Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea Triennale in Informatica

AGGREGAZIONE DI EVENTI GEOLOCALIZZATI

Tesi di Laurea in Architettura degli Elaboratori

Relatore:
Chiar.mo Prof.
VITTORIO GHINI

Presentata da:
ANDREA CATALINI

I Sessione
Anno Accademico 2011/2012

Introduzione

Ci apprestiamo a discutere dello sviluppo di un'applicazione mobile su piattaforma Android, sistema operativo di casa Google. È questo un settore fortemente in espansione, che ha visto le luci della ribalta circa un decennio fa grazie ad Apple che ha fatto fortuna con i suoi iPhone.
Mi sono interessato a sviluppare un'applicazione che facesse da contenitore di ciò che il mercato intorno a noi, geolocalizzandoci col GPS, ha da offrire. In questo modo si ha raccolto, nel palmo di una mano, un gran numero di offerte e negozi che offrono particolari servizi magari per periodi limitati. L'uso per cui è concepita l'applicazione non è da relegare solo a negozi che hanno qualcosa di materiale da vendere, ma anche ad esempio un comune potrebbe voler pubblicizzare la sua festa di paese che si tieni in quei giorni. Con tale applicativo può e senza dover avviare quelle grandi campagne di volantinaggio che tutti conosciamo. Lo stesso volantinaggio che ogni volta dovrebbe sostenere un locale notturno per poter pubblicizzare le proprie serate.

In questo documento si illustrerà tutto il processo di sviluppo che c'è stato dietro, a partire dal concepimento dell'idea per arrivare alle considerazioni sulle possibili implementazioni di migliorie future. Nel primo capitolo verrà data un prospettiva generale sui sistemi operativi mobile attualmente in commercio e spiegherò perché ho preferito Android alle altre piattaforme. A quel punto passerò ad illustrare l'idea su cui mi concentrerò per terminare con l'analisi dello stato attuale in materia di applicazioni simili.
Nel secondo capitolo illustrerò quali sono gli obiettivi che mi sono preposto di sviluppare per questa applicazione dando enfasi a che cosa porto come

innovazione rispetto alla concorrenza.
Si passa quindi nel terzo capitolo, dopo una breve ma dovuta introduzione tecnica alla struttura di Android, ad illustrare come è stato strutturato il progetto, anche attraverso l'ausilio di frammenti di codice per aiutare la comprensione. Terminerò il capitolo illustrando servizi e ide impiegati per portare a termine il lavoro.
Nel quarto ed ultimo capitolo studierò la qualità del mio operato. Prima però spiegherò quali sono, a mio avviso, i parametri importanti che devono essere valutati in applicazioni di questo genere; definirò quindi una metrica sulla quale baserò la successiva analisi della qualità.
Il documento verrà concluso dall'illustrazione delle possibili implementazioni future che renderebbero più performante e funzionale l'applicazione.

Indice

Elenco delle figure

Capitolo 1

Scenario

1.1 Piattaforma

1.1.1 Piattaforme in commercio

Oggigiorno le maggiori piattaforme mobile che si trovano in commercio sono Blackberry OS, Android, IOS e Windows Phone, di minor rilievo troviamo il neo-nato Bada e l'ormai dismesso Symbian. Ogni sistema ha, ovviamente, i suoi "pregi" e "difetti", ma tutti condividono la grandiosa idea di un negozio virtuale (generalmente chiamato market) in cui l'azienda stessa ma anche sviluppatori esterni hanno la possibilità di mettere in vendita le loro applicazioni.

Windows Phone e Bada, sistema proprietario Samsung, sono stati subito scartati perché troppo giovani e quindi ancora poco maturi e soprattutto con troppa poca utenza. Blackberry OS negli ultimi anni ha perso un po' il suo "splendore" iniziando ad essere un po' snobbato e quindi con utenza, almeno nel contesto italiano, non molto elevata. Symbian è ormai "arrivato alla frutta" per obsolescenza e semi abbandono dello sviluppo da parte dello stesso creatore (Nokia) che l'ha relegato a dispositivi di "fascia bassa". Non restava che scegliere tra IOS, sistema operativo di casa Apple, ed Android, O.S. di casa Google. IOS è sicuramente il sistema più maturo che si trova

attualmente in commercio, con molto marketing alle spalle che l'ha portato, nonostante l'elevato prezzo, ad essere presente in ogni parte del mondo. Ha però un grande problema: costo da sostenere elevatissimo dovuto al fatto che bisogna possedere sia un iPhone abbastanza recente, così da avere il sistema operativo il più vicino possibile all'ultima versione, sia un Mac dato che l'SDK viene rilasciato dalla casa solo per i propri computer.

Android, seppure più giovane di IOS, ha raggiunto in fretta un alto livello di maturità e qualità per via di due ragioni: la prima è che comunque è un progetto open-source che, come di consueto, ha il supporto di un' eccellente comunità e la seconda è che dietro questo ambizioso progetto c'è una grande azienda che ha sempre dimostrato grandi potenzialità. Inoltre questo O.S. riesce a girare discretamente anche su piattaforme di basso costo, ed è stato proprio questo uno dei suoi maggiori punti di forza che ha dato vita alla rapidissima diffusione che tutti conosciamo.

1.1.2 Quale scegliere?

Alla luce di quanto esposto la scelta non è stata poi troppo difficile anche se forse non si dimostrerà essere la più semplice.

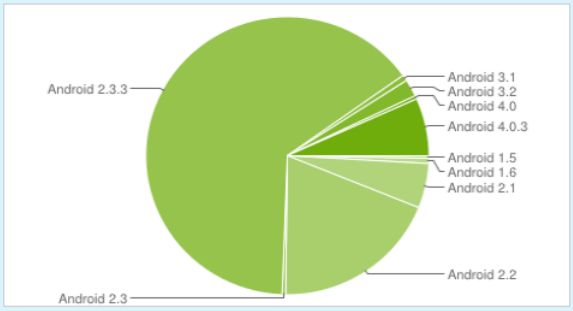
Il sistema scelto è stato Android. Il motivo preponderante di questa scelta, ma non l'unico, è stato il budget richiesto: molto basso a paragone dell'alternativa Apple. Ha comunque molti altri pregi di cui parlare, infatti a differenza di IOS questa volta l'SDK viene rilasciato da Google per qualsiasi piattaforma, Linux compreso, ed è inoltre ben integrato col noto IDE eclipse, i manuali ufficiali sono ben curati e sempre affiancati da validissimi esempi ed inoltre ha avuto un'altissima diffusione visto che riesce ad avere delle prestazioni discrete anche su dispositivi di fascia bassa, i famosi entry-level.

Se da un lato la grande diffusione che ha avuto questa piattaforma è stata il motivo di maggior successo, dall'altro lato tutto ciò ha ha dato vita al problema più grande di Android ovvero la sua eccessiva frammentazione sia in caratteristiche hardware che in versione di sistema operativo installata. Il problema è talmente grande da aver spinto Google stessa a scendere in campo

per aiutare quanto più possibile gli sviluppatori, raccogliendo con regolarità dati statistici a risoluzioni e dimensione degli schermi, versioni di sistema operativo che vanno per la maggiore e modelli più venduti.

Questo problema non è da sottovalutare ma comunque i vantaggi sono talmente elevati che hanno avuto comunque la meglio su questa frammentazione.

Tanto per avere un'idea dell'entità del problema riporto alcuni grafici che Google ha messo a disposizione:

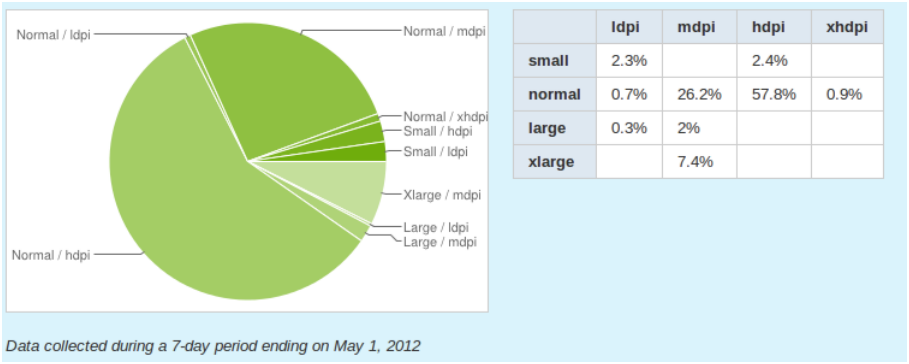


Platform	Codename	API Level	Distribution
Android 1.5	Cupcake	3	0.3%
Android 1.6	Donut	4	0.6%
Android 2.1	Eclair	7	5.2%
Android 2.2	Froyo	8	19.1%
Android 2.3 - Android 2.3.2	Gingerbread	9	0.4%
Android 2.3.3 - Android 2.3.7		10	64.6%
Android 3.1	Honeycomb	12	0.7%
Android 3.2		13	2%
Android 4.0 - Android 4.0.2	Ice Cream Sandwich	14	0.4%
Android 4.0.3 - Android 4.0.4		15	6.7%

Data collected during a 14-day period ending on June 1, 2012

Figura 1.1: Percentuale che ogni versione del firmware Android ha in relazione ai device immessi sul mercato

`http://developer.android.com/resources/dashboard/platform-versions.html`



	ldpi	mdpi	hdpi	xhdpi
small	2.3%		2.4%	
normal	0.7%	26.2%	57.8%	0.9%
large	0.3%	2%		
xlarge		7.4%		

Data collected during a 7-day period ending on May 1, 2012

Figura 1.2: Densità, in percentuale, degli schermi dei dispositivi Android sul mercato

`http://developer.android.com/resources/dashboard/screens.html`

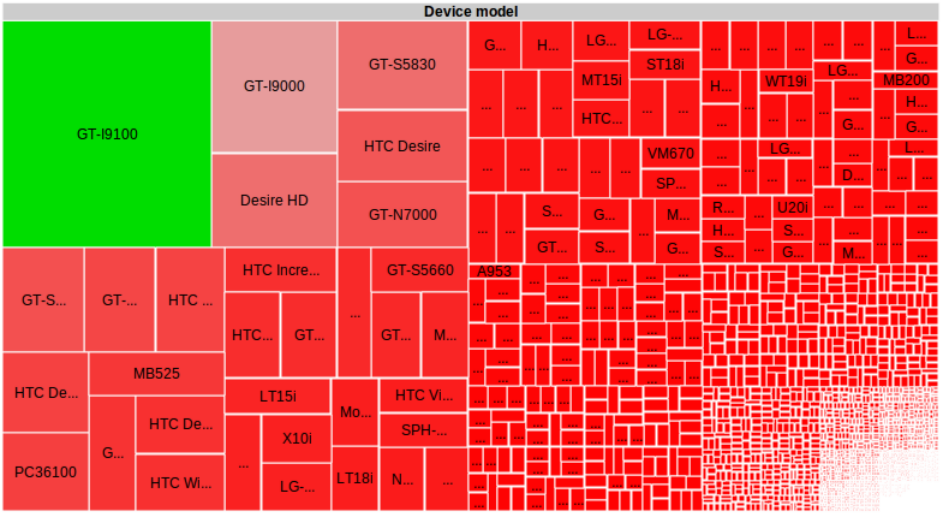


Figura 1.3: Fetta di mercato per ogni device Android rilasciato

`http://opensignalmaps.com/reports/fragmentation.php`

1.2 L'idea

La prima versione di Android viene rilasciata il 23/09/2008 e dopo 4 anni dalla nascita all'interno del suo market si trovano ormai applicazioni per tutti i gusti, a volte persino di dubbia utilità!

La mia attenzione è stata subito catturata da un particolare dispositivo hardware di cui ormai tutti i dispositivi Android odierni sono dotati, sto parlando del ricevitore GPS. Le applicazioni in cui è stato impiegato sono molteplici e disparate: si va dal geo-tagging nelle fotografie al classico navigatore per arrivare all'indicazione di quali servizi/interessi/persone ci sono nelle nostre vicinanze. È stata proprio quest'ultima applicazione che ha solleticato molto la mia fantasia tanto da spingermi ad avvicinarmi al mondo di Android non più solo come utente ma anche come sviluppatore.

L'idea grezza che subito mi è balzata in testa è stata: perché non raggruppare insieme quante più "cose" possibili? Dove le "cose" di cui sto parlando potrebbero essere servizi, attività, eventi, promozioni, sconti, feste e qualsiasi altra cosa un utente possa ricercare e un commerciante possa invece voler pubblicizzare e successivamente vendere. Detto così sembrerebbe più un grande canale di spam che una vera e propria applicazione ma non è così, infatti tutte queste "cose" non sarebbero mostrate a caso ma filtrate attraverso apposite impostazioni, dove la più importante sarà la posizione dell'utente ed il range entro il quale si ha interesse/possibilità di muoversi ed immancabilmente cosa effettivamente la persona voglia visualizzare nel proprio dispositivo, quindi cosa sta cercando in quel momento.

Tirata giù questa idea striminzita bisognava cercare di aumentare le funzionalità dell'applicazione in modo da renderla il più utile, "alla moda" ed invitante possibile. Quindi non si poteva far a meno di un lato social e per questo in qualche modo si sarebbe dovuta integrare una qualche funzionalità che permettesse alle persone di relazionarsi tra loro, scambiare opinioni in tempo reale, postare foto e quant'altro un social network contemporaneo mette a disposizione; per rendere il servizio utile si sarebbe dovuto sviluppare un canale di feedback con relativi commenti. Molte altre cose utili ma di minor entità si sarebbero potute aggiungere, come la possibilità di salvare la posizione di un luogo visitato che si è apprezzato etc. , ma non conoscendo ancora cosa c'era sul mercato e cosa la piattaforma Android mettesse veramente a disposizione non mi sono spinto oltre, sicuro del fatto che di idee

nuove sarebbero nate durante lo sviluppo e altre sarebbero state scartate per via di obsolescenza, inutilità o chissà quale altro problema.

Bisognava un attimo lasciar perdere la questione delle idee, tanto il cardine dell'applicazione era ormai stato partorito; bisognava piuttosto, prima ancora di ispezionare il market alla ricerca della concorrenza, vedere se comunque un'applicazione del genere avesse potuto effettivamente attirare sia l'utente che il venditore. La questione era delicata in quanto chi effettivamente dava un senso a tutto ciò sarebbe stata propria l'utenza, come d'altronde avviene in un social network. I vantaggi per la parte commerciale erano chiari: avrebbero avuto un canale di comunicazione diretta col pubblico senza dover passare per spam noioso e fastidioso o dover ricorrere a chissà quali spese in stampe di volantini riempiendo piazze intere per pubblicizzarsi, riassumendo avrei dato loro una sorta di vetrina virtuale che avrebbe fatto risparmiare parecchi soldi per la questione del marketing ed in più sarebbe stata più efficace come pubblicità in quanto chi la visualizzerebbe lo fa solo perché veramente interessato in quel preciso momento. Il problema che più mi preoccupava a riguardo dei sellers era: col tempo avrebbero poi continuato a tenere aggiornata la loro pagina? Per far si che questo accadesse avrei dovuto creare un'interfaccia web per il venditore il più semplice e veloce possibile, in cui con pochi click avrebbe aggiunto un evento e inviatolo a chi ne era interessato. Passando invece dal lato dell'utente, che sarebbe stato il potenziale cliente, si incontrava un problema non molto diverso in quanto anche se il canale fosse pieno di offerte ma nessun utente lo guardasse allora tutto il lavoro fatto sarebbe risultato inutile. Avrei quindi dovuto porre particolare attenzione oltre che alla qualità dei servizi offerti anche alla progettazione l'interfaccia, cercando di disegnarla il più semplice e accattivante possibile.

Mancava solo da pensare a quali possibili problematiche tecnologiche avrei potuto incontrare sul cammino dello sviluppo. Avrei avuto sicuramente bisogno di un database dove inserire tutti i dati degli utenti e dei venditori, bisognava quindi, almeno per la fase iniziale e di test, cercare qualche database

gratuito; usando un database avrei avuto bisogno di conoscere qualche linguaggio di scripting per gestirlo e fare le query inoltre trattando con dispositivi con banda limitata e byte di connessione contati un formato per lo scambio di dati che sia il più leggero possibile come ad esempio Json; dovendo poi realizzare una pagina web con il form per l'inserimento del prodotto/servizio da parte del venditore avrei avuto bisogno di un dominio per poter appoggiare il mio sito e ovviamente un'immancabile conoscenza del java visto che è il linguaggio per la programmazione in Android.

1.2.1 Stato dell'arte

A questo punto, avendo in mano un'idea strutturata, non mancava altro che iniziare a sondare il terreno per scoprire quanto di essa aveva già effettivamente visto la luce.

Dalle varie ricerche effettuate in giro per la rete sono riuscito ad estrapolare 3 correnti principali che le applicazioni in commercio hanno seguito:

"social-network based": in questa tipologia le applicazioni raggruppano tutte le attività commerciali che sono intorno all'utente in un raggio più o meno ampio; viene messa a disposizione la possibilità di lasciare un voto al locale e magari anche un commento sul servizio, precisando però che il locale non ne sa niente del feedback del singolo utente ma la votazione rimane all'interno dei database del servizio in questione; per ogni locale l'utente vi può anche caricare un foto commentandola; viene inoltre data la possibilità di condividere la propria posizione, mostrando quindi in quale locale si è passata la serata o in quale negozio si è fatto acquisti.

Fin qui sembra una semplice guida per l'utente su quale locale/negozio è migliore della concorrenza sulla base dei voti lasciati dagli utenti ma non è così, infatti viene data una più o meno forte enfasi al lato sociale: ogni singola applicazione della categoria ha un propria rete sociale sviluppata dalla stessa azienda, questa rete però varia in funzionalità

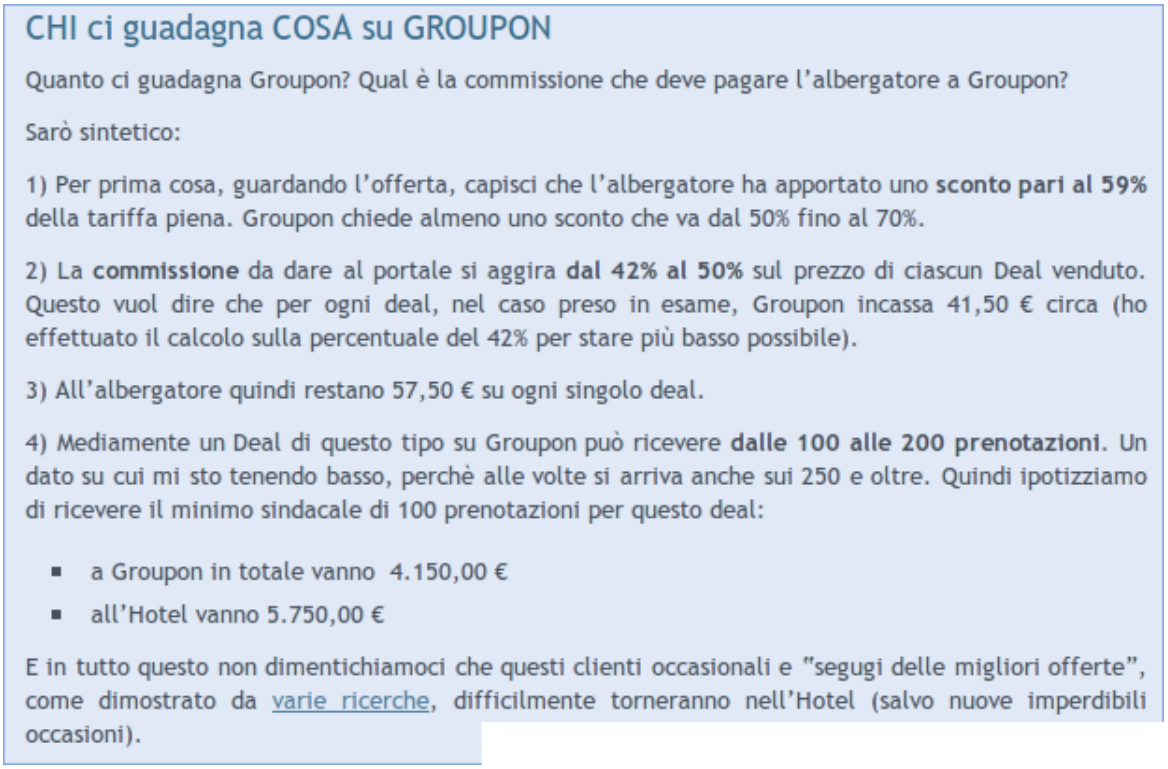
data dall'investimento che l'azienda si è potuta permettere nello sviluppo. Per sopperire a questo problema diverse applicazioni hanno scelto, oltre che di mantenere la loro piccola rete sociale, di aprirsi anche a social network famosi come Facebook; questa scelta, oltre che fornire un servizio più completo e funzionante agli utenti, dovrebbe anche portare un po' pubblicità gratuita (suppongo) al servizio. Alcune applicazione hanno anche azzardato un sezione per le offerte che i negozianti fanno, ma da quanto ho potuto constatare, almeno in Italia, sono pressoché sezione vuote o quasi.

In ultima analisi ho notato che le applicazioni più grandi, quelle che hanno già di per sé un social network robusto com'è quello di FourSquare, hanno introdotto quell'aspetto che è chiamato "gamification" ovvero l'utilizzo delle meccaniche e dinamiche dei giochi come livelli, punti o premi, in contesti esterni al gioco per creare più interesse. Nello specifico FourSquare mette a disposizione tutta una serie di badge da vincere e titoli da guadagnare come ad esempio chi si è registrato di più in un certo luogo o cose simili.

"offers based": per la maggior parte questa categoria di applicazioni sono nate prima come servizio online e, grazie al successo ottenuto, generato in seguito l'applicazione per le maggiori piattaforme. Tale applicazione però effettivamente non ha delle particolari funzionalità se non mostrare in modo più agevole a smartphone e tablet quello che già il sito web della stessa azienda mostra a tutti gli utenti.

Passiamo ora a spiegare in cosa consistono effettivamente questi servizi; tre sono le parti in gioco e cioè i clienti, i venditori e il servizio che si frappone fra le prime due, che altro non fa che mettere a disposizione una sorta di mercato virtuale, come se fosse un e-commerce. Non è però un normale e-commerce come tutti noi lo concepiamo (vedi ebay), infatti questi servizi vengono definiti "gruppi di acquisto". Quello che fanno è mostrare ai clienti delle vantaggiosissime offerte che un'attività commerciale decide di fare per i più svariati motivi. Per far capire

come ciò sia possibile è riportato in figura 1.4 un esempio trovato in rete. Il caso specifico in questione è "Groupon" ma le differenze che si potrebbero incontrare con un servizio simile dovrebbero essere solo i margini di guadagno ripartiti diversamente tra le parti in gioco.



CHI ci guadagna COSA su GROUPON

Quanto ci guadagna Groupon? Qual è la commissione che deve pagare l'albergatore a Groupon?

Sarò sintetico:

1) Per prima cosa, guardando l'offerta, capisci che l'albergatore ha apportato uno **sconto pari al 59%** della tariffa piena. Groupon chiede almeno uno sconto che va dal 50% fino al 70%.

2) La **commissione** da dare al portale si aggira **dal 42% al 50%** sul prezzo di ciascun Deal venduto. Questo vuol dire che per ogni deal, nel caso preso in esame, Groupon incassa 41,50 € circa (ho effettuato il calcolo sulla percentuale del 42% per stare più basso possibile).

3) All'albergatore quindi restano 57,50 € su ogni singolo deal.

4) Mediamente un Deal di questo tipo su Groupon può ricevere **dalle 100 alle 200 prenotazioni**. Un dato su cui mi sto tenendo basso, perchè alle volte si arriva anche sui 250 e oltre. Quindi ipotizziamo di ricevere il minimo sindacale di 100 prenotazioni per questo deal:

- a Groupon in totale vanno 4.150,00 €
- all'Hotel vanno 5.750,00 €

E in tutto questo non dimentichiamoci che questi clienti occasionali e "segugi delle migliori offerte", come dimostrato da varie ricerche, difficilmente torneranno nell'Hotel (salvo nuove imperdibili occasioni).

Figura 1.4: Esempio delle meccaniche che sottostanno alle applicazioni appartenenti alla categoria "offers based"

`http://www.danilopontone.it/web-marketing-turistico/chi-ci-guadagna-cosa-su-groupon-ecco-un-esempio-pratico/`

"geo-chat based": quest'ultima categoria ha molti punti in comune con la "social network based". Ad esempio infatti entrambe le categorie visualizzano le attività commerciali, divise per categorie, che si trovano nelle vicinanze dell'utente; danno la possibilità di fare il check-in in un luogo per condividerlo con gli amici; postare foto e relativi commenti.

Hanno ancora dei social network integrati più o meno performanti e funzionali ma allo stesso tempo cercano di mantenere comunque allacci con i social network più famosi; non sempre viene invece mantenuta quell'idea di canale di feedback da condividere con altri utenti su luoghi o negozi.
Fin qui tutto sommato non sembrerebbe formare una categoria a sé, ma piuttosto sembrerebbe essere pressoché la stessa cosa della "social network based" ed invece non è così. Quello che veramente differenzia le due categorie non è tanto le differenti funzionalità proposte, che poi di differenti ce ne sono ben poche, ma è principalmente la filosofia con la quale ognuna è stata concepita; cerco di spiegarmi meglio: mentre la prima punta a tenere i nostri amici aggiornati su cosa stiamo facendo, dove siamo stati, cosa ci è piaciuto di un certo luogo, se lo rifaremmo etc. la seconda offre comunque la stessa funzionalità ma inoltre permette, a chi vuole, di cercare nuove persone intorno a se che magari condividono interessi, attività, gusti o qualsiasi altra cosa la gente cerchi nell'instaurare nuove amicizie. Per rendere questa funzionalità possibile nella mappa oltre ai luoghi di interesse vengono mostrate anche le persone che sono in quella zona; anche in questo caso, come per le attività commerciali cercate, di solito si cerca di mettere a disposizione dei filtri per evitare di avere la mappa piena di puntini, magari filtrando per valori come interessi, età, sesso etc.. Alcune permettono anche di chattare in tempo reale con persone che non conosciamo ma che semplicemente sono nel rage di ricerca inserito.
C'è da dire anche che non sono molte le applicazioni che effettivamente fanno tutto questo perché la maggior parte si disinteressano di tutta la questione delle attività commerciali e si indirizza solo verso il generare nuove amicizie, flirt o quant'altro uno cerca in nuove relazioni. Tengo a precisare che queste applicazioni non le considero assolutamente appartenenti a tale categoria.

Capitolo 2

Obiettivi

Come in un'azienda che si rispetti bisogna stilare degli obiettivi chiari e ben definiti che descrivono cosa ci si dovrà aspettare dal prodotto finale.

2.1 Caratteristiche dell'applicazione

Cerchiamo ora di definire meglio quell'idea grezza che ho esposto nel precedente capitolo. L'applicazione deve mettere a disposizione del proprio utente tutto ciò che "il mercato" intorno a lui sta offrendo in quel momento. Cerchiamo ora di fare qualche esempio: un negozio di abbigliamento che in quel periodo sta facendo dei saldi, un ristorante che sta facendo una promozione per il menu del pranzo, un pub che quella sera fa happy hour fino ad una certa ora, una discoteca fa una serata con uno special guest importante, una mostra che ha aperto da poco o qualsiasi altra iniziativa che enti pubblici o privati stanno facendo, la festa del paese che si tiene in quei giorni, un hotel che sta facendo sconti per quella notte e molte altre tipologie di offerte compariranno all'interno dell'applicazione. Ovviamente è proprio l'interessato/i stesso/i a crearsi un account e a tenerlo continuamente aggiornato con promozioni, eventi ed info; per creare tale account non bisogna per forza avere un dispositivo Android dato che è stata messa a disposizione al venditore una pagina web per la registrazione e la successiva gestione del

proprio profilo. Sarebbe riduttivo se nell'applicazione venissero mostrate solo le attività commerciali registrate, quindi vengono mostrate anche attività che non si sono registrate con la differenza che su di esse non vi sono offerte da visualizzare; queste attività ovviamente verranno prelevate, in modo automatizzato, da servizi come pagine bianche o Google map. L'applicazione mostra sulla mappa dove si trova l'utente e dove invece il posto prescelto e a richiesta guida l'utente fino alla meta, appoggiandosi a navigatori esterni installati nel device. Per restringere il numero di eventi visualizzati nell'applicazione l'utente può selezionare oltre alla categoria anche entro quale raggio effettuare la ricerca. In ultima analisi il cliente può scambiare pareri con gli altri clienti registrati su una certa attività commerciale; dopo essere stato in un locale che è piaciuto l'utente può selezionarlo per rimanere continuamente aggiornato sulle promozioni e gli eventi che tale posto propone; va da se che per usufruire di tali funzionalità ci si è dovuto precedentemente registrare al servizio, senza tale registrazione si può comunque usare l'applicazione ma non disponendo però delle funzionalità social avanzate appena descritte.

Andiamo quindi ad illustrare, in modo schematico, quali funzionalità ha effettivamente l'applicazione che voglio realizzare:

- filtro di ricerca per categoria
- filtro di ricerca per range, sarà un'impostazione delle applicazione
- visualizzazione degli eventi disposti in ordine crescente per distanza
- calcolo della distanza esatta tra l'utente e il luogo scelto
- lettura a schermo della descrizione dell'offerta/evento
- visualizzazione sulla mappa della posizione del luogo scelto e dell'utente
- possibilità di farsi guidare fino al luogo scelto, attraverso il navigatore integrato nel device

- possibilità di seguire una certa attività commerciale per rimanere sempre aggiornati sulle sue promozioni/iniziative[1]
- possibilità di lasciare commenti riguardo la soddisfazione su di una certa attività commerciale[1]
- possibilità di registrarsi e gestire il proprio account da web

2.2 Innovazioni sulla concorrenza

A questo punto sono state definite le caratteristiche che assumerà la mia applicazione; riprendendo le tre categorie di applicazioni esistenti che avevo stilato nel paragrafo 2.2.1, "Stato dell'arte", riesco ad illustrare quali innovazioni la mia applicazione porta sul mercato, vediamole. Quello su cui punto non è portare effettivamente grandi funzionalità mai viste prima, anche perché risulterebbe abbastanza difficile considerato che ormai è rimasto poco spazio alla fantasia. Miro piuttosto ad offrire un insieme di servizi che fino ad ora non si sono mai visti tutti insieme nella stessa applicazione, voglio insomma prendere quel che c'è di buono nelle tre categorie da me definite ed inglobarle in un'unica applicazione.
Cerco di spiegarmi meglio, consideriamo la categoria "social-network based" dove si dà la possibilità all'utente di condividere con i suoi amici ogni istante della giornata attraverso check-in, foto, commenti e feedback; la categoria "offers based" che riesce a racchiudere in se un insieme di offerte ultra conveniente per l'utente e che fanno una grande pubblicità al negoziante; e la "geo-chat based" che cerca invece, sulla base di interessi, luoghi visitati etc. , di mettere in comunicazione persone sconosciute non solo virtualmente ma, grazie al GPS, di far continuare la discussione faccia a faccia. È l'insieme di queste "specialità" che voglio cercare di racchiudere in una sola applicazione così da dare un servizio ed un'esperienza il più possibile completa agli utenti. Cercando quindi di concretizzare quanto detto nella mia applicazione in-

[1] per poter usufruire di tali funzionalità bisogna essere registrati presso il servizio

serisco la possibilità di seguire un locale o negozio rimanendo così aggiornato su cosa propone di giorno in giorno, sia come sconti che come eventi particolari. È questa una funzionalità che non ho incontrato in nessuna applicazione visionata. Inoltre non voglio far rimanere i feedback dei clienti sulle attività commerciali come una semplice frase che rimane poco più che fine a se stessa, voglio invece che il negoziante sappia come la sua attività sia reputata dai customer, in modo tale che possa, nei limiti, sempre migliorarsi per dare un servizio continuamente superiore ai proprio clienti.

Capitolo 3

Progettazione

3.1 Introduzione ambiente Android

Prima di procedere a spiegare la struttura del mio progetto è bene illustrare brevemente qualche nozione basilare sulla piattaforma Android. Ho scelto di riportare delle immagini riassuntive, in inglese, del contenuto delle slide del corso di Laboratorio Applicazioni Mobili tenuto dal Professor Luciano Bononi e dagli assistenti Luca Bedogni e Marco Di Felice. Ho scelto queste slide (scritte in inglese) perché sintetiche, ben realizzate e perfette al mio scopo. Esse sono raggiungibili al sito `http://www.cs.unibo.it/projects/android/index.html#resources`

Figura 3.1: Cos'è *Android*

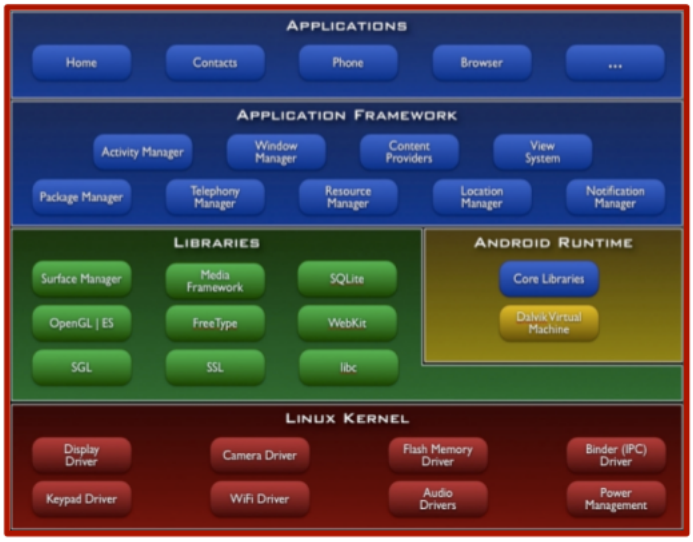


Figura 3.2: Architettura *Android*

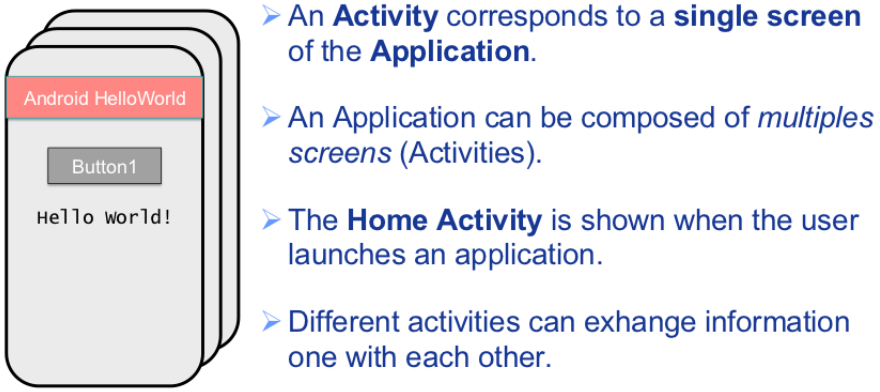
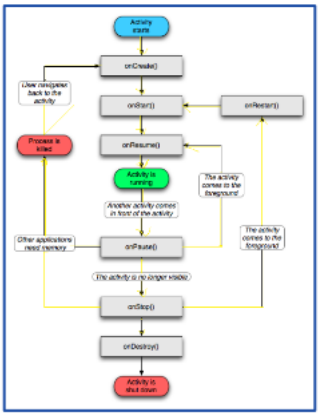


Figura 3.3: *Activity Android*

- Each activity is composed by a list of *graphics components*.
- Some of these components (also called **Views**) can interact with the user by handling **events** (e.g. Buttons).
- Two ways to build the graphic interface

Figura 3.5: Struttura *Activity*



- The **Activity Manager** is responsible for creating, destroying, managing activities.
- Activities can be on different **states**: *starting*, *running*, *stopped*, *destroyed*, *paused*.
- Only one activity can be on the **running** state at a time.
- Activities are organized on a **stack**, and have an event-driven life cycle (details later …)

Figura 3.4: Ciclo di vita *Activity*

- *Android applications typically use both the approaches*!

DECLARATIVE APPROACH

XML Code → Define the Application **layouts** and **resources** used by the Application (e.g. labels).

PROGRAMMATIC APPROACH

Java Code → Manages the **events**, and handles the **interaction** with the user.

Figura 3.6: Programmazione funzionalità e layout *Activity*

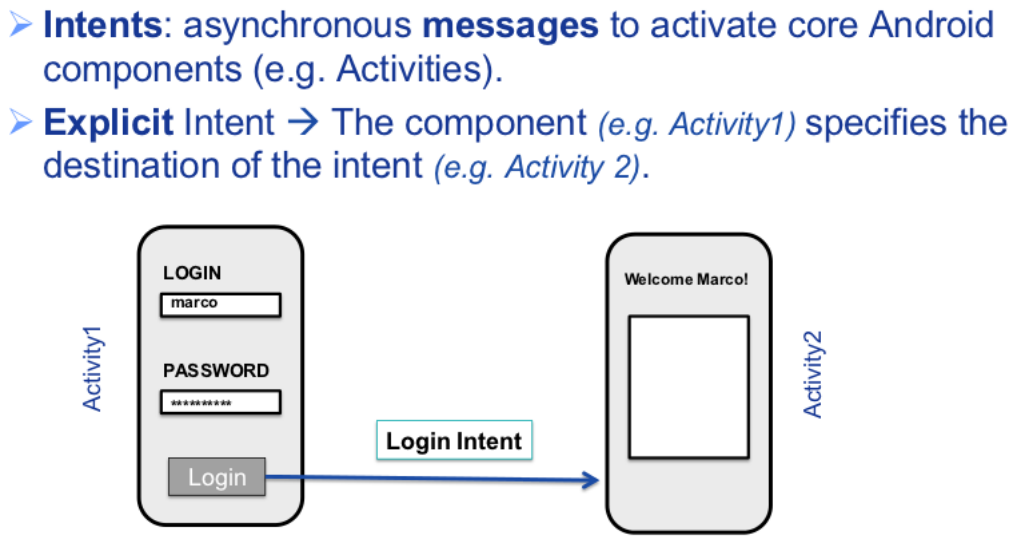


Figura 3.7: *Explicit Intent*

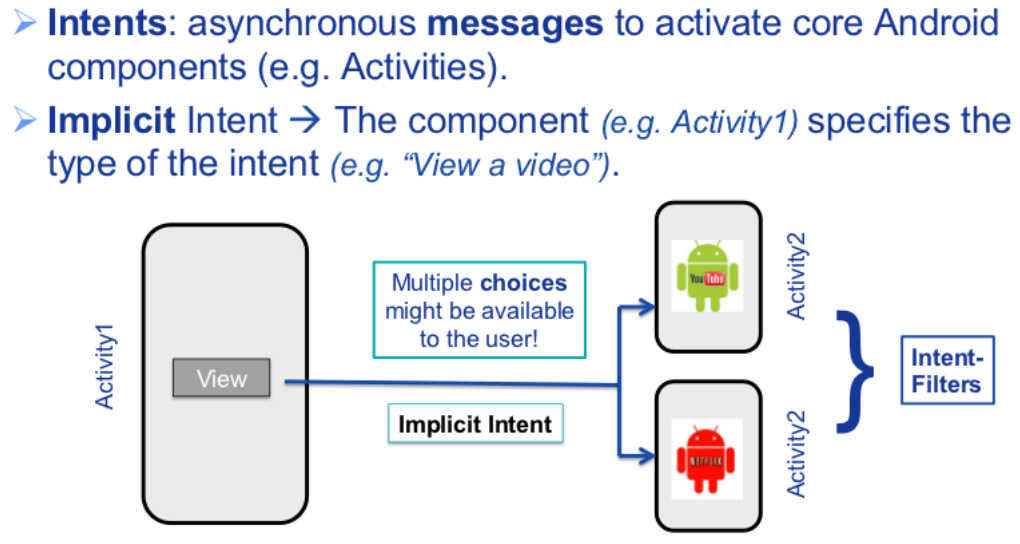


Figura 3.8: *Implicit Intent*

3.2 Struttura del progetto

3.2.1 Server side

Passiamo ora a parlare della struttura dell'applicazione. Avendo bisogno di un database per memorizzare le informazioni relative agli account ho sviluppato l'applicazione basandomi sul modello client-server. Almeno per la fase di sviluppo ho scelto di utilizzare il database gratuito e relativo spazio web offerto da Altervista. Il linguaggio per il database impostomi dal web hosting è mysql. Per la gestione del lato server ho scelto di utilizzare il linguaggio di scripting PHP all'interno del quale ho integrato delle query mysql come in figura 3.9.

```
case "list_categories":

        $query = "SELECT * FROM Categories";
        $result = mysql_query($query);
        $type='Categories';
        break;
```

Figura 3.9: Esempio di query eseguita all'interno del codice PHP

Tale script, *script_ api.php*, viene chiamato attraverso una richiesta http inviata dal dispositivo Android. Una volta attivato fa una connessione al database, controlla che la connessione sia andata a buon fine ed in caso affermativo selezione il giusto database; preleva attraverso una get uno specifico parametro presente nella richiesta http ed in base ad esso seleziona la query che deve effettuare. Ottenuta risposta dal db crea un array associativo e vi mette dentro la risposta appena ricevuta. Lo scopo ultimo dello script è tornare proprio tale risposta ed è tramite la funzione *echo* che compie il suo dovere stampando la risposta correttamente costruita; infine prima di terminare chiude la connessione instaurata col database.

I dati tra il device Android ed il server vengono scambiati attraverso JSON (JavaScript Object Notation) che altro non è che un semplice formato per lo scambio di dati. Avrei potuto scegliere anche di usare XML ma JSON ha dalla

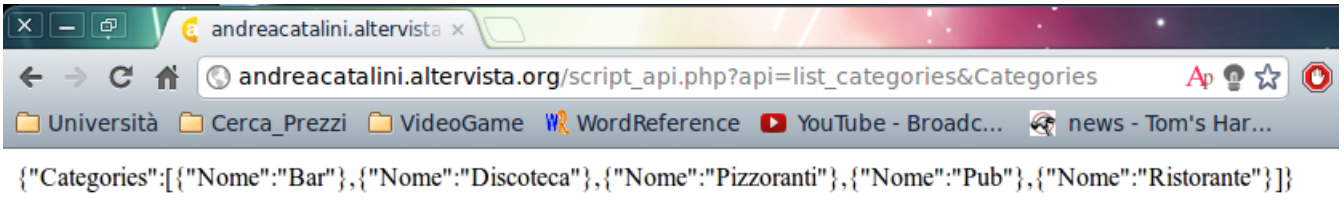


Figura 3.10: *Esempio di output codificato in JSON ottenuto dalla chiamata a* script_api.php

sua una grande leggerezza nella quantità di dati scambiati e un'alta leggibilità da parte dell'uomo. Chi ha il compito di convertire i dati provenienti dal database nel formato JSON è proprio *script_ api.php* tramite il comando php *json_ encode()*. Un esempio di come l'output viene codificato è mostrato in figura 3.10.

L'ultima cosa di cui parlare è la pagina web che serve per la registrazione, da parte del proprietario, di un'attività commerciale. In essa è stato utilizzato, come detto prima, lo spazio web offerto da Altervista. La pagina si compone di un semplice form per l'iscrizione con tutti i campi necessari. L'unica particolarità di cui discutere è il campo categoria. È ovvio che chi si sta iscrivendo non può creare una categoria a caso in cui inserire il suo negozio, ma deve sottostare a quelle che ho creato io all'interno del mio database. Si deve quindi tenere aggiornato in continuazione quel campo. La soluzione che ho adottato è stato di mettere nel tag *<input>* il ritorno della funzione php mostrata nella figura 3.11 .

Tale funzione dopo essersi connessa e aver selezionato il giusto database, fa la query per selezionare le categorie e tale risultato viene poi, attraverso un ciclo, inserito nell'array *$cat;* infine chiude come solito la connessione. A questo punto abbiamo ottenuto un array, che all'interno del codice HTML del form viene scandito tutto tramite un *foreach* PHP. Il risultato finale di questo lavoro è mostrato nella figura 3.12

```
mysql_select_db("my_andreacatalini", $link);


        $query = "SELECT * FROM Categories";
        $result = mysql_query($query);
        $type='Categories';

if (!$result) {
   die("chiamata database fallita");
}

$cat = array();
while($row=mysql_fetch_array($result)) {
array_push($cat,$row[0]);
}
```

Figura 3.11: Parte della funzione PHP che interroga il db per prelevare le categorie

3.2.2 Client side

Componenti funzionali

Per La programmazione in Android c'è ben poco da scegliere, i linguaggi che si devono utilizzare sono Java ed XML. Passiamo ora a parlare dell'effettiva struttura del progetto in Android. Il progetto si compone di tre *package*: il primo si interessa di utility per il controllo del GPS, il secondo gestisce la maggior parte della comunicazione http tra device e server e l'ultimo gestisce le funzionalità e le varie schermate dell'applicazione.

lapeste.tesi.gpscontroll *:* è questo il primo *package* che contiene un'unica classe chiamata *GpsUtility.java* . Qui troviamo un metodo per convertire le coordinate da double a int e un altro per che seleziona il miglior provider, GPS o Wifi, in quel momento da utilizzare per prelevare le coordinate della posizione.

lapeste.tesi.httpSide : questo è il *package* che gestisce la comunicazione

Figura 3.12: Pagina Web del form di registrazione al servizio `http://andreacatalini.altervista.org/`

tra dispositivo e server. Il *package* ha due classi. La prima, *costanti.java*, in cui viene solamente definita la classe *Events* utilizzata nel terzo package per riuscire a salvare tutti i dati strutturati che vengono dal server. La seconda classe, *HttpRequest.java* invece è uno dei moduli più importanti di tutto il progetto. Esso riceve un url con scritto l'indirizzo a cui effettuare la richiesta, si connette a tale server come mostrato in figura 3.13 e, in caso la connessione sia andata a buon fine, parsa l'output codificato in JSON (esempio mostrato in figura 3.14) e restituisce un array contenente tutti gli oggetti richiesti.

```
HttpClient client = new DefaultHttpClient();
HttpGet request = new HttpGet(url);

HttpResponse response = client.execute(request);
```

Figura 3.13: Connessione iniziale e successiva richiesta al server

```
for(int i=0 ; i<jArray.length() ; i++ )
{
    try {
        //estrapolo l'iesimo JSONObject dal JSONArray
        //e su questo eseguo il case "tyPe"
        switch (tyPe)
        {
            case costanti.CATEGORIES:

                    result[i] = jArray.getJSONObject(i).getString("Nome");
                    break;
```

Figura 3.14: Esempio di parsing di una risposta codificata in JSON

lapeste.tesi.mainprogram : questo è il *package* più grande del progetto, contiene infatti 7 classi.

- *CategoryListView.java* : questa classe genera la prima *Activity* ad essere mostrata quando l'applicazione viene lanciata, in essa verranno visualizzate le possibili categorie tra cui scegliere. Se è la prima volta che viene avviata l'applicazione non si avrà ancora settato nessun range per la ricerca, quindi ho impostato che si apra subito un *EditText*, ovvero un pop-up editabile, dove si deve inserire un numero che rappresenterà il range massimo, in chilometri, per le successive ricerche. Il modo in cui si tiene traccia se è la prima volta che viene avviata l'applicazione non è un contatore, come ci si potrebbe aspettare, ma sfrutto le proprietà delle preferenze come mostrato nella figura 3.15
  Vediamo che alla creazione dell'*Activity* vengono create le *SharedPref-*

```
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);

  SharedPreferences prefs = getSharedPreferences(FIRST_START, Context.MODE_PRIVATE);
  SharedPreferences.Editor editor = prefs.edit();
  SharedPreferences prefRange = getSharedPreferences(RANGE, Context.MODE_PRIVATE);

  if (prefs.getBoolean(FIRST_START, true))
    {
        createEditext();
        editor.putBoolean(FIRST_START, false);
        editor.commit();
    }
```

Figura 3.15: Pop-up che viene visualizzato al primo avvio

erences prefs e *prefRange*, ovvero delle preferenze persistenti, dove della prima ne viene letto il contenuto tramite la *getBoolean()* e se ancora vuota viene tornato true e quindi si entra nel ramo if dove *prefs* viene settata a false, in modo tale che la volta successiva si la *getBoolean()* torni false e quindi il flusso d'esecuzione non entrerà più all'interno di quel ramo. La *prefRange* viene usata più avanti per tenere salvato il range impostato. È comunque possibile cambiare il range di ricerca in un secondo momento premendo l'apposito tasto fisico del device An-

droid, che mostrerà a schermo un menu con due bottoni, il primo dei quali se premuto apre ancora quell'*EditText* per impostare la quantità di chilometri per effettuare la ricerca. L'altro tasto invece serve per aprire la schermata in cui vengono mostrate le attività che stiamo seguendo. Vediamo in figura 3.16 come tale menu viene creato.

Vediamo semplicemente che, al momento in cui il tasto del menu è

```
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
    // Handle item selection
    switch (item.getItemId())
    {
        case R.id.range:
            createEditext();
            return true;
        case R.id.followed:
            Intent myIntent = new Intent(this, FollowedView.class);
        //    myIntent.putExtra("id", id);
              startActivity(myIntent);
              return true;
        default:
            return super.onOptionsItemSelected(item);
    }
```

Figura 3.16: Menu della *CategoryListView*

premuto, se si clicca range viene avviato un *EditText* attraverso la *createEditext()* mentre se premiamo l'altro bottone verrà lanciato l'*intent* che apre l'*Activity* che mostra gli eventi/offerte delle attività commerciali seguite.

Illustrati i menu e tutte le varie preferenze passiamo a parlare del flusso di esecuzione che segue questa classe. Inizialmente si controlla che, sia la connessione ad internet, che quella GPS siano attive, a questo punto ricorrendo alla *HttpRequest.java* si prelevano le categorie di attività commerciali/eventi che sono presenti nel database e le si visualizzano in una lista; contemporaneamente si prelevano le coordinate GPS da fornire alla prossima activity che le userà per effettuare la query. Quan-

do l'utente, dopo averne scelta una, clicca su di una categoria viene generato l'intent mostrato in figura 3.17

vediamo che, nel momento in cui viene premuto lo schermo è chiam-

```
@Override
public void onItemClick(AdapterView<?> parent, View view, int position, long id)
{
    Intent myIntent = new Intent(view.getContext(), ListDeals.class);
    myIntent.putExtra("type", ((TextView) view).getText());
    myIntent.putExtra("coordinateLat", latitude);
    myIntent.putExtra("coordinateLon", longitude);
    startActivity(myIntent);
}
```

Figura 3.17: Intent generato nel momento in cui si clicca su di una categoria visualizzata nella lista

ato il metodo *onItemClick()* all'interno del quale c'è *l'intent* di cui stavamo parlando ovvero *myIntent*. In esso vengono inseriti dati aggiuntivi, tramite il metodo *putExtra,* da inviare alla prossima Activity. I dati che vengono inseriti sono il tipo, la latitudine e la longidutine del device. Infine viene lanciato effettivamente l'intent con *startActivity(myIntent)* ed a questo punto si conclude il lavoro di questa classe, passando il testimone a *ListDeals.java.*

- *FollowedView.java* : c'è poco da spiegare in questa classe se non che è del tutto simile alla *CategoryListView.java* nell'aspetto e nelle funzionalità. L'unica differenza sta nei dati che mostra, ovvero quelli delle attività commerciali che stiamo seguendo. Essi sono ancora prelevati attraverso solita richiesta al server composta, stavolta, con i gli ID dei negozi che seguiamo.

- *ListDeals.java* : questa classe genera la seconda *Activity* dell'applicazione in cui verranno visualizzate le offerte/eventi per la categoria selezionata nella precedente *Activity.* La prima cosa che la classe fa è leggere quali dati *l'Activity* precedente ha passato nell'*intent,* come mostrato in figura 3.18

Vediamo quindi che attraverso il metodo *getIntent()* vengono prelevati

```
String type = this.getIntent().getStringExtra("type");
int Latitude = this.getIntent().getIntExtra("coordinateLat", -1);
int Longitude = this.getIntent().getIntExtra("coordinateLon", -1);
```

Figura 3.18: Lettura dati aggiuntivi scambiati tra le Activity attraverso un intent

i dati e successivamente salvati in delle variabili. Questi dati prelevati verranno inseriti nell'url da inviare alla *HttpRequest.java* come mostrato in figura 3.19

si vede quindi che, dopo aver istanziato un array di Object[], costruis-

```
Object[] arrayObject = new Object[]{};
String url = "http://andreacatalini.altervista.org/script_api.php?api=" +
      "list_events&categoria="+type+"&latitudine="
        +Latitude+"&longitudine="+Longitude+"&range="+range;
arrayObject = new HttpRequest().JSONparse(url, "Events", costanti.EVENTS);
```

Figura 3.19: Richiesta effettuata al database tramite il metodo *JSONparse()*

co l'url appropriato inserendo i valori ottenuti dall'intent ovvero *type, Latitude e Longitude;* l'ultimo parametro range viene letto invece dalle impostazioni dell'applicazione. A questo punto viene effettivamente fatto richiesta al server col metodo *JSONparse()* e tale risposta viene poi salvata in quell'array di Object[] istanziato all'inizio. *arrayObject* viene parsato per estrapolarne i dati che devono essere visualizzati nella lista delle offerte visualizzate da questa *Activity.* Una volte ottenuti questi dati, tra cui anche la distanza tra l'utente ed il luogo prescelto, riempie la lista con tutte le offerte/eventi presenti per la categoria selezionata ordinati per distanza crescente; fatto questo conclude il suo compito lanciando un intent, con aggiunto come extra il valore id, passando così la palla a *EventDescription.java* .

- *EventDescription.java* : questa classe genera la terza *Activity* in cui andrà mostrata la descrizione dell'evento/offerta selezionata. In questa classe un menu diverso da quello visto in precedenza e bottoni per diverse funzionalità. Inizialmente si leggono i soliti dati aggiuntivi proveniente dall'intent; con essi si costruisce l'url per effettuare la richiesta al server, si esegue tale richiesta e si visualizza nell'*Activity* i dati ottenuti; si conclude col solito lancio dell'intent per chiamare la prossima *Activity*. Fin qua tutto rimane circa simile al comportamento delle precedenti *Activity* ma nel mezzo delle varie operazioni c'è qualcosa di più, infatti l'intent stavolta non è lanciato selezionando un elemento della lista, visto che qui neanche esiste una lista, ma ben sì viene creato un apposito bottone il quale, una volta premuto, lancerà quest'intent, come mostrato in figura 3.20

  come vediamo viene creato un *Button goMap* che prende lo stile definito

```
final Button goMap = (Button) findViewById(R.id.goMap);
goMap.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        Intent myIntent = new Intent(v.getContext(), MyMapView.class);
        myIntent.putExtra("id", id);
        startActivity(myIntent);
    }
});
```

Figura 3.20: Creazione del bottone che lancia l'intent per la seguente *Activity*

nella risorsa di id *goMap*; viene poi usato il metodo *setOnClickListener()* per legare un *listener* che rimane in ascolto per qualche evento sull'oggetto, nel nostro caso di tipo *Button;* viene quindi sovrascritto il metodo *onClick()* ridefinendo la sua azione ovvero creare l'intent, inserire il dato aggiuntivo ed infine lanciare tale intent.

Altra cosa che viene eseguita per la prima volta è mostrare a schermo

la distanza esatta tra l'utente e l'evento. Dico che è la prima volta che viene mostrata tale distanza perché quella sulla quale era ordinata la lista degli/lle eventi/offerte della precedente *Activity* era una distanza approssimata calcolata in linea d'aria; stavolta invece sono ricorso ad un servizio di Google, che spiegherò nella prossima sezione, che calcola la distanza esatta tenendo quindi in considerazione come è sviluppata la rete stradale; l'uso di tale servizio è mostrato nella figura 3.21

dove possiamo vedere che viene fatta la richiesta usando sempre il

```
//*****************************Google Matrix API*****************
  arrayGoogleObject = new HttpRequest().JSONparse("http://maps.googleapis.com/maps/api/...
  String[] results = new String[]{};
  results = Arrays.copyOf(arrayGoogleObject, arrayGoogleObject.length, String[].class);

  //*****************************Google Matrix API*******************
```

Figura 3.21: Uso del Google matrix api all'interno della classe *EventDescription.java*

metodo *JSONparte()* ma stavolta quello che passiamo è un url che rispetta le api del servizio di Google. La risposta viene inserita nell'*arrayGoogleObject* e convertita tramite il metodo di *Arrays.copyOf()* (che prende come parametri l'array che deve convertire, la sua lunghezza e la classe in cui deve essere convertito) in un Array di stringhe.

C'è ancora un altro bottone che serve per lanciare l'intent che verrà cattura dall'applicazione che il sistema operativo ha impostata di base per rispondere a quell'intent. Nella figura 3.22 si vede com'è fatto tale intent.

L'ultima cosa di cui parlare è il menu a cui si accede premendo l'apposito tasto del device. In esso vengono mostrati due pulsanti, uno per aggiungere ai preferiti il gestore dell'evento che si sta visualizzando e quell'altro per passare alla schermata in cui vengono mostrati tutti gli eventi dei commercianti che seguiamo, ovvero quelli aggiunti ai prefer-

```
// formato di lat e long , esempio :
//DESTINATION_LAT = "42.54335"
//DESTINATION_LONG = "9.324423"

String navigateToLat = DESTINATION_LAT;
String navigateToLong = DESTINATION_LONG;

Intent i = new Intent(Intent.ACTION_VIEW, Uri
                .parse("google.navigation:q="
                                + navigateToLat + ","
                                + navigateToLong));
startActivity(i);
```

Figura 3.22: Intet per chiamare il navigatore

iti. L'implementazione è praticamente uguale a quella mostrata per il menu della *CategoryListView.java* (figura 3.8).

- *OverlayOnMap.java* : prima di passare alla prossima *Activity* bisogna un attimo parlare di questa classe. Essa estende *ItemizedOverlay* ovvero, come riportato nel manuale di Google, una classe base per un *Overlay* che consiste in un elenco di *OverlayItems*. Questo gestisce l'ordinamento nord-sud per il disegno, creando limiti di copertura, disegnando un marker per ogni punto, e mantenendo l'elemento focalizzato. Esso abbina inoltre i tocchi sullo schermo ad elementi ed invia eventi di cambiamento di messa a fuoco ad un listener opzionale.
  Spiegato che cos'è la classe che estende vediamo i punti salienti di come ho costruito la mia classe estesa. Ho aggiunto un metodo, *addOverlay()*, per caricare nella lista di *Overlay* l'oggetto passato in input come mostrato nella figura 3.23
  dove vediamo, oltre al costruttore che utilizza il metodo *boundCenterBottom()* della classe che sta estendendo, viene creato *mOverlays* un ArrayList<OverlayItem>, cioè una array di OverlayItem, che è utilizzato nel metodo *addOverlay()* per tenere tutti gli elementi aggiunti all'istanza di questa classe; viene infine chiamata la *populate()* chiamerà il metodo *createItem(int)* che ho sovrascritto più avanti nel codice. Cer-

```
public class OverlayOnMap extends ItemizedOverlay {

    private ArrayList<OverlayItem> mOverlays = new ArrayList<OverlayItem>();
    private Context mContext;

    public OverlayOnMap(Drawable defaultMarker, Context context) {
        super(boundCenterBottom(defaultMarker));
        mContext = context;
    }

    public void addOverlay(OverlayItem overlay) {
        mOverlays.add(overlay);
        populate();
    }
```

Figura 3.23: Metodo della classe *OverlayOnMap.java* per aggiungere in elemento alla lista

cando di chiarire le idee possiamo figurarci questa classe come una lista di *OverlayItem*
L'ultimo metodo di cui è importante parlare è la *onTap()* che è un metodo ereditato dalla superclasse *ItemizedOverlay* e che viene sovrascritto per dare un comportamento specifico nel momento in cui viene premuto sullo schermo l'oggetto di tipo *OverlayOnMap*. Vediamo in figura 3.24 come tale comportamento è definito.

Vediamo che viene creato un *item* di tipo *OverlayItem* all'interno del quale viene inserito l'indice della oggetto che stiamo definendo nella classe. Viene poi creato *dialog*, un'istanza della classe AlertDialog cioè un oggetto che rappresenta una piccola finestra di dialogo che si posiziona al livello più alto nella visualizzazione delle schermate dell'applicazione. A *dialog* viene poi settato un titolo e un messaggio da mostrare nel piccolo pop-up che verrà effettivamente creato dal metodo *show()*. Il metodo infine ritorna *true* per indicare all'applicazione che il "tap" sullo schermo per quell'oggetto deve essere gestito.

- *MyMapView.java* : possiamo ora passare a parlare dell'ultima *Activity*.

```
@Override
protected boolean onTap(int index) {
  OverlayItem item = mOverlays.get(index);
  AlertDialog.Builder dialog = new AlertDialog.Builder(mContext);
  dialog.setTitle(item.getTitle());
  dialog.setMessage(item.getSnippet());
  dialog.show();
  return true;
}
```

Figura 3.24: Metodo *onTap()* sovrascritto all'interno della classe *OverlayOnMap.java*

Questa classe è quella che va a generare l'ultima *Activity* dell'applicazione ed è anche una delle più complesse. Questa complessità risiede principalmente nel fatto che vengono usate molte classi che si interfacciano coi servizi offerti da "Google Maps", quindi bisogna studiare un attimo tutto quello di cui si ha bisogno; spiegherò brevemente di volta in volta le nuove classi che introduco nella spiegazione.
Si nota sin da subito le differenza con le altre classi del genere, essa infatti non estende più le normali *Activity* ma estende invece una *MapActivity*. Detto questo passiamo ora a spiegare l'operato della classe. All'inizio la classe effettua la solita lettura dei dati inviatigli dalla precedente *Activity* per poter fare, successivamente, la solita richiesta al server dal quale in questo caso riceve le coordinate dell'evento che vogliamo visualizzare sulla mappa. Parsa ancora l'output trasformandolo in un array *JSONObject* e ne estrae tutti i dati di cui ha bisogno mettendoli in una array di *Events[],* classe definita nel package *lapeste.tesi.httpSide.* Fin niente di troppo diverso dal solito, ma d'ora in avanti interverranno nuove classi.
Prende il riferimento alla *MapView* e ne setta alcune impostazioni per la mappa da visualizzare, come ad esempio li livello di zoom, se mostrare o no i controlli per lo zoom e cose del genere. A questo punto crea *itemizedoverlay,* del tipo *OverlayOnMap* prima spiegato, passando

nel costruttore il marker che rappresenterà gli *OverlayItem* raffigurati sulla mappa come mostrato in figura 3.25

poi carica le informazioni provenienti dalla risposta del server dentro

```
List<Overlay> mapOverlays = mapView.getOverlays();
Drawable drawable = this.getResources().
                        getDrawable(R.drawable.red_marker);
OverlayOnMap itemizedoverlay = new OverlayOnMap(drawable, this);
```

Figura 3.25: Creazione dell'istanza della classe *OverlayOnMap*

l'*itemizedoverlay* e quindi carica tale oggetto nel *mapOverlays* tramite il metodo *add()* e aggiunge la posizione geografica del device usando la classe *myLocationOverlay* e vi chiama poi il metodo *enableMyLocation()* che serve per cercare di abilitare *MyLocation* registrandolo per aggiornamenti dal GPS; il tutto è mostrato nella figura 3.26.

Il *MyLocationOverlay* è un *Overlay* per disegnare sulla mappa la po-

```
mapOverlays.add(itemizedoverlay);

//aggiungo la mia posizione
myLocationOverlay = new MyLocationOverlay(this, mapView);
mapOverlays.add(myLocationOverlay);
myLocationOverlay.enableMyLocation();
```

Figura 3.26: Aggiunta di *itemizedoverlay* e della posizione del device alla *MapView*

sizione corrente dell'utente e l'accuratezza di tale rilevazione, e / o un riquadro della bussola-rosa.

Il prossimo oggetto che verrà introdotto è di tipo *LocationManager*. Questa classe fornisce l'accesso ai servizi di localizzazione del sistema. Questi servizi permettono all'applicazione di ottenere periodici aggiornamenti sulla posizione geografica del device, o lanciare un specifico

Intent quando il device entra in prossimità di una data posizione geografica. Ho introdotto questo oggetto perché la classe a questo punto ha bisogno di ottenere un riferimento a tale *LocationManager* per controllare lo stato del GPS il quale se è spento viene lanciato un intent che viene catturato dal gestore delle impostazione del sistema operativo, il quale apre la pagina delle impostazioni del GPS. Tutto ciò si capisce meglio guardando la figura 3.27 che riporta quanto spiegato.

L'ultimo oggetto di cui bisogna parlare è il *locationListener.* Esso è

```
// Ottengo il riferimento al LocationManager
   locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);

// Verifichiamo se il GPS sia abilitato altrimenti apriamo
// le impostazioni del sistema per permettere all'utente di attivarlo
   if(!locationManager.isProviderEnabled("gps")){

       Intent intent = new Intent(
               android.provider.Settings.ACTION_LOCATION_SOURCE_SETTINGS);
       startActivity(intent);

   }
```

Figura 3.27: Ottenimento riferimento al *LocationManager* e successivo controllo dello stato del GPS

usato per ricevere notifiche dal *LocationManager* quando la posizione è cambiata. I metodi della classe sono chiamati se il *LocationListener* è stato registrato con il servizio del location manager usando il metodo *requestLocationUpdates().* Questo oggetto serve alla classe per definire determinati comportamenti che l'applicazione deve avere in risposta a certi eventi. Non saranno mostrati tutti ma solo i più importanti ovvero *onLocationChanged()* che serve a definire il comportamento dell'applicazione quando la posizione del device cambia di una distanza stabilita; vediamo come è definito nella figura 3.28 .

Vediamo quindi che nel metodo sovrascritto per prima cosa viene invalidata l'intera vista attraverso il metodo *invalidate()* della *mapView;*

```
locationListener = new LocationListener()
{
    @Override
    public void onLocationChanged(Location location)
    {
            // Aggiorna il marker della mappa
            mapView.invalidate();
            mapView.getController().animateTo(point);
    }
```

Figura 3.28: Definizione di cosa fare in risposta al cambiamento della posizione del device

in questo modo si ottiene una rilettura con successivo refresh dei punti sulla mappa. Poi viene chiamato il metodo *...animateTo()* che serve per centrare la mappa sullo schermo nella posizione del punto dato in input. Si conclude così le operazioni che compie tale classe.

- *costanti.java* : l'ultima classe contenuta in questo package è semplicemente un contenitore di costanti varie utilizzate all'interno di tutto il progetto. Utili per evitare "magic-number" all'interno del codice che renderebbero ardua l'interpretazione.

Componenti presentazionali

Per quanto riguarda la parte grafica dell'applicazione ho scelto di costruire l'interfaccia seguendo l'approccio dichiarativo ovvero quello basato sul codice XML; con esso possiamo definire il layout dell'applicazione e le risorse che andrà ad utilizzare. Sull'aspetto delle componenti grafiche non sarò specifico come la parte sulle componenti funzionali in quanto, a mio avviso, meno importante ai fini dello studio, ma è comunque giusto darne almeno un accenno. Principalmente è stato definito un file XML per ogni *Activity* più qualche altro file in cui ci sono state definite le risorse che andrà ad usare l'applicazione. Per la prima *Activity* vediamo quali sono i parametri impostati (figura 3.29):

Vediamo che è un file XML abbastanza semplice in cui , dopo aver settato

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:textSize="25sp" >
</TextView>
```

Figura 3.29: File XML per la definizione dell'aspetto grafico della prima *Activity* cioè quella per la selezione delle categorie

il *namespace*, viene impostato che l'altezza e la larghezza della *TextView,* ovvero una vista che mostra testo all'utente, riempiano completamente il padre che la contiene e poi con la voce *padding* si determina, in pixel, lo spessore del bordo che racchiude tale vista; con *textSize* viene invece impostata la dimensione che dovrà avere il testo all'interno del contenitore.
Gli altri file XML per le viste non sono molto diversi se non per qualche impostazione in più. Ad esempio possiamo mettere più *TextView* o altri tipi di viste nello stesso layout come fatto per il *list_ deals.xml.* L'ultimo file di cui vale la pena discutere è quello relativo al lato presentazionale della *MapView* mostrato nella figura 3.30
tra i vari tag XML troviamo quello chiamata *<com.google.android.maps.MapView />*

```
<com.google.android.maps.MapView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/mapview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:clickable="true"
    android:apiKey="0ygbeX                    iHh_cJg"/>
```

Figura 3.30: Layout per la *MapView*

che è stato riportato nell'immagine 4.17. Notiamo che, dopo aver settato

il *namespace,* si rende , tramite il l'attributo *id,* referenziabile tale risorsa con un nome, nel nostro caso "mapview". Poi troviamo ancora gli attributi *layout_ width* e *layout_ height* descritti prima, il *clickable* impostato a true permette all'utente di interagire con la mappa. L'ultimo attributo, senza il quale non funzionerebbe niente, è *apiKey* che detiene la chiave delle API di Maps per l'applicazione, la quale chiave prova che l'applicazione e il certificato del firmatario sono stati registrati con il servizio di Maps. Ciò è necessario al fine di ricevere i dati della mappa, anche mentre si sta sviluppando (per maggiori info guardare sezione 4.3 Servizi Web utilizzati)

3.3 Servizi Web incorporati

Per poter visualizzare e poi interagire con la mappa all'interno della mia applicazione sono ricorso a *Google Maps Android API.* Esse danno la possibilità di aggiungere funzioni di cartografia alla mia applicazione; per includerle nell'applicazione è sufficiente importare la libreria esterna delle Google APIs chiamata *com.google.android.maps.* Le classi della libreria Maps offrono un built-in download, rendering e memorizzazione nella cache delle piastrelle di Maps, così come una varietà di opzioni di visualizzazione e controlli.
La classe chiave nella libreria Maps è la *MapView*, una sottoclasse delle *ViewGroup* nelle librerie standard di Android. Una *MapView* mostra una mappa i dati ottenuti dal servizio di Google Maps. Quando la mappa ha il focus, essa può catturare i tasti premuti e le gesture zoom e de-zoom, includendo la gestione delle richieste di rete per piastrelle aggiuntive delle mappe. Esso fornisce anche tutti gli elementi della UI (user interface) necessari agli utenti per il controllo della mappa. L'applicazione può anche utilizzare i metodi della classe *MapView* per controllare la programmazione della *MapView* e disegnare diversi tipi di *Overlay* nella parte superiore della mappa. Riassumendo la classe MapView fornisce un wrapper per l'API di Google Maps che consente all'applicazione di manipolare i dati di Google Maps attraverso i metodi della classe, e consente di lavorare con i dati di Maps come si farebbe

con altri tipi di *View*.

L'altro servizio che ho utilizzato si chiama *The Google Distance Matrix APIThe Google Distance Matrix API*. Esso è un servizio che fornisce la distanza di viaggio ed il tempo per una matrice di origini e destinazioni. Le informazioni ritornate sono basate sul percorso raccomandato tra i punti di inzio e fine, come calcolati da Google Maps API, e consistono di righe contenenti i valori *duration distance* per ogni coppia.
Questo servizio non ritorna però le informazioni dettagliate sul percorso, ma per il mio scopo ovvero mostrare la distanza esatta tra l'utente e il luogo desiderato vanno più che bene.

3.4 Software e IDE

Per sviluppare questo progetto ho avuto bisogno di utilizzare contemporaneamente diversi tool. Come IDE, Integrated development environment, ho utilizzato *Eclipse* (figura 3.31) in quanto Android offre uno speciale plugin per questo IDE, chiamato *Android Development Tools (ADT)*. Questo plugin è disegnato per dare un ambiente potente ed integrato nel quale sviluppare applicazioni Android. Esso estende le capacità di *Eclipse* per impostare rapidamente nuovi progetti, costruire l'interfaccia grafica, debuggare l'applicazione, esportare, firmata o meno, il pacchetto eseguibile dell'applicazione per la successiva distribuzione.

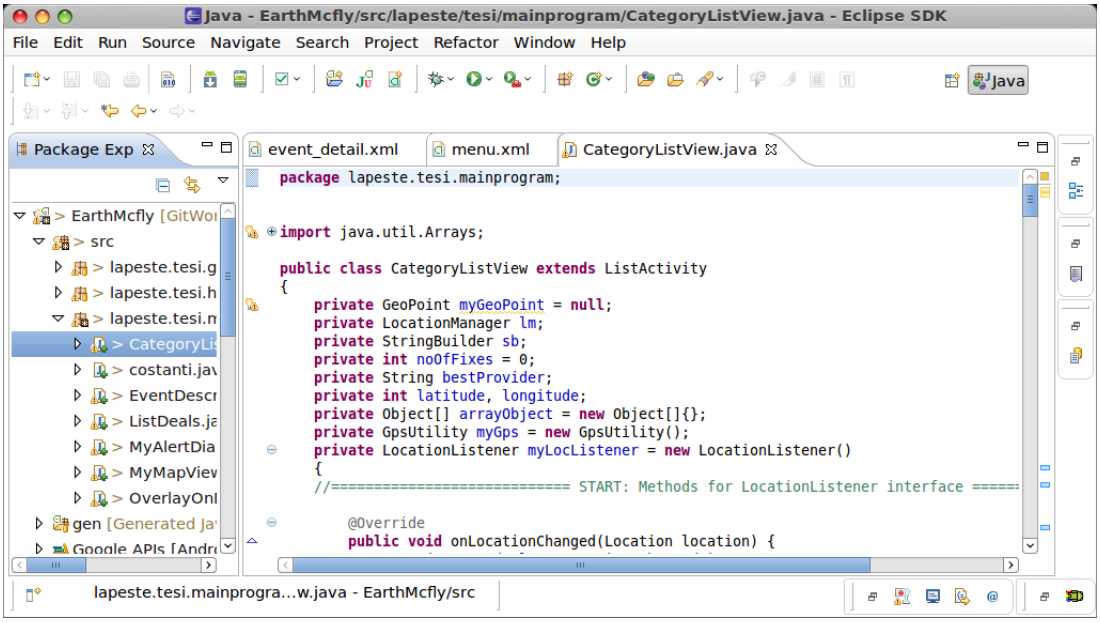


Figura 3.31: Eclipse screenshot

Per la programmazione in PHP ho invece usato il normalissimo *Gedit* ovvero l'editor incluso nelle distribuzioni Gnome. Il file che stavo però generando, ovvero *script_api.php,* avevo bisogno di caricarlo nel server di Altervista abbastanza di frequente per la fase di test e debug. Per evitare di fare ogni volta a mano l'upload dall'interfaccia web fornita da Altervista ho usato il client FTP chiamato *Filezilla* (figura 3.32).

L'ultimo tool, ma sicuramente non meno importante, è quello che mi ha permesso di testare l'applicazione ovvero l'emulatore Android fornito insieme all'SDK (figura 3.33). L'applicazione è stata comunque testata anche nel mio personale terminale Android.

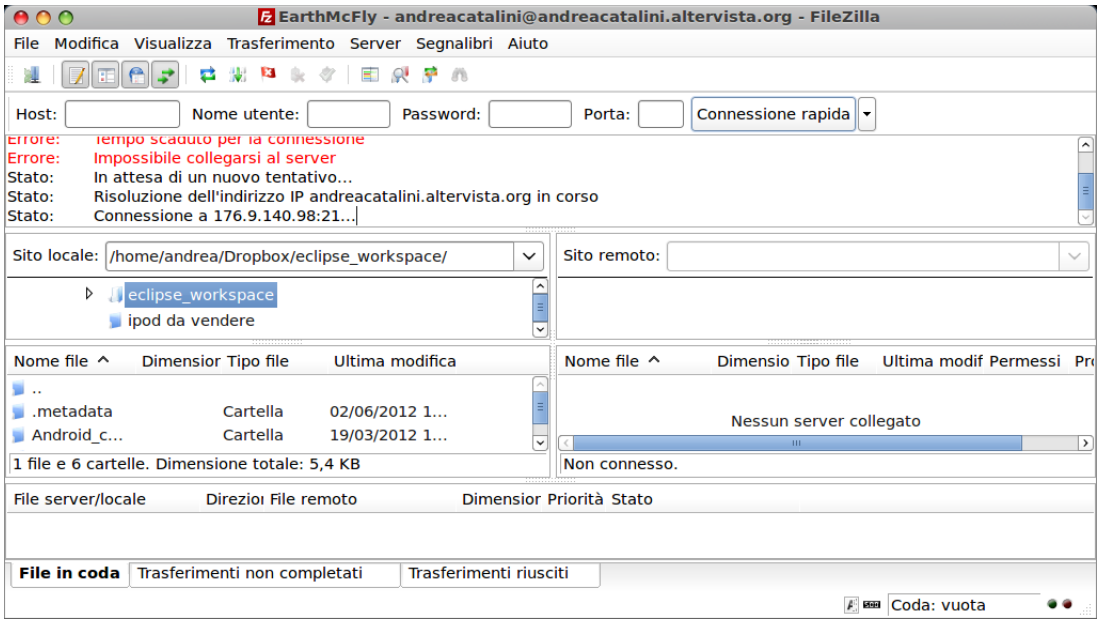


Figura 3.32: *Filezilla* screenshot

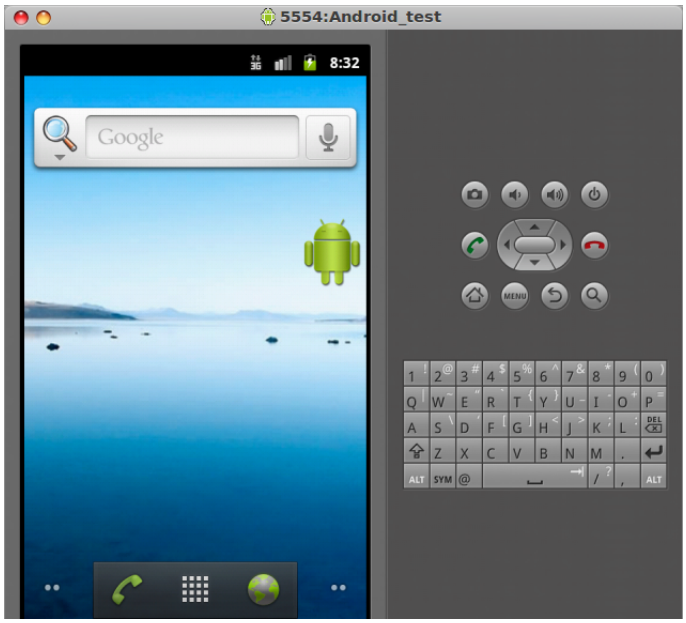


Figura 3.33: Emulatore Android fornito con l'SDK

Capitolo 4

Valutazioni

Dopo aver terminato la stesura del codice ed averne testato il corretto funzionamento bisogna passare a valutarne il risultato. Per poterlo fare si ha bisogno di definire a priori una metrica di valutazione della qualità, ovvero selezionare quali aspetti reputo più importanti di altri in applicazioni di questo genere.

4.1 Metrica

Un fattore che ritengo molto importante è l'esperienza utente, quindi tutta quella parte che riguarda l'usabilità dell'applicazione. Essa deve allora mostrare all'utente un'interfaccia chiara, semplice, intuitiva; non lo deve far perdere in un'infinità di menu, impostazioni e qualsiasi altra cosa renda troppo dispersivo e macchinoso il raggiungimento del risultato.
Non è da meno l'estetica. È vero che un'applicazione dall'interfaccia intuitiva avrà sicuramente più successo di una con l'interfaccia complessa e costruita con poca logica; ma se l'aspetto grafico non è curato anche un'applicazione con l'interfaccia ben struttura soffrirà. Come si dice "anche l'occhio vuole la sua parte" intendendo che se l'UI risulta realizzata con stile e buon gusto essa attirerà più utenti perché darà piacere nella sua visualizzazione.
Un altro importante punto da non sottovalutare assolutamente riguarda la

chiarezza con la quale si definiscono gli obbiettivi che ci si prefigge di voler realizzare. È risaputo che progetti mal strutturati sin dall'inizio hanno vita breve. Tutti possono commettere errori però più il progetto risulta piccolo più facilmente si riesce a tenerlo pulito e ben strutturato, soprattutto quando il team di sviluppo è composta da poche persone, nel nostra caso una sola. Viene da sé che meno funzionalità vengono aggiunte alla rinfusa e più quelle poche che ci sono risulteranno robuste, poco affette da errori e anche ben integrate tra loro.
L'aver esposto in modo chiaro e semplice gli obbiettivi aiuta anche nello step successivo alla messa in funzione di tutto il sistema, ovvero come porre l'applicazione al pubblico. Questo è un punto molto importante, in quanto determinerà il reale successo o meno dell'applicazione. Se i punti sono ben definiti e non troppo dispersivi allora sarà più semplice porre sul mercato il prodotto perché si riesce a trasmettere meglio al pubblico cosa effettivamente l'applicazione fa, quale settore intende coprire e soprattutto nella mente dell'utente finale si inculca l'idea che quel prodotto risponde ad una ben definita necessità e non è invece un minestrone di funzionalità che non hanno neanche senso di esistere accostate tra loro.

Quelli trattati fin'ora sono solamente aspetti estetici e funzionali, passiamo ora a trattare invece gli aspetti riguardanti l'utilizzo delle risorse da parte dell'applicazione.
Sicuramente un'applicazione mobile che per la maggior parte del tempo mostra dati scaricati da un database ha bisogno di non eccedere troppo con la quantità di byte scambiati per due ragioni, la prima è che comunque le connessioni sono ancora lente quindi pensare di scaricare troppi dati non farebbe altro che rallentare inutilmente l'applicazione e il secondo motivo è che i piani dati che i gestori di telefonia mobile mettono a disposizione sono comunque limitati nella quantità di byte, quindi consumarne troppi ogni qualvolta l'applicazione viene avvita non invoglia sicuramente l'utente ad utilizzarla.
Deve essere tenuto in considerazione anche l'uso della cpu visto che è principalemente essa a determinare la durata della batteria. Non bisogna però

sottovalutare il GPS, è anch'esso un dispositivo che ha bisogno di un certo quantitativo di energia e se messo troppo sotto stress con continui controlli causati da impostazioni sconsiderate, come ad esempio range di spostamento troppo piccoli per l'aggiornamento della posizione, di sicuro la batteria non ne gioverà. Penso che sono tutti d'accordo sul fatto che il consumo della batteria è un punto cruciale nel momento di scegliere a quale applicazione delegare quel determinato compito; è per questo che bisogna prestare la massima attenzione al carico che la cpu ed il GPS hanno in ogni momento dell'esecuzione del programma.
Dobbiamo anche tenere in considerazione che stiamo trattando con dispositivi mobile più o meno performanti, quindi non solo la cpu non deve essere troppo stressata ma anche la quantità di memoria richiesta deve essere contenuta. A maggior ragione perché ci troviamo in ambiente Android dove, come spiegato nel paragrafo 1.1.2, c'è una grande frammentazione nelle caratteristiche hardware dei device in commercio.
Alla fine non basta però che l'applicazione consumi poco, perché tutti sono in grado di far consumare poca batteria se poi l'applicazione risulta lentissima, sono pochi invece quelli che riescono a far coesistere velocità elevata e consumi contenuti. È quindi questo un altro fattore che deve essere tenuto in considerazione, la velocità di esecuzione.

Cerchiamo di ricapitolare, con un lista, i punti che applicazioni di questo genere devono cercare di soddisfare il più possibile:

- Buona usabilità dell'applicazione
- grafica ben curata dell'UI (user interface)
- obbiettivi contenuti, semplici e ben definiti
- campo di azione dell'applicazione ben definito
- scambio di dati il più possibile contenuto
- uso moderato della cpu

- buon compromesso tra precisione e consumi del GPS
- memoria RAM richiesta contenuta
- velocità e fluidità di esecuzione

4.2 Analisi

Dopo aver definito una metrica, passiamo a valutare il mio operato. L'interfaccia è di sicuro semplice ed intuitiva; sono poche le opzioni che lascia scegliere all'utente, direi quindi che questo punto è ben soddisfatto. Il problema è però quello della grafica, essa è grezza e dozzinale, ho infatti lasciato la grafica di base offerta dal sistema operativo. Ho preso questa scelta per concentrare maggiormente i miei sforzi sulla cura delle funzionalità. Sono riportate in figura 4.1, 4.2, 4.3 le varie schermate dell'applicazione per dare un'idea della loro struttura semplice ed intuitiva, ma si può vedere come la grafica sia stata lasciata in secondo piano.

Gli obiettivi dell'applicazione sono stati definiti nel capitolo 2 intitolato appunto obiettivi. Mi sembrano esposti in modo abbastanza chiaro e sintetico, per giunta alla fine del paragrafo 2.1 vengono anche riassunti in una lista le varie funzionalità dell'applicazione. Dagli obiettivi si evince anche chiaramente che l'ambito in cui si vuole piazzare questa applicazione è quello delle offerte commerciali che circondano l'utente, allo scopo è molto significativa la frase con cui esordisco nel paragrafo 2.1: "L'applicazione deve mettere a disposizione del proprio utente tutto ciò che "il mercato" intorno a lui sta offrendo in quel momento". Non ho comunque tralasciato quell'immancabile pizzico di social che ormai è insito in ogni applicazione dei giorni nostri. Quindi obiettivi semplici e contenuti e campo di azione ben definito, direi che anche questi punti son ben soddisfatto dalla mia applicazione.

Terminata la sezione sugli aspetti funzionali ed estetici passiamo ora alla parte sull'uso delle risorse. La quantità dei dati scambiati col server dipendono di volta in volta dalla dimensione della risposta dell'interrogazione al

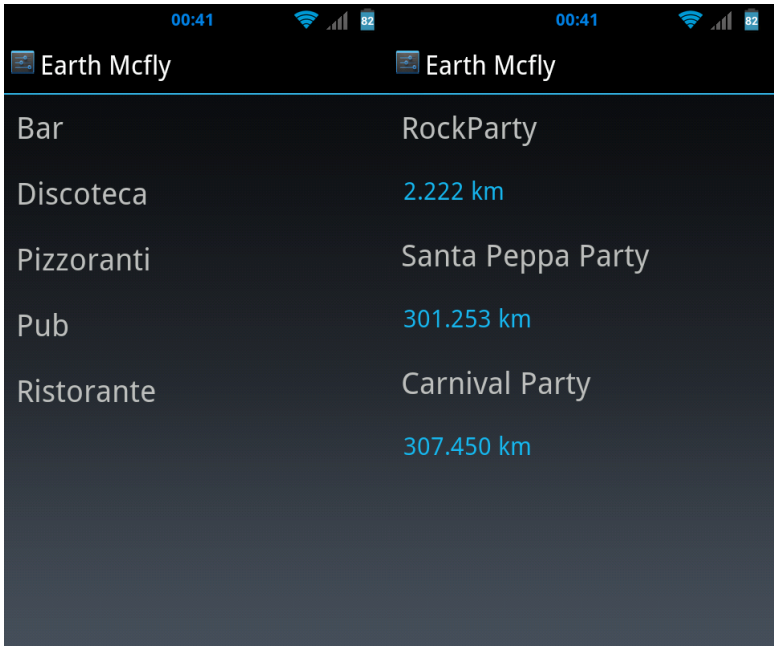


Figura 4.1: Rispettivamente 1- *CategoryListView.java* & 2- *ListDeals.java*

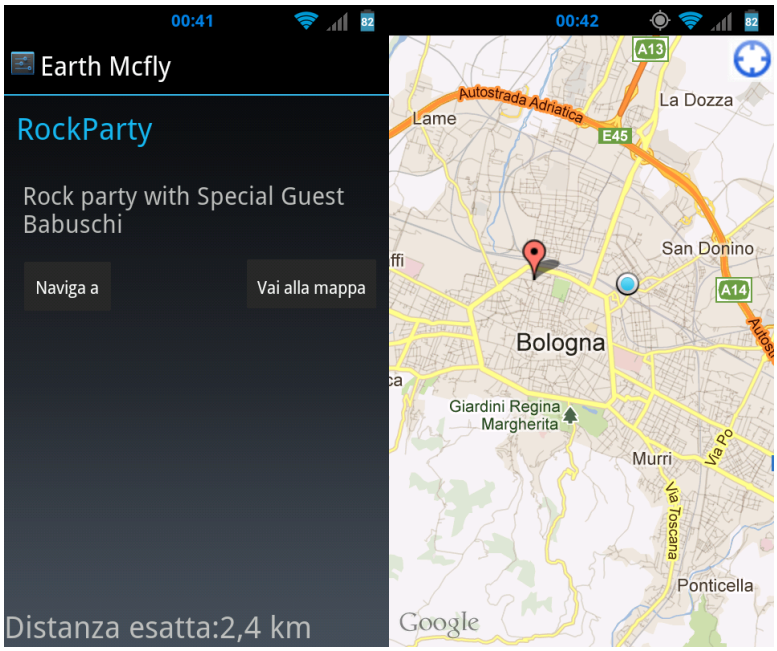


Figura 4.2: Rispettivamente 3- *EventDescription.java* & 4- *MyMapView.java*

Figura 4.3: I menu rispettivamente di 1- *CategoryListView.java* & 2- *EventDescription.java*

database. A questo stadio di sviluppo non sono in grado di fare un'analisi troppo dettagliata sulla quantità di dati richiesti in quanto per ora il database contiene pochissime righe, da me inserite, sufficienti a rendere possibile la fase di testing. La mia applicazione non ha un meccanismi di caching e quindi rispetto alle concorrenti che lo implementano, il mio elaborato risulterà molto più avaro in byte di trasmissione richiesti. Direi quindi che in una visione ottimistica, dove cioè la mia applicazione abbia preso piede, i dati scambiati di volta in volta sarebbero troppi e quindi non ho soddisfatto in pieno questo requisito.

Per quanto riguarda invece il discorso sull'uso di cpu e memoria mi ritengo soddisfatto in quanto dai test che ho eseguito non ho notato mai che la cpu raggiunga il 10% di utilizzo e la memoria superi i 20 MB. Il discorso però non è troppo diverso da quello sui byte necessari alle comunicazioni nel senso che non so di quanto potrebbe aumentare la cpu e la memoria necessari ad elaborare e mantenere temporaneamente i dati su cui si sta lavorando. Non

sono riuscito a trovare un modo per stimare la quantità di energia richiesta dal GPS però all'atto della stesura del codice ho cercato di attenermi alle linee guida diramate dalla stessa Google (`http://developer.android.com/reference/android/location/LocationManager.html`)).
Nel complesso sono comunque soddisfatto della velocità e fluidità dell'applicazione.

Conclusioni

Dopo diverso tempo di sviluppo sono arrivato in fondo. Non sono stato in grado di sviluppare tutto quello che mi sono prefisso negli obiettivi. Qualcosa è stato più complesso di quanto immaginavo ed il tempo non ha sicuramente giocato a mio favore. Non c'è stato tempo a sufficienza per implementare la gestione online degli account, ma è stato implementato solo il form per la registrazione. Questo per quanto riguarda il lato web, mentre per quanto riguarda il lato client non è stato possibile incorporare il canale dei feedback che sarebbe dovuto arrivare fino all'account del venditore, oltre che essere visibile ai clienti. Il problema principale è che la parte di gestione degli account richiede diverse tecnologie sia lato web che lato client e comunque non è un lavoro da prendere alla leggera. I dati degli utenti sono molto importanti; si sente quasi con frequenza giornaliera di database violati con conseguenti dati sottratti. Piuttosto che fare un lavoro spicciolo tanto per dire di averlo fatto ho preferito non inserirlo. Sarà comunque mia premura portare a termine questi obiettivi in quanto sono molto interessato all'ambiente mobile ed avendo inoltre fiducia nel progetto lo vorrei ultimare per renderlo disponibile al grande pubblico. Per il resto tutti gli altri punti degli obiettivi sono stati implementati, testati e ritenuti funzionanti.

Durante lo sviluppo, prendendo confidenza con l'ambiente Android, mi sono venute in mente alcune idee per migliorare ulteriormente l'applicazione. Esse andranno a far parte dei possibili sviluppi futuri.
Molto importante è introdurre un meccanismo di caching per risparmiare traffico che viene generato ogni qual volta che l'applicazione viene avviata.

Si potrebbe anche aggiungere una sezione per la chat tra utenti in un certo range.

Per dare un servizio più completo e utile sin dall'inizio si potrebbero integrare i locali che mette a disposizione Google. Ovviamente questi locali che verrebbero visualizzati non hanno dietro un account fatto dal proprietario del locale, quindi bisogna stare attenti a come gestire la questione del negoziante che crea l'account per un locale che era stato già aggiunto al servizio, ma che ancora non possedeva un account. Rimanendo in tema: qualche attività troppo recente o troppo piccola potrebbe non essere ancora segnata sui database di Google. In questo modo non sarebbe visualizzata all'interno della mia applicazione. Come soluzione si potrebbe adottare quella di lasciare all'utente la possibilità di segnalare un'attività e poi quando il proprietario farà l'account trattare la questione come descritto poco prima.

L'ultima cosa di cui mi sono reso conto durante lo sviluppo è che gli eventi dovranno avere una data di scadenza e non posso lasciarli sul database anche dopo tale data, altrimenti nel giro di poco tempo avrei bisogno di una quantità di spazio enorme ed inutile, ma soprattutto agli utenti sarebbero sottoposti eventi scaduti il che non avrebbe senso. È per questo che ho pensato che dovrà essere definito un qualche servizio che, in modo automatico, elimini quegli eventi che sono scaduti.