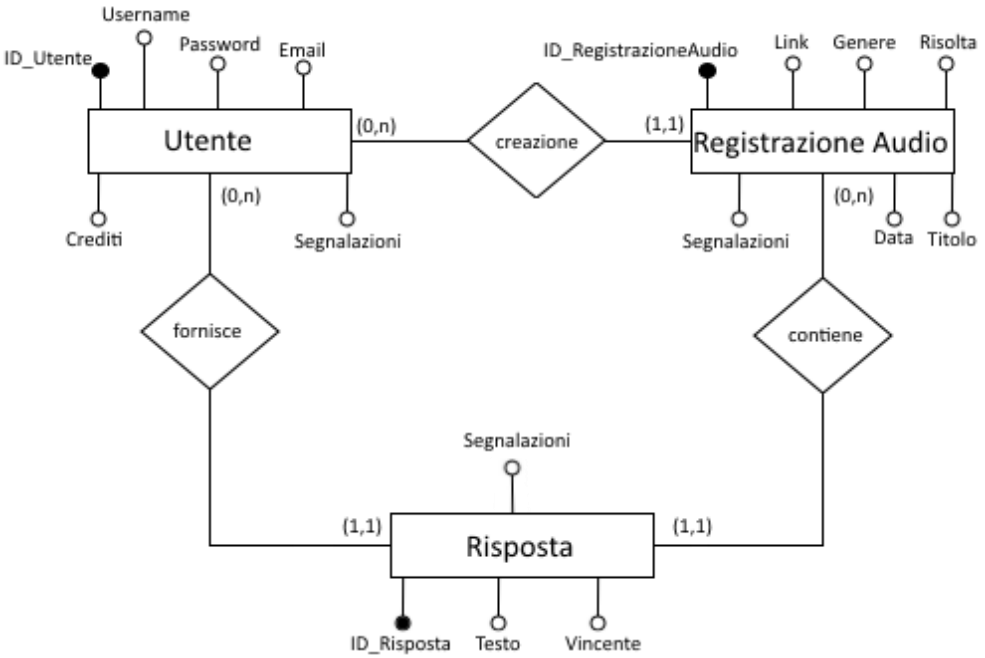

Figura 2.5

D'ora in avanti la descrizione scenderà nel dettaglio delle dinamiche interne al *database* appunto, e si cercherà di motivare le scelte che sono state.

2.6 DATABASE WHAT'S THIS SONG

E' stato già detto a più riprese quanto sia importante essere connessi ad internet per l'applicazione, esplicando le operazioni che questa compie in comunicazione col *server*. Ed effettivamente si può dire che il *server* stesso sia alla base

del progetto, e con esso il *database* che fornisce tutta l'organizzazione necessaria. Il *database* è composto, come appena descritto, da tre grandi tabelle-entità che conterranno tutti i dati utili al corretto funzionamento dell'applicazione.

2.6.1 Utente

La tabella Utente viene riempita principalmente durante l'operazione di registrazione di un nuovo utente. Procederò ad analizzare uno per gli attributi.

- **ID_Utente:** è un campo rappresentato da un numero intero. Questo numero è automatico, si auto-incrementa ed è la chiave primaria della tabella.
- **Username:** è un campo rappresentato da una stringa, al massimo di 20 caratteri, *case-insensitive* e unico anch'esso. Si sarebbe potuto rendere questo attributo come chiave primaria, essendo richiesta l'unicità, tuttavia ai fini delle ricerche nel database è sempre meglio che la chiave primaria si un numero e non una stringa (le operazioni di ricerca sono molto più veloci).
- **Password:** è un campo stringa pure questo, in particolare *case-sensitive*. La *password* ovviamente non è univoca, in quanto può capitare che due utenti scelgano inconsapevolmente la medesima *password*, anche perché da una ricerca è venuto fuori che alcune di esse (123456, password, qwerty) sono estremamente comuni. L'unica cosa che viene richiesto all'utente, come già detto, è che la password sia di almeno 8 caratteri.
- **Email:** è stato implementato nel *software* un metodo che garantisca il fatto che l'indirizzo *email* digitato dall'utente sia reale (quindi in stile xxx@yyy.zzz). Non è stata ad ogni modo inserita la "verifica" dell'account (tramite invio di un *link* tramite *email* da cliccare, per attivare il proprio profilo), che è tuttavia nei

progetti. L'email come già spiegato servirà per contattare privatamente gli utenti in casi "speciali".

- **Crediti:** è un campo rappresentato da un numero intero. Tiene semplicemente conto dei crediti che l'utente in questione è riuscito a raccogliere. Ogni operazione effettuata dall'utente può far guadagnare o perdere crediti, un po' come descritto nel capitolo 1 riguardo *Yahoo Answers*. In generale, facendo "domande" (vale a dire registrando audio) i crediti calano, rispondendo i crediti crescono. Le risposte selezionate come "vincenti" fanno crescere considerevolmente i crediti.
- **Segnalazioni:** anche questo è un campo rappresentato da un intero, ma raccoglie piuttosto le segnalazioni ricevute dagli altri utenti. Quindi un utente virtuoso è capace di mantenere questo campo a 0.

2.6.2 Registrazione Audio

La relazione che c'è tra Utente e Registrazione Audio è quello di creazione, poiché banalmente l'utente crea la registrazione. Le cardinalità sono presto spiegate. Ogni utente può scegliere se registrare o meno qualcosa. Tecnicamente un utente potrebbe anche rispondere e basta, senza fare registrazione/domande. E non ha limiti di domande. Per cui la cardinalità da Utente a Registrazione Audio è 0,n. Guardandola dal lato invece della registrazione, è necessario affermare che una registrazione appartenga ad uno ed un solo utente, da cui la cardinalità 1,1. Di seguito la descrizione degli attributi.

- **ID_RegistrazioneAudio:** è un intero ed è la chiave primaria. Si sarebbe potuto rendere come chiave primaria anche l'incrocio tra ID_Utente (importato esternamente) e Data. Tuttavia è stato ritenuto più comodo creare un identificatore proprio per l'entità, anche ai fini della

ricerca delle registrazioni e del *download* (sopra descritto) di esse.

- **Link**: è un campo stringa. Viene creato univocamente al momento della creazione di un nuovo file audio, quindi di un nuovo record nella tabella. Il link serve per collegare ciò che viene registrato nel database al file vero e proprio. Per essere più chiari, un link all'interno del database può essere del tipo xxxyyyzzz.mp4; all'interno del server, nell'apposita cartella, vi sarà il file xxxyyyzzz.mp4 che è l'audio registrato dall'utente. Per cui andando all'indirizzo www.davidecanducci.altervista.org/regs/xxxyyyzzz.mp4 si ottiene il file registrato.
- **Genere**: è un campo stringa contenente il genere dell'audio scelto dall'utente al momento della registrazione. L'utente non ha possibilità di aggiungere nuovi generi ma solo di scegliere tra quelli già esistenti. Questo semplifica enormemente le ricerche, annullando tutti i possibili errori di battitura e le possibili sfumature (oramai infinite) tra i generi musicali.
- **Risolta**: questo sarebbe un campo da rappresentare con un *boolean true/false*, tuttavia sarebbe stato più complicato da gestire, per cui ho optato per un semplice intero. 1 se la registrazione ha trovato risposta, 0 altrimenti. Ogni nuova registrazione ha questo campo inizializzato a 0.
- **Data**: questo campo rappresenta esattamente la data di creazione dell'audio. Come già spiegato, poteva essere chiave dell'entità assieme all'attributo ID_Utente appositamente importato, ma una chiave numerica è estremamente più facile da gestire. Sarà utile al momento del *download* della lista delle registrazioni, per

mostrare il momento in cui ogni registrazione è stata pubblicata.

- **Titolo**: il campo "stringa" titolo viene utilizzato per mostrare appunto una specie di "anteprima" sempre nel momento del *download* della lista delle registrazioni. L'inserimento o meno di questo campo è stato oggetto di una riflessione, in quanto se una domanda è per sua stessa natura testuale, per un audio vale la regola opposta: non si può rappresentare. Per cui un utente che inserisce un audio è possibile che sia nella posizione di non saper dare un titolo a questo. Tuttavia potrebbe aiutare sapere magari dove l'ha sentito, o qualunque altra informazione voglia dare l'utente in anteprima. Ed inoltre nella lista (visibile) degli audio almeno questi differiscono decisamente l'uno dall'altro.
- **Segnalazioni**: anche la registrazione può essere segnalata, in caso in cui contenga parole o versi non consoni (come magari possono essere gli insulti ad altri utenti). In questo caso l'utente creatore stesso viene segnalato.

2.6.3 Risposta

Questa sarà la tabella più corposa di tutta l'applicazione, in quanto conterrà tutte le risposte date a tutte le registrazioni. In caso in cui l'applicazione, come si suol dire, prenda seriamente "piede", sarà obbligatorio rivedere questo aspetto del *database*. Di seguito gli attributi.

- **ID_Risposta**: anche in questo caso, come per ID_Registrazione, si sarebbe potuta creare una chiave esterna (questa volta completamente esterna) tra Utente e Registrazione. Invece si è optato ancora una volta per un identificatore interno, sotto forma di intero, che

potesse garantirmi più velocità nelle ricerche. E' un campo intero e auto-incrementante.

- **Testo:** è il campo stringa contenente il testo vero e proprio della risposta. Le risposte non sono pensate per essere eccessivamente lunghe, per questo i caratteri sono limitati ad un numero accettabile.
- **Vincente:** come per l'attributo "Risolta" dell'entità "Registrazione", questo campo sarebbe stato perfetto come *boolean*, in quanto sarebbe stato *true* in caso la risposta fosse stata selezionata come, appunto, "vincente", o *false* altrimenti. Per i motivi già descritti in precedenza è stato scelto di rendere questo attributo un numero intero. Il valore 1 quindi sostituisce *true* mentre il valore 0 sostituisce *false*.
- **Segnalazioni:** si terrà conto delle segnalazioni (negative) ricevute per ogni risposta. Anche se una risposta ottiene tante segnalazioni, vale sempre come +1 alle segnalazioni dell'utente.

Con la spiegazione del *database* è terminato il secondo capitolo della tesi, quello riguardante l'intera progettazione del *software* e di tutto ciò che esso ha bisogno per sopravvivere e prosperare (si spera) nel tempo. Sarebbe comunque folle pensare che il progetto sia già completo e perfetto, per cui la speranza rimane quella di riuscire a migliorarlo, magari anche grazie a nuove tecnologie che nel frattempo potranno dare quella spinta in più per evolvere l'applicazione.

CAPITOLO 3

Dopo aver affrontato nel capitolo precedente la progettazione riguardante What's This Song, è il momento di esporre nel dettaglio lo sviluppo che sta alla base dell'applicazione.

L'applicazione prevede uno sviluppo sia lato *client* che lato *server*. Come espressamente dichiarato nel primo capitolo, What's This Song sarà un applicativo diretto a dispositivi mobili (*smartphones* su tutti), aventi il sistema operativo Android. Per sviluppare il *software*, è stato utilizzato l'ambiente di sviluppo **Android SDK** con l'IDE **Eclipse**. Le parti dinamiche dell'applicazione sono state scritte utilizzando il linguaggio di programmazione **JAVA**, mentre le parti statiche in **XML** (questa è una dualità che caratterizza tutte le applicazioni *Android*). Per quanto riguarda la parte Server è stato utilizzato il linguaggio **PHP** e per la gestione del *database* **MySQL.**
Lo sviluppo dello scheletro del software, sarà fedele a quanto riportato nell'ambito della progettazione. Alcuni dettagli riguardo implementazioni di determinate sezioni e algoritmi, possono invece subire sostanziali modifiche, dettate dalla necessità di efficienza richieste da un dispositivo mobile.

3.1 SCHERMATA DI SPLASH ED INIZIO

Prima di procedere con questo primo paragrafo, è utile fare una premessa notazionale. Un' attività (Activity) in Android è da intendersi sostanzialmente come una schermata dell'applicazione, definibile graficamente attraverso appositi *layouts* grafici dichiarati in file aventi estensione .xml; le attività vengono create come oggetti che estendono la **classe Activity** (definite nelle API LEVEL 1), da cui si

ereditano proprietà e metodi. Quindi tutte le schermate principali dell'applicazione avranno un nome (che sarà il più possibile descrittivo) composto dal termine "Activity", ad esempio "*LoginActivity*", "*RecordAudioActivity*" e così via. La prima schermata visualizzata, quando l'utente preme l'icona dell'app *What's This Song* sul proprio dispositivo, è detta "*SplashActivity*". Questa attività ha il compito principale di caricare in ***background*** determinati dati, che saranno utili nel prosieguo dell'esperienza applicativa. In questo caso caricherà delle cartelle nella memoria per contenere gli audio, un piccolo "database" privato creato tramite la classe *SharedPreferences* già disponibile grazie ad *Android* e si occuperà di controllare se l'utente ha già eseguito il *login*, poiché come già spiegato senza autenticazione l'applicazione non permette nulla. Se dunque il *login* risulta effettuato, l'applicazione mostrerà le operazioni da poter eseguire, altrimenti mostrerà l'activity dedicata appunto al *login*.

Figura 3.1

In figura 3.1 viene rappresentata l'immagine "**splash**" dell'*activity* "*SplashActivity*" attuale. Questa immagine verrà visualizzata per un tempo limitato (si parla di 2-3 secondi), per essere sicuri che il riconoscimento dell'utente sia stato fatto correttamente e questi abbia tutte le possibilità di utilizzare al meglio l'applicazione. Verrà inoltre controllato se c'è rete internet. In caso in cui non ci sia l'applicazione non ha senso che si avvii, quindi viene notificato all'utente.

Nel caso in cui lo *smartphone* sia connesso ad internet si aprono comunque due possibilità relative a ciò che può accadere successivamente. La prima possibilità riguarda il fatto che non è stato trovato un login effettuato precedentemente, quindi l'applicazione forzerà l'ingresso dell'*activity* "***LoginActivity***". In caso in cui invece l'utente sia stato riconosciuto, sarà possibile la visualizzazione di tutte le funzioni relative all'applicazione. Saranno analizzate dunque queste due possibilità in paragrafi dedicati, poiché fanno parte dell'implementazione vera e propria dell'app.

3.2 IMPLEMENTAZIONE DELL'ACCOUNT UTENTE

3.2.1 IMPLEMENTAZIONE DEL LOGIN

La gestione del login viene dunque fatta tramite la classe già menzionata "*LoginActivity*". Ma da cosa è composta questa classe? È stato già sottolineato che le applicazioni create per funzionare su sistemi *Android* hanno la particolarità di avere una gestione dinamica ed una gestione statica; in particolare è estremamente utile collegare i fogli xml di *layout* alle classi per gestire meglio le caratteristiche grafiche delle varie componenti. Queste componenti altro non sono che **TextView** e **EditText** che, opportunamente distribuite, creano una classica ***form*** di ***login***. Entrambe le sopracitate componenti fanno parte della sezione riguardante i *widgets* di

Android (*android.widget*), e mentre la prima fornisce una semplice casella di testo non editabile dall'utilizzatore, la seconda fornisce una casella (di default vuota, ma questo è personalizzabile) nella quale l'utente ha la possibilità, selezionandola, di scrivere del testo "sua sponte". E' dunque importante creare un *layout* fortemente intuitivo col quale l'utente potrà interagire inserendo il proprio *username* e la propria *password*; in aggiunta a tutto ciò va inserito un pulsante (Button, sempre da *android.widget*) con la scritta "Login", il quale si occuperà che, alla pressione, l'*activity* possa dialogare col *server*. In figura 3.2 un riassunto visivo di quanto appena detto. Alla pressione del pulsante "Non sono registrato" dovrà apparire la schermata di registrazione.

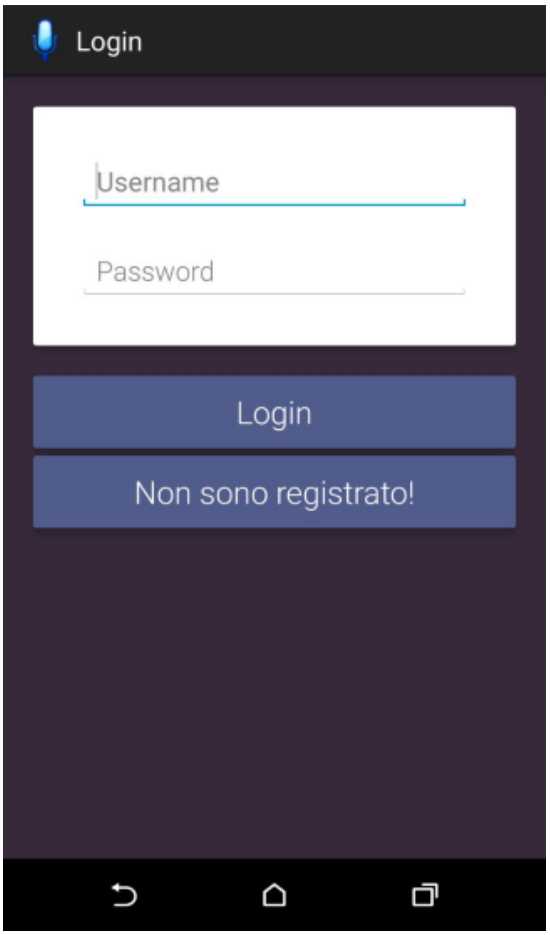

Figura 3.2

3.2.2 EXCURSUS SU ASYNCTASK

Dall'inizio della scrittura della tesi si è praticamente sempre parlato di continui dialoghi fra l'applicazione ed il *server*, ed è dunque venuto il momento di descrivere come faccia esattamente l'applicazione a spedire al *server* le informazioni fondamentali al proseguimento delle operazioni e dunque ad ottenere risposta. Verrà portato l'esempio del *login*, che è pressoché uguale a tutte le altre comunicazioni via *internet*, differente (in minima parte) solo rispetto all'*activity* che si occupa di inviare al *server* il file con l'audio registrato. Va detto anzitutto che gli sviluppatori Android hanno (ben) pensato che l'utente potesse preoccuparsi del fatto che lo schermo del proprio *smartphone* potesse "congelarsi" in una schermata per diversi secondi. Generalmente infatti, quando accade, è sintomo che c'è qualcosa che non sta funzionando o che sta funzionando male. Ma nel caso in cui ci sia poco campo (o magari una connessione molto lenta) è verosimile pensare che l'invio di un *file* anche di piccole dimensioni via internet possa occupare diverso tempo. Pertanto lo schermo dello *smartphone* (e lo *smartphone* stesso) dovrebbero rimanere fermi in attesa per tutto il tempo necessario alla trasmissione del file (sia esso in uscita ma anche in entrata). E questo, detto francamente, è assolutamente inaccettabile. Per cui i suddetti sviluppatori hanno reso impossibile la comunicazione via *internet* tramite il processo (*task*) principale, obbligando i programmatori delle varie applicazioni ad avvalersi di una classe chiamata "***AsyncTask***". Per il corretto uso di questa classe, è necessario crearne un'altra, privata, all'interno della classe-*activity* di riferimento, che estenda la suddetta. In questo caso è stata creata, all'interno della classe *LoginActivity*, la classe privata *LoginAsyncTask*. Estendendo dunque la classe *AsyncTask* se ne può fare il cosiddetto ***Override*** dei metodi, vale a dire utilizzare dei metodi della

classe madre personalizzandoli. Tramite questo stratagemma dunque è possibile inviare e ricevere dati dalla rete senza il pericolo di rendere l'applicazione inutilizzabile, questo perché, come dice il termine stesso, viene creato un apposito processo asincrono, che agisce sotto quello principale, completando tutti i lavori che gli sono richiesti. Viene dunque proposto qualche pezzo di codice che semplificherà la spiegazione.

```
private class LoginAsyncTask extends AsyncTask<String, Integer, HttpResponse> {

            @Override
            protected HttpResponse doInBackground(String... params) {

                  return postData(params);
            }

…
```

Come appena descritto, la classe privata che si deve andare ad implementare dovrà estendere la classe *AsyncTask*, e per farlo dovrà dunque selezionare i "*types*" nececessari alla corretta implementazione. Nel mio caso il primo *type* è una stringa, il secondo è un intero e il terzo è un oggetto *HttpResponse*. Il metodo ***doInBackground*** viene attivato all'esecuzione del task, che avviene, come già detto, alla pressione del pulsante "Login" tramite questo codice:

```
new LoginAsyncTask().execute(eTusername.getText().toString(),
                                    eTpassword.getText().toString());
```

Fondamentale quindi il metodo *execute()* con al suo interno i parametri prelevati dalle EditText sopra descritte, quella dedicata allo *username* e quella dedicata alla *password*. Questi parametri vengono inseriti dentro l'*array* di stringhe *params*, e questo giustifica il primo *type*, *String* appunto. Tornando al metodo *doInBackground*, che come si può ben notare è un *Override*, esso è di fondamentale importanza poiché si occupa di inviare i parametri al server tramite *post http*. E'

preferibile sorvolare tutta la descrizione su come essa avviene, ma rimane importante sottolineare il fatto che il metodo *postData*, richiamato immediatamente da *doInBackground*, restituisca una *HttpResponse*. E con questo viene giustificato il terzo *type* della classe. Questo oggetto contiene la risposta del *server*, che si è occupato di controllare se i dati inseriti dall'utente sono giusti e, in caso affermativo, di iniziare una "***session php***", tramite la quale mantenere l'utente "loggato" grazie ai ***cookies***. Dunque è necessario prelevare il *cookie* inviato dal *server* (che come spiegato qualche pagina fa ha una durata più che sufficiente) e salvarlo nel *database* privato creato con *SharedPreferences*. Ma esattamente a chi, il metodo, restituisce la *response*? Precisamente al metodo:

```
@Override
protected void onPostExecute(HttpResponse result)
```

E' facilmente notabile che anche questo è un metodo sovrascritto, e si attiva nel momento in cui il metodo *doInBackground* termina il suo compito. Per cui è proprio il metodo ***onPostExecute*** che si occupa di gestire la risposta del *server* e di memorizzare tutto (*cookie*, crediti, codice utente) in *SharedPreferences*. Tutto ciò è indispensabile in quanto essendo il sopradescritto metodo *execute()* un metodo che lavora in maniera asincrona rispetto all'applicazione, per avere la certezza di operare dopo aver ricevuto la risposta del *server* è necessario farlo all'interno del metodo *onPostExecute*, che analizzerà, leggendo la risposta del server, se tutto è andato per il verso giusto o meno. A questo punto il *login* è presto completato. Nelle prossime comunicazioni col *server* sarà sufficiente prelevare il *cookie* dal database locale ed inserirlo all'interno degli *headers* delle prossime richieste *post* (o *get*) *http*. L'unico *type*

dell'*AsyncTask* non descritto è l'*Integer*: esso nel è utile semplicemente per visualizzare nello schermo una *ProgressBar* che mostrerà all'utente che il procedimento di *login* è in atto. Si è deciso di spendere molte parole su questo tipo di procedura poiché, come già scritto, si ripete in continuazione per tutto il resto del programma. Si sarebbe potuto utilizzare anche il la classe ***JSON*** per inviare al *server* i dati dell'autenticazione, ma per un semplice *login* questo sistema è più rapido ed intuitivo da implementare. Tuttavia sarà utilizzato *JSON* lato *server-response* e questa è la lieve differenza già descritta all'inizio del sotto paragrafo tra i vari tipi di comunicazione con il *server* delle *activities* implementate.

3.2.3 IMPLEMENTAZIONE DELLA REGISTRAZIONE

Una rapida descrizione la merita anche la registrazione, che è pressoché identica al *login* ma aggiunge qualche metodo interessante. E' possibile giungere alla registrazione semplicemente premendo sopra la scritta "**Non sono registrato**" (come in figura 3.2). La pagina di registrazione si presenta in maniera estremamente simile a quella di *login*, quindi sempre con *TextViews* ed *EditTexts*. L'utente dovrà dunque inserire i propri dati all'interno delle caselle *EditText* e confermare la registrazione. Vi sono tuttavia delle possibilità che necessitano di essere prese in considerazione: lo *username*, come già puntualizzato, deve essere univoco. Quindi il *server* dovrà controllare l'esistenza o meno nel *database* di uno stesso "*nickname*". E stessa cosa vale per l'indirizzo *email*. Per cui, nel momento in cui l'utente preme sopra "Registra", prima di procedere con la vera e propria registrazione saranno fatti una serie di controlli, ai fini di garantire il perfetto funzionamento della registrazione nel server. Il primo dei controlli sarà il seguente: verrà

esaminato l'indirizzo *email* tramite il metodo `public static boolean isEmailValid(String email)` se sia un indirizzo reale o un *fake*. Il secondo sarà la banale comparazione tra le due *passwords* inserite ed il terzo sarà la comunicazione col *server* che tramite la risposta informerà l'applicazione se esistono già *username*, *email* od entrambi. Se questo controllo risulterà positivo, verrà lanciata un'altra *post http* e questa volta le credenziali dell'utente verranno inserite all'interno del *database*. Di ritorno dal *server*, verrà registrato localmente il codice dell'utente (cosa che all'utente non importa ma ai fini dell'*app* è utilissimo).

3.3 CUORE DELL'APPLICAZIONE

3.3.1 REGISTRA – ASCOLTA – INVIA AL SERVER

Finora è stato descritto quanto accade all'apertura dell'applicazione nel caso in cui la stessa si accorga che non esiste alcun utente autenticato. D'ora in avanti invece sarà descritto tutto ciò che accade post-autenticazione, quindi l'implementazione di ciò che permette di fare il *software*.

L'utente dunque avrà la possibilità di scegliere cosa fare una volta terminati i processi in *background* durante la visualizzazione della *splash image*. La schermata presenterà vari *buttons* che indurranno l'utente a scegliere cosa vuol fare in quel momento. Verrà inoltre visualizzato il suo nome utente ed i crediti che gli appartengono. La prima opzione per l'utente è l'*activity* riguardante la registrazione degli audio. Una volta premuto l'apposito pulsante, la schermata apparirà più o meno come in figura 3.3 (rimango vago poiché non escludo cambiamenti anche in un immediato futuro). Il pulsante "*Registra*" farà da protagonista in rosso in una schermata che prevede altri due pulsanti, uno dedicato all'ascolto di ciò che è stato appena registrato e l'altro all'invio dell'audio al *server*. Per motivi ovvi renderò

impossibile l'ascolto della registrazione o l'invio al server prima che l'audio sia stato registrato. Saranno presenti inoltre un menu a tendina per scegliere il genere dell'audio e una casella di testo nella quale inserire il titolo dello stesso.

Figura 3.3

La registrazione dei *files* audio verrà effettuata tramite oggetti della classe *MediaRecorder* (android.media.MediaRecorder); lo stesso pulsante permette di registrare e fermare la registrazione. Questo significa che l'utente deve essere avvisato visivamente dell'inizio della registrazione, per cui l'aspetto stesso dei pulsanti cambierà (così come mostrato dalla figura 3.4). L'oggetto *MediaRecorder* che viene creato all'inizio dunque del metodo `private void`

`startRecording()` e dunque la registrazione vera e propria inizia tramite il comando `mRecorder.start();` (notare che *mRecorder* è l'oggetto *MediaRecorder* creato appositamente). Alla pressione del pulsante dunque il *layout* subisce piccole variazioni (figura 3.4) indicanti i secondi di registrazione e la possibilità di stoppare quest'ultima e l'applicazione è pronta a proseguire con lo "stop" al momento della successiva pressione del *button*. Il tutto è gestito tramite variabili `boolean mStartRecording` che si aggiorneranno ogni qualvolta il pulsante verrà premuto. Dunque allo stop verrà richiamato il metodo `stopRecording()` che rilascerà l'oggetto *mRecorder* appena creato terminando il *file audio* (che, ricordo, avrà estensione .mp4).

Figura 3.4

E' utile aggiungere inoltre che questa classe si occupa di non permettere all'utente di creare *files* audio che poi non verranno più utilizzati. In particolare, è senza dubbio pratico tenere in memoria, nell'apposita cartella ***myregs*** creata al momento della *SplashActivity*, solo le registrazioni che l'utente sceglie poi di condividere. Per cui tutte le volte che un utente, nella stessa sessione, decide di registrare nuovamente qualcosa, il *file* appena creato viene distrutto. In caso contrario rimane in memoria, come già descritto.

Appena fermata la registrazione sarà quindi possibile scegliere se condividere l'audio appena creato inviandolo al *server*. Anzitutto però ho pensato fosse indubbiamente utile riascoltare ciò che si è registrato, per cui sarà possibile premere, oltre al *button* di **invio**, anche il *button* di **ascolto**.

Il pulsante dedicato all'ascolto farà (strutturalmente) grosso modo ciò che fa il pulsante della registrazione, solo con la differenza che mentre quest'ultimo registra, il primo crea un oggetto di classe `MediaPlayer` (*android.media.MediaPlayer*) per eseguire le istruzioni che gli vengono richieste. Questo oggetto dunque è in grado di far riprodurre audio e fermarli. Lo stop è ancora una volta comandato dall'utente, alla pressione dello stesso pulsante che nel frattempo avrà cambiato forma. All'interno del metodo `private void startPlaying()` è presente un ***listener***. Questi oggetti, che pur non avendo mai citato hanno già fatto "involontariamente" la loro comparsa, altro non sono che degli "ascoltatori". E possono mettersi in ascolto di praticamente qualunque cosa, come ad esempio un *button*, un particolare gesto che l'utente fa nello schermo e anche di un oggetto di classe ***Media Player*** come il seguente che sta all'interno del già citato metodo *startPlaying()*:

```
mPlayer = new MediaPlayer()
```

Per cui è interessante notare la creazione del seguente metodo:

```
mPlayer.setOnCompletionListener(new OnCompletionListener() {

                    @Override
                    public void onCompletion(MediaPlayer mp) {

                    …
```

E' importante sottolinearlo per il semplice motivo che se il pulsante Registra/Stop Registrazione è modificabile al tocco dell'utente, per il pulsante *play* questa cosa non vale, in quanto una volta finito l'audio possono anche non esserci interazioni fisiche con l'utente. Questo metodo viene dunque incontro, e specifica cosa bisogna fare alla fine dell'ascolto della registrazione.

L'invio dell'audio al *server* infine è gestito tramite i già visti *AsyncTask*. Merita un focus l'interno del metodo *doInBackground* questa volta, perché inviare un file non è come inviare stringhe. E' stata utilizzata, per questo, la classe ***MultipartEntityBuilder*** che permette la creazione di un *builder* il quale può contenere diversi tipi di dati, ed essere spedito tramite una semplice *post http* (*org.apache.http.entity.mime.MultipartEntityBuilder*). Per utilizzare questa classe è stato necessario inserire le librerie *httpcore* e *httpmime*, delle quali sono state scelte le versioni (rispettivamente) 4.3.3 e 4.3.6. Viene proposto quindi un piccolo frammento di codice relativo all'uso del builder:

```
MultipartEntityBuilder builder = MultipartEntityBuilder.create();
builder.setMode(HttpMultipartMode.BROWSER_COMPATIBLE);
builder.addBinaryBody("songName", new File(songName[0]));
builder.addTextBody("user_code", songName[1]);
builder.addTextBody("genere", songName[3]);
builder.addTextBody("titolo", songName[4]);
httpPost.setEntity(builder.build());
```

E' semplice evincere da questo frammento che il builder si occupa di associare le varie chiavi ai vari parametri, contenuti dentro l'array *songName*. Si può facilmente notare che non esiste *songName[2]* ma questo per il semplice fatto che in quella cella è presente il *cookie*, che andrà inserito nell'*header* più avanti.
Una volta inviato l'audio al *server* l'app tornerà al menu iniziale e sarà di nuovo in attesa di un successivo tocco dell'utente.

3.3.2 – DOWNLOAD LISTA REGISTRAZIONI

È già stato sottolineato il fatto che per la vita dell'applicazione è necessario che la comunità collabori il più possibile. Per iniziare tuttavia ad ascoltare le registrazioni altrui ed a rispondere, si è pensato di suddividere gli audio per categorie, in particolare (inizialmente) per genere. Per cui la prima schermata che comparirà al momento della pressione sul pulsante "Ascolta gli audio della comunità e rispondi" sarà relativa alla scelta del genere. Essa è banalmente composta da uno ***spinner***, vale a dire l'oggetto fornito da *Android* (sempre reparto *widget*) per mostrare a video un menu a "tendina" dal quale è possibile selezionare le diverse possibilità di scelta che lo sviluppatore ha voluto fornire. Attualmente i generi disponibili sono una decina circa, ma evolvendosi l'applicazione, potranno essere aggiunti senza problemi. Una volta scelto il genere dunque si può procedere premendo l'apposito *button* della pagina e tramite questa riga di codice

```
new ChooseGenreAsyncTask().execute((String) spGenere.getSelectedItem());
```

il solito *AsyncTask* provvederà ad inviare al *server* la stringa prelevata appunto dallo *spinner*. Il *server* dunque provvederà tramite una ***query*** di questo tipo

```
SELECT R.ID_Registrazione, R.Data_Registrazione, R.Titolo, R.Link, R.Utente,
R.Genere, U.Username
FROM RegistrazioneAudio R, Utente U
WHERE R.Utente = U.ID_Utente
AND Genere = '$genere' AND Risolta = '0'
```

il *server* comunicherà col *database* interno per prelevare tutto ciò che ritiene utile alla realizzazione della lista con le registrazioni, che poi mostrerà nella schermata successiva. Per la restituzione di tutti questi dati si è optato per il formato *JSON* (*JavaScript Object Notation*). Esso, data la sua semplicità d'uso e la sua leggerezza, è perfetto per il salvataggio di tanti piccoli dati come semplici stringhe ed interi. Col comando

```
echo json_encode($json);
```

è dunque semplicissimo ottenere dentro un oggetto di tipo *HttpResponse* tutti i dati codificati in formato JSON (N.B. la variabile *$json* contiene un *array* con il risultato della *query*).
Per cui sarà poi necessario "decodificare", tramite un comando del tipo

```
private String jsonResult;
jsonResult=inputStreamToString(response.getEntity().getContent()).toString();
```

tutto il contenuto della risposta. Una volta ottenuti tutti i dati utili alla creazione di una ipotetica lista con titolo degli audio in primo piano, data e utente creatore è possibile passare alla schermata successiva, chiudendo questa ***ChooseGenreAndDownloadAcvitity*** e passando direttamente a ***ManageRegListActivity***.

Tramite quindi il metodo *putExtra(String name, String value)* proprio degli ***intent*** (android.content.Intent) è possibile, alla creazione di una nuova *activity*, inviare a quest'ultima la stringa contenente il *JSON* sopra descritto. All'apertura della nuova *activity* dunque, dopo aver provveduto ad estrarre la stringa dall'*intent*, l'app procede a "disegnare" la lista delle registrazioni; ogni riga sarà composta da titolo della registrazione, data ed autore della stessa. Propongo di seguito i due *layouts* delle *activities* appena descritte.

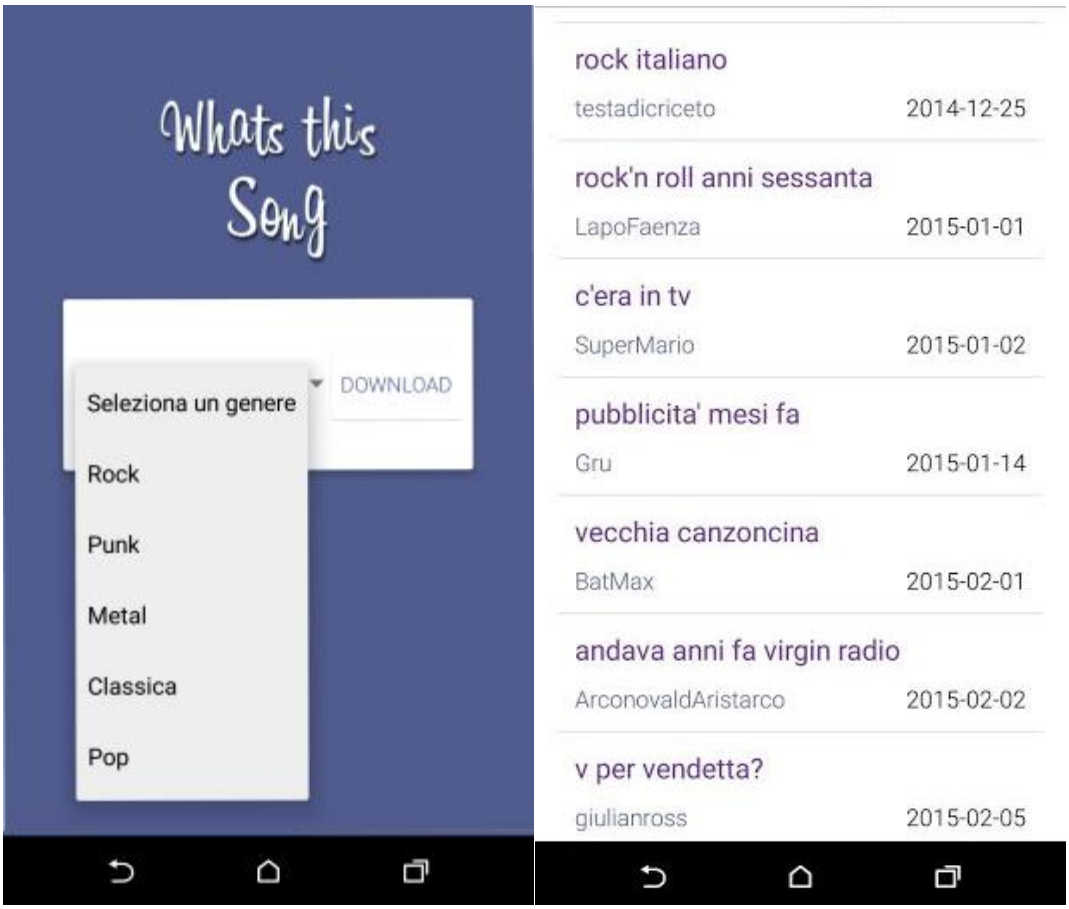

Figure 3.5 e 3.6

Il metodo che si occupa di "disegnare" la lista quindi andrà a scorrere tutte la stringa codificata *JSON* e a riempire l'apposita *listaRegistrazioni*. Menzione particolare merita quest'ultima, che a livello di codice è implementata in questa maniera:

```
private ArrayList<AnswerModel> listaRegistrazioni =
                                                new ArrayList<AnswerModel>();
```

dove *AnswerModel* è il modello di "domanda" (registrazione) della lista. Si è pensato di implementare questo modello tramite una classe apposita atta semplicemente a contenere i dati utili (ma anche quelli attualmente superflui) alla lista. Questa quindi conterrà i codici di utente e registrazione, il titolo, la data, il *link* e il genere di questa e lo *username*. Il metodo *crea_elemento_lista* quindi si occuperà di riempire la lista e verrà invocato ad ogni nodo del *JSON*, esattamente in questo modo:

```
listaRegistrazioni.add(crea_elemento_lista(id_Registrazione,
                                                data_Registrazione, link, titolo,
                                                   id_utente, genere, username));


…


private AnswerModel crea_elemento_lista(long id_Registrazione,
                    String data_registrazione, String link, String titolo,
                    long id_utente, String genere, String username) {
            AnswerModel singola_riga_registrazioni = new AnswerModel();
            singola_riga_registrazioni.id_registrazione = id_Registrazione;
            singola_riga_registrazioni.data = data_registrazione;
            …
            return singola_riga_registrazioni;
      }
```

Una volta completata la lista sarà sufficiente inserirla dentro un ***adapter*** appositamente creato per contenerla. Gli *adapter* sono elementi utilissimi per la personalizzazione delle liste, e nel mio caso ho esteso la classe *BaseAdapter* (android.widget.BaseAdapter) per crearne uno secondo i bisogni del *software*, che potesse appunto contenere le informazioni utili all'utente utilizzatore dell'applicazione. Alla lista sarà associato un *listener*, che starà in ascolto di eventuali pressioni per ogni elemento che contiene. Alla pressione quindi, l'applicazione si prepara a cambiare schermata ed a far partire una nuova *activity*, in particolare quella dedicata

all'ascolto della registrazione selezionata ed all'interazione con essa (intesa come possibilità di rispondere).

3.3.3 ASCOLTA E RISPONDI

A questo punto non rimane che descrivere la sezione riguardante il vero apporto che ogni utente dà alla comunità: vale a dire l'*activity ListenAndAnswerActivity*.
Alla pressione dunque di un elemento della lista presente in ***ManageRegListActivity***, viene caricata una pagina contenente un *Button* che permette di fare il *download* della registrazione selezionata. Questo avviene per mezzo di un *AsyncTask* e tramite il metodo `getInputStream()`. Più precisamente, la pagina che verrà visitata sarà suggerita dal *link* precedentemente salvato, ed in essa sarà presente esclusivamente il file musicale con estensione .mp4. Per cui l'***InputStream*** procederà al *download* del *file* per intero, e lo memorizzerà in un *file* interno alla memoria del telefono, nell'apposita cartella precedentemente creata. Durante il *download* è stato scelto di implementare un particolare aspetto visivo di *Android*, che prende il nome di *ProgressDialog* (android.app.ProgressDialog). Esso rappresenta in breve il progresso del *download* in corso, indicandolo con un valore tra 1 e 100 ed una grafica intuitiva. Esso viene inizializzato, come la *ProgressBar*, all'interno del metodo *onCreate* e mostrata nel metodo *onPreExecute* dell'*AsyncTask* (denominato *DownloadSongAsyncTask*), dopo la pressione del pulsante ovviamente. Il suo progresso viene gestito invece nel metodo ***OnProgressUpdate***. Appena il file sarà stato scaricato il *ProgressDialog* sparirà e partirà il metodo *play()* precedentemente visto nell'*activity* ***RecordAudioActivity***. Ho optato inoltre per mostrare la lista delle (eventuali) risposte alla registrazione esclusivamente *post-download*.

Tutte le risposte saranno mostrate tramite una lista, che questa volta mostrerà gli attributi *titolo*, relativo alla registrazione, e *username*, relativo all'utente che ha risposto. La lista stessa implementa la possibilità di selezionare una delle risposte ed eventualmente segnalarla come "offensiva". In questo caso il *database* verrà informato della segnalazione, sempre tramite *AsyncTask*. Terminata la risposta, l'applicazione forzerà il ritorno alla lista di registrazioni precedentemente descritta.

Figure 3.7 e 3.8

In merito alla segnalazione come offensiva, occorre precisare la decisione di utilizzare il seguente metodo:

```
@Override
	public void onCreateContextMenu(ContextMenu menu, View v,
			ContextMenuInfo menuInfo) {

		super.onCreateContextMenu(menu, v, menuInfo);
		menu.setHeaderTitle("Cosa vuoi fare");
		menu.add(1, 1, 1, "Segnala come risposta offensiva");
		if (admin1) {
			menu.add(1, 2, 2, "Marca come risposta esatta!");
		}
	}
```

Esso, una volta associata la lista delle risposte, concede all'utente di tenere premuto su una risposta della lista, mostrandone le opzioni. Se l'utente è amministratore (cosa che spiegherò più avanti) gli comparirà anche la seconda opzione, vale a dire "Marca come risposta esatta". Altrimenti attualmente è possibile solo la segnalazione della risposta. I numeri presenti all'interno del metodo ***add()*** riguardano il gruppo interno al menu, l'identificatore della scelta e l'ordine col quale verranno visualizzati. Tramite infine l'*Override* del metodo

```
@Override
public boolean onContextItemSelected(MenuItem item)
```

è possibile definire esattamente cosa fare alla scelta di una opzione del menu.

3.3.4 GESTIONE DELLE PROPRIE REGISTRAZIONI

La schermata relativa alla gestione delle proprie registrazioni è pressoché identica a quanto appena descritto. Per facilitare l'uso agli utenti infatti la scelta è ricaduta su un *layout* del tutto similare. Dunque premendo l'apposito pulsante della *home* si giunge direttamente alla lista delle proprie registrazioni del tutto simile alla già descritta

lista dell'*activity* *ManageRegListActivity*. In questa *activity*, denominata ***MyOwnRegsActivity***, verrà tuttavia visualizzato solo titolo e data della registrazione, in quanto l'utente creatore è ovviamente il medesimo che utilizza l'app. Quindi tramite una semplice *query* al server, del tipo

```
SELECT *
FROM RegistrazioneAudio
WHERE Utente = '$user_code'
ORDER BY ID_Registrazione
```

si è in grado di ottenere tutte le informazioni che "interessanti". Una volta ottenuta la lista è quindi semplice riutilizzare l'*activity* *ListenAndAnswerActivity* per poter interagire con gli altri utenti in ogni singola registrazione. Questa volta però è necessario che l'*app* sappia che chi apre la registrazione è il creatore. Per ovviare a questo problema è stata creata una variabile *boolean* che, inserita con l'apposito metodo all'interno dell'*intent*, viene estratta dall'*activity* successiva già descritta in precedenza. E con questo è spiegato anche il frammento di codice relativo al metodo *onCreateContextMenu*, in particolare il controllo `if (admin1)` che permette di creare, all'interno del menu, anche l'opzione "marca come risposta corretta". Un altro *AsyncTask* si occuperà quindi infine, in caso di pressione su quest'ultima opzione, di comunicare al *server* che la risposta selezionata è corretta.

Con quest'ultimo impegnativo paragrafo, si conclude il capitolo 3, dedicato allo sviluppo dell'applicazione What's This Song. Ho cercato di spiegare in maniera più chiara possibile le tante meccaniche che si celano alla base di un progetto apparentemente semplice.

CAPITOLO 4

Ho pensato di concludere mia tesi con una breve riflessione su quello che può essere lo sviluppo in ottica futura dell'applicazione. Più volte sono stati fatti accenni nei precedenti capitoli riguardo alle piccole migliorie possibili a proposito del *software* che ho implementato. E' arrivato il momento di fare un po' di ordine e di descrivere ciò che è nei progetti e ciò che sogno di creare nel futuro.

4.1 PICCOLE MIGLIORIE

- "**Ho dimenticato la password**": uno dei "problemi" degli internauti del giorno d'oggi è sicuramente il fatto di avere minimo 4-5 *accounts* sparsi in giro per il *web* (*Facebook*, *Twitter*, *email*, forum…). E talvolta sono richieste *passwords* che devono variare a distanza di mesi, oppure può capitare che un sito voglia caratteri maiuscoli e minuscoli ma non speciali mentre un altro vuole anche questi ultimi. E mentre gran parte degli utenti usano sempre la stessa *password*, c'è una minoranza che preferisce variare. Ma alla reinstallazione dell'applicazione o al cambio di telefono (eventi rari) il *software* si comporta come se fosse stato aperto per la prima volta. Per cui ricordarsi la *password* è indispensabile ed al contempo una missione impossibile. Un servizio che permette l'invio di una nuova *password* od eventualmente della vecchia tramite *email* sarebbe dunque estremamente utile. Occorrerebbe quindi implementare un *server* di posta elettronica che si occupi di questa evenienza.
- **Sicurezza della password**: l'attuale *password* come già scritto a più riprese ha la semplice necessità di essere lunga non meno di 8 caratteri. E dal momento che non vi è un limite di tentativi, è plausibile pensare che prima

o poi sia possibile trovarla e rubarla. Per cui due piccole migliorie possono essere le seguenti:

  - **Necessario aumento della complessità**, magari tramite inserimento di lettere maiuscole e minuscole, caratteri speciali e numeri;
  - **Limite di tentativi accettati da parte del server**, vale a dire che dopo un tot di tentativi il server blocca l'accesso e permette solo successivamente altri tentativi, allungando di molto i tempi utili al furto.

- **Audio eccessivamente lunghi**: attualmente non esistono restrizioni riguardo alla lunghezza (in secondi) di una registrazione. Ma se essa prenderà piede sarà un problema non da poco, per cui è già stato pensato di inserire un tempo limite di 10 secondi per audio, onde evitare audio interminabili. Anche perché facendo qualche *test* un audio della durata di 20 secondi è capace di occupare ben più di 30 *Kilobytes*. Quindi cento registrazioni già occuperebbero quasi 3 *Megabytes*. Un'enormità considerata l'attuale portata del *server* (del quale spenderò parole più avanti). Per cui per come la vedo attualmente 10 secondi saranno più che sufficienti per poter esplicare il proprio motivetto.
- **Intro & help**: per quanto l'applicazione sia estremamente intuitiva, sarebbe utile creare una sorta di introduzione invitando l'utente (tramite apposite animazioni) all'uso corretto dell'applicazione. Una pagina di aiuto inoltre, con i contatti dello sviluppatore potrebbe aiutare la crescita dell'app.
- **Segui/salva le registrazioni altrui**: l'applicazione necessiterebbe di un modo estremamente facile e veloce per controllare le registrazioni che ci interessano, quindi manca ancora della possibilità di salvare in un reparto a parte gli audio altrui.

- **Stelline / voti**: una registrazione è fatta particolarmente bene, come mettere in evidenza questo fatto? Poi: perché favorire solamente coloro che rispondono correttamente? Oppure come premiare un utente che è stato particolarmente disponibile? Sarebbe interessante inserire delle stelline che valutino la registrazione, oltre a dei voti (semplici *likes* ma anche *dislikes*) per ogni singola risposta (proprio in stile *Yahoo Answers*).
- **Ottimizzazione spazio interno dello smartphone**: qualche riga fa è presenta una piccola analisi a proposito dello spazio occupato dagli audio. Ed ovviamente è nei migliori interessi occupare meno memoria possibile del terminale dell'utente. Per cui sarebbe utile implementare un'opzione che permetta al proprietario dello *smarphone* di eliminare ogni audio salvato tramite un semplice pulsante. Ovviamente si tratterebbe di una scelta non reversibile.
- **Possibilità di ricercare le registrazioni ("a random")**: questa opzione facilmente inseribile permetterebbe agli utenti di non ricercare più per genere, ma in maniera completamente casuale. Sarebbe utile inoltre permettere agli utenti di ricercare il titolo di un audio, o magari l'autore, facendo sì che l'app restituisca una lista di audio come quelle già descritte, ma secondo un'altra chiave di ricerca.
- **Ban a tempo per gli utenti tossici**: le segnalazioni attualmente comportano avvertimenti tramite *mail* (*mail* da inviare privatamente e singolarmente). E alla reiterazione dei "reati" scatta il ban (vale a dire la disattivazione/cancellazione dell'account). Più giusto sarebbe un ban a tempo, per permettere agli utenti scorretti di "redimersi" e crescere (ove possibile). Anche questo è un fine della comunità.

4.2 GRANDI MIGLIORIE / EVOLUZIONI

- **Facebook/Twitter**: è già in corso l'opera di integrazione dell'applicazione con Facebook, vale a dire la possibilità di *login* all'interno della stessa. Tuttavia le autenticazioni diverrebbero due all'interno della stessa app, vale a dire una per *Facebook* e l'altra ovviamente per *What's This Song*. Un passo avanti sarebbe il permettere all'utente di "loggarsi" esclusivamente con *Facebook*, anche se la cosa comporterebbe comunque una registrazione all'interno del *server* di *What's This Song* che l'utente non vede (per gestire crediti, segnalazioni e quant'altro). *Facebook* (e *Twitter*) sarebbero inoltre fondamentali per il cosiddetto *sharing* (condivisione). Esso è già in via d'implementazione per il servizio offerto da *Zuckerberg*, e la condivisione permette ora come ora un semplice *post* all'interno della propria pagina *Facebook* con un link all'audio, dato che ogni *browser web* (anche mobile) è in grado di aprire un *file* .mp4 oramai. Altra cosa però sarebbe l'auto-pubblicità, vale a dire l'invito a coloro che cliccano sul *post* condiviso a scaricare l'applicazione, aumentando in breve tempo il numero degli utilizzatori. Per *Twitter* sarebbe utile proporre un servizio come *#whattheflora*, dunque un semplice *hashtag* che, richiamando il nome dell'applicazione, inglobasse all'interno dello stesso *tweet* la condivisione dell'audio.
- **Passaggio ad un server privato**: sarà fondamentale trasportare tutto il *database* in un *server* privato e più capiente (l'attuale è registrato su http://www.davidecanducci.altervista.org. Un nuovo *server* permetterebbe molte più personalizzazioni e molta più libertà. Potrei anche aggiungere all'interno dello

stesso la creazione di un lettore di *files* audio completamente a tema.

- **Push notifications:** un servizio della massima importanza è senza dubbio quello delle *push notifications*, vale a dire le notifiche che mette a disposizione *Android* per avvertire l'utente di eventuali novità nelle *apps* installate. Avrei due strade disponibili per implementare questo tipo di servizio che ai fini della mia applicazione si rivelerebbe utilissimo:
  - **Creazione di un service:** per fare sì che l'applicazione possa comunicare col *server* anche senza che l'utente la utilizzi sarebbe necessario implementare un *service*, vale a dire un processo che in *background* invii e riceva piccole quantità di dati, e che in casi particolari (spetta allo sviluppatore decidere) si permetta di creare notifiche per l'utente, anche se questi nel frattempo sta facendo tutt'altro. In sostanza è come per le *chat* come *Whatsapp* o la stessa *Facebook*. Nel mio caso i "casi particolari" di cui parlavo prima potrebbero essere semplicemente un commento alla propria registrazione, oppure una risposta ad un proprio commento, o ancora l'evento in cui una propria risposta è stata ritenuta vincente.
  - **Android push notifications:** un servizio reso disponibile da Google, che permette sostanzialmente ciò che ho appena descritto riguardo al *service*, solo che si aggrega ai servizi già esistenti di *Google*, vale a dire *Gmail, G+* eccetera. Quindi invece che creare un nuovo *service* posso pensare di unirmi a quelli già gestiti da *Google*.
- **Ottimizzazione delle prestazioni:** essendo ancora in fase di sviluppo, l'applicazione presenta sicuramente dei rallentamenti che più avanti è possibile cercare di

ridurre al minimo, anche perché maggiore velocità implica anche (spesso) maggior semplicità e quindi maggior possibilità d'uso.

- **Miglioramenti grafici:** l'attuale *layout* è studiato per essere di semplice utilizzo, ma non è ancora perfettamente conforme al *material design* proposto da *Google* all'uscita di *Android* 5.0. Un grande passo avanti a probabilmente necessario è quello di rendere il *layout* il più accattivante possibile, rimanendo nelle specifiche suggerite dall'azienda di *Mountain View*.
- **Sviluppo per altri devices e altre piattaforme:** la descrizione del *software*, sin dal primo capitolo, gravita attorno al fatto che l'applicazione è fatta appositamente per *smartphones* e a tutti gli effetti è stata sviluppata per quello. Sarebbe comunque importante avere la possibilità di svilupparla anche per *tablets* e, perché no, eventualmente anche per altri dispositivi che nel frattempo si saranno evoluti (*smartwatch*?). Questo richiede sicuramente tanto tempo, eppure è qualcosa che rimane nei piani. Un progetto ancora più ambizioso sarebbe rendere disponibile *What's This Song* anche per terminali *Apple*, quindi forniti di un sistema operativo completamente diverso come *iOS*. E non dispiacerebbe sicuramente ampliare il mercato anche a *Windows phone* (ed a quel punto passare alla versione *desktop* sarebbe un buon passo ulteriore).
- **Sviluppi commerciali:** forse questo punto meriterebbe un capitolo a parte per la mole di possibilità esistenti, ma ritengo inutile spendere troppe parole sull'argomento. E' innegabile però che il sogno sia quello di vedere un giorno l'app *What's This Song* installata su ogni terminale (o quantomeno il numero più alto possibile). Più che un sogno è un'utopia, ma il pensare che questo progetto possa essere la pietra d'angolo per il mio

futuro lavoro è un'ambizione alla quale non è possibile non pensare. Quindi un aspetto che interessante è senza dubbio quello commerciale, o, per meglio dire, le risposte alla domanda "come posso monetizzare questa idea?". Vi sono più modi su *Android* di guadagnare grazie alle proprie app, e vanno prezzo iniziale (esempio: il costo dell'app è 2€) al servizio a pagamento (alcune notifiche arrivano solo a coloro che pagano per il servizio in più). Vi sono inoltre i banner pubblicitari, che permettono un guadagno in base a quanti utenti utilizzano l'applicazione; vi sono le versioni "pro", vale a dire versioni che hanno qualcosa in più (non solo in termini di servizi); c'è la possibilità di mettere in vendita pacchetti con *layouts* differenti e tante altre cose. In pratica i modi per unire l'utile al dilettevole sono tanti, e vanno studiati tutti in egual misura.

CONCLUSIONI

Termina qui la stesura della tesi, o almeno della sua parte puramente descrittiva. La progettazione del capitolo 2 unita all'implementazione del capitolo 3 forniscono sufficienti informazioni per la comprensione del *software*. A livello didattico, ho approfittato di questa esperienza per praticare diversi linguaggi di programmazione appresi durante il mio cammino universitario, oltre che a varie metodologie per lo sviluppo e l'ingegnerizzazione del software. Si è cercato inoltre di esplorare nuovi tipi di implementazioni, andando a sviluppare tipologie di comportamento del *software* (per me) nuove, come l'invio e la ricezione via *mobile* di interi *files*.

Ho avuto modo di dirlo in precedenza, ma ci tengo a sottolineare che questa applicazione non vuole essere "fine a se stessa". La stragrande maggioranza delle tesi di laurea muore al momento dell'esposizione; nel caso di *What's This Song* farò di tutto affinché ciò non avvenga, impegnandomi ad evolvere e migliorare un prodotto che è già parzialmente innovativo (non esistono ancora servizi stabili e conosciuti in questo campo). Il mercato è in continua espansione, e personalmente spero di poter coronare il mio sogno di commercializzazione dell'applicazione, spero di poter condividere questo progetto con qualcuno che possa credere nel futuro di questa app, spero in ultimo che questa aiuti chiunque abbia la fastidiosa sensazione che si prova quando si ha un motivetto in testa ma non ne si riesce a stabilire la fonte.

BIBLIOGRAFIA

Massimo Carli, "*Android 4. Guida per lo sviluppatore*", APOGEO

Android developer,
http://developer.android.com/about/dashboards/index.html
(Figura 1.2)

Statista, http://www.statista.com/ (Figura 1.0)

Wikipedia, *Smartophone*,
http://it.wikipedia.org/wiki/Smartphone

Wikipedia, *Yahoo Answers*,
http://it.wikipedia.org/wiki/Yahoo! Answers (Figura 1.1)

Wikipedia, *Sistema operativo per dispositivi mobili*,
http://it.wikipedia.org/wiki/Sistema operativo per dispositivi mobili

Wikipedia, *Android*,
http://it.wikipedia.org/wiki/Android

RINGRAZIAMENTI

Sarebbe veramente ingiusto da parte mia concludere davvero senza i dovuti ringraziamenti. In primo luogo grazie al mio relatore, il dottor Mirko Ravaioli, persona terribilmente disponibile, paziente e competente, sempre aperto a nuove idee e pronto a consigli. Grazie anche a tutti i professori che, a loro modo, hanno saputo darmi tutti qualche lezione di vita prima che degli argomenti trattati nei loro corsi.
Il grazie più grande lo devo tuttavia sicuramente ai miei genitori, che mi hanno sempre sostenuto nella buona e nella cattiva sorte senza uccidermi d'ansia o di pressioni, ed alla mia famiglia tutta. Un grazie va anche ai miei amici tutti, che non starò a citare uno per uno altrimenti dovrei scrivere un altro capitolo.

Mi preme comunque fare i nomi di persone senza le quali non avrei terminato questo percorso: grazie ad Ezio, più fratello che amico da molti anni ormai, col quale ho condiviso le esperienze più importanti della mia vita (università a parte); grazie a Michelangelo "*the unstoppable*" Arcuri, amico praticamente dall'iscrizione a Scienze dell'Informazione (allora si chiamava così) che ha condiviso con me gran parte delle soddisfazioni e delle delusioni di questa lunga carriera universitaria; grazie a Massimo "maccola" Neri, per la pazienza, l'amicizia e la tenacia dimostrata nei miei confronti, del quale posso aggiungere che se non è alla NASA è solo colpa sua; grazie a Giorgio "testadicriceto" Scioni, che in caso di bisogno è sempre presente, con sorriso e battuta pronta. Con questi ultimi tre ho davvero passato quasi tutti gli anni universitari, e conoscono meglio di me quanto è stata dura, quanta è stata la fatica e quante le "vittorie" riportate. Se avessi dovuto scrivere una lettera al me stesso del futuro, 7 anni fa, non avrei potuto sperare in amicizie

così belle e sincere. Abbiamo iniziato l'università convinti chi di smontare un PC e saperci mettere i pezzi nuovi finiti i tre anni, chi di saper lavorare perfettamente con *Adobe Photoshop* alla vista del corso "Metodi numerici per la grafica" (che ovviamente si è dimostrato tutt'altro). Penso di poter dire che ho cominciato che ero un adolescente ed ho terminato che sono un uomo. Ancora con la u minuscola magari. Sono quindi giunto alla fine, anche se questa parola non fa parte del mio dizionario, in quanto la fine non è che un nuovo inizio. Spero di poter ricominciare con voi.
Concludo il tutto con un paio di frasi che la fanno da padrona in questo periodo; una è di *Sam Ewing*, che ho fatto mia (tra l'altro condivisami proprio da Michelangelo): "non sono le ore che ci metti, è quello che ci metti nelle ore". Guardandomi indietro posso dire che le ore sono state davvero piene. Alle volte un po' troppo di svago, ma in generale un "pieno" positivo. Un caro prof mi disse poi qualche anno fa: "l'università non è una corsa a tempo: ognuno ci mette quanto ci deve mettere". Aggiungo io: dato che il nostro tempo è limitato, spero solo di non avercene messo troppo. Anche se a conti fatti, sono felice così.

Davide