ALMA MATER STUDIORUM - UNIVERSITA' DI BOLOGNA

CAMPUS DI CESENA

SCUOLA DI SCIENZE

CORSO DI LAUREA IN SCIENZE E TECNOLOGIE INFORMATICHE

Applicazione mobile per la condivisione di interessi geolocalizzati

Relazione finale in

MOBILE WEB DESIGN

Relatore

Mirko Ravaioli

Presentata da

Emiliani Paolo

Sessione III

Anno accademico 2012/2013

Introduzione

In appena 18 anni i cellulari sono diventati sottili, veloci, intelligenti e poco telefoni. La comunicazione ormai è solo su canali dati, il futuro sembra proprio quello.

Dati significa rete. Ogni giorno siamo sempre più connessi.

Il web permea ogni secondo della nostra vita; ce ne accorgiamo quando infiliamo lo smartphone nella nostra tasca, quando siamo in treno, in strada, al cinema, persino a

Questa progressiva evoluzione del nostro modo di comunicare e connetterci con altre persone, aziende e servizi non è una prerogativa dei Paesi occidentali ma riguarda praticamente tutto il mondo. Quelle che prima erano informazioni ed applicazioni raggiungibili ed eseguibili esclusivamente da PC, ora sono accessibili da dispositivi sempre più potenti, con la caratteristica di essere portatili e di dimensioni notevolmente ridotte. Grazie alla loro grande versatilità e alla loro ampia dotazione di strumenti gli smartphone stanno diventando a tutti gli effetti delle estensioni della nostra persona. Ora il nostro device è il nostro portafogli, il nostro senso dell'orientamento e anche il nostro mezzo per comunicare, ma soprattutto è sempre con noi. Si potrebbe dire che il MAC address del nostro dispositivo sia un estensione del nostro codice fiscale, ci identifica, siamo il nostro telefono.

Questo incredibile trend va associato all'altrettanto sorprendente sviluppo e diffusione dei social network. Oggi il 35% della popolazione mondiale è connessa a internet e il 26% degli abitanti del pianeta ha almeno un account su un social network[i]. Questa situazione ha un impatto molto forte sulle abitudini di consumo e di relazione tra persone e brand in un mondo dove le informazioni sui nostri interessi sono una delle chiavi marketing più importanti.

L'obiettivo di questa tesi è quello di creare un'applicazione mobile che consenta ai dispositivi dove è installata la possibilità di identificarsi e quindi interagire fra loro, tutto all'interno di un raggio di azione ridotto. Una delle prerogative è che l'utente non debba effettuare nessuna ricerca ma essere informato automaticamente della vicinanza di un punto di interesse, nel nostro prototipo di un altro utente con cui condivide interessi. Per fare questo verrà utilizzata una tecnologia recente e in espansione, il Wi-Fi Direct.

La stesura del documento sarà organizzata secondo il seguente schema: nel Capitolo 1 verrà fatto uno studio sui dispositivi mobile, sull'influenza che hanno sulla vita di tutti i giorni, sui sistemi operativi mobile più utilizzati e sulle nuove tecnologie, in

particolare il Wi-Fi Direct. Nel secondo Capitolo verrà presentata la fase progettuale dell'applicazione nella quale saranno introdotte le principali funzioni dell'applicazione e descritte le specifiche di progettazione. Nel terzo Capitolo sarà descritta la fase implementativa, le classi e le funzioni per l'implementazione del server e dell'applicazione sul dispositivo. Seguiranno i possibili sviluppi futuri e le conclusioni.

Indice

Capitolo 1. Dispositivi Mobili

1.1 - Mondo Mobile: Smartphone

Smartphone, telefono cellulare con capacità di calcolo e connessioni molto più avanzate rispetto ai normali telefoni cellulari, questo è il nome della rivoluzione tecnologica ed economica che ha interessato l'inizio del nuovo secolo.

Dall'introduzione sul mercato del primo iPhone, nel 2007, e successivamente con l'introduzione di un store online e quindi la possibilità di scaricare applicazioni (app) aggiungendo funzionalità al proprio dispositivo lo sviluppo mobile non si è più fermato. Dal giorno del lancio ufficiale, avvenuto il 10 luglio 2008, l'App Store ha registrato tassi di crescita straordinari: nel giro di pochi mesi è passato da poche centinaia di applicazioni disponibili a circa 35.000 app e un milione di download effettuati nel mese di aprile 2009. Attualmente Apple ha raggiunto da poco la quota delle 900.000 app disponibili mentre Google Play (il market del principale concorrente) conta circa 732.000 applicazioni ed entrambi non sembrano accennare a fermarsi[1]. Queste straordinarie cifre rendono solo in minima parte l'idea dell'enorme business che si è sviluppato intorno al mondo delle app e delle potenzialità che questo offre a sviluppatori ed aziende in termini non solo di profitti economici.

I 100 milioni di device mobili venduti in tutto il mondo, contro i 93 milioni di pc, non rappresentano solamente una semplice inversione di tendenza a livello di mercato, ma una vera e propria svolta tecnologica e sociale, l'inizio di una nuova era.[2] Cambiano modalità di fruizione e contenuti, cambia il modo di pensare degli utenti, di interagire tra loro, di effettuare tutte quelle operazioni quotidiane che prima richiedevano, se non la presenza fisica in un determinato luogo, almeno l'utilizzo di un dispositivo fisso o relativamente ingombrante come computer e notebook.

La forza dei nuovi dispositivi mobili sta proprio qui, nella capacità di essere a portata di mano in qualsiasi momento e di dare all'utente la possibilità di gestire e consultare informazioni che fino a pochi anni fa implicavano una modalità di fruizione totalmente diversa. Queste potenzialità si sposano perfettamente con la necessità di connessione e con la globalizzazione della società moderna andando a creare nuovi bisogni, e quindi una nuova domanda.

In un ambiente già in continua evoluzione come quello elettronico informatico, dove le possibilità sembrano essere infinite, si è venuto a creare un nuovo mercato (che ha inglobato quello della telefonia mobile) dove le maggiori aziende produttrici di software, spinte dall'occasione di nuovi profitti, hanno largamente investito.

L’incredibile opportunità e quindi la concorrenza venutasi a creare hanno portato le aziende produttrici di hardware e software a dotare i dispositivi di sempre più funzionalità e integrando nuove caratteristiche hardware.

Un risultato possibile solamente grazie alla parallela evoluzione dei sistemi operativi mobili, che hanno radicalmente modificato la loro semplice struttura a favore di una più complessa che potesse dare all'utente la possibilità di interagire con il proprio dispositivo in modo intuitivo e garantendo, inoltre, un maggior livello di personalizzazione.

Ecco i prodotti dell’investimento di miliardi di dollari, in meno di una decina d’anni, in ricerca e sviluppo delle principali aziende di hardware e software.

1.2 - Sistemi Operativi: Ios, Windows Phone e Android

Negli ultimi tre anni sono state sviluppate oltre 300.000 applicazioni per i sistemi operativi iOS, Android e Windows Phone. Un mercato che varrà 37 miliardi di dollari nel 2015. [2]

Secondo le analisi della società di mercato IDC, i sistemi operativi di Google e Apple costituiscono il 95,7% di tutte le consegne smartphone del quarto trimestre 2013 e il 93,8% di tutte le consegne annuali. Si tratta di un incremento di 4,5 punti rispetto allo share del 91,2% che le due piattaforme hanno condiviso nel quarto trimestre del 2012, e una crescita di 6,1 punti rispetto all'87,7% dell'intero 2012. [3]

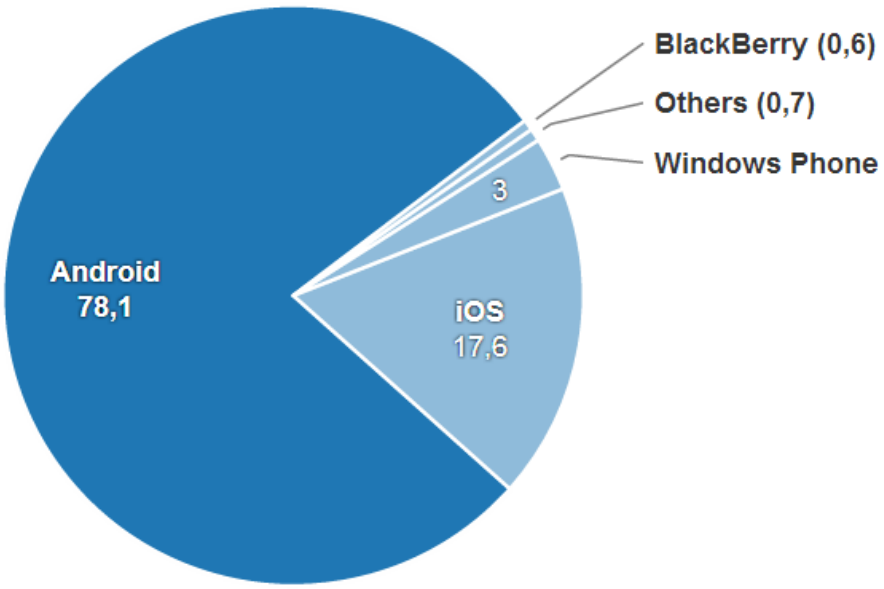


Figura n.1 – Diffusione sistemi operativi mobile inizio 2013 [3]

Il sistema operativo mobile più diffuso è quindi Android che vanta il primo posto con il del mercato mondiale, mantenendo la sua leadership grazie all'elevato numero di device sul quale è installato, mentre al secondo posto troviamo IOS con il 13,2% che perde qualche punto percentuale rispetto all'anno precedente.

Nonostante il suo rilascio sia avvenuto più recentemente rispetto ai già citati OS, Windows Phone ricopre già il terzo posto con il 3% destinato nei prossimi anni probabilmente ad aumentare grazie anche alla sua partnership/acquisizione di Nokia. Da segnalare come uno dei sistemi operativi storicamente più diffuso quale Symbian abbia subito un crollo tale da arrivare allo 0,2% del mercato mondiale. [4]

1.2.1 - iOS

E' sviluppato per una determinata serie di dispositivi quali iPhone, iPad, iPod touch ed Apple TV. Apple sviluppa sia l'hardware che il software dei suoi dispositivi e questo garantisce un'integrazione massima delle migliaia di app fornite dallo App store ufficiale, le uniche a poter essere installate. Per gli sviluppatori e a disposizione un SDK, previa registrazione a pagamento ad uno specifico programma di development Apple, che permette di creare applicazioni e testarle nel simulatore supportato dal kit. L'ambiente di sviluppo per l'SDK e XCode presente esclusivamente su computer Mac OS X. Le applicazioni interagiscono con l'hardware attraverso interfacce di sistema che le proteggono da modifiche hardware. Questo dona all'OS di Cupertino una grande semplicità di utilizzo ma ne limita pesantemente la personalizzazione. Un ulteriore vantaggio è dato dall'assenza di frammentazione, le applicazioni create per iOS sono infatti compatibili sin da subito con circa il 90% dei dispositivi montanti una qualsiasi versione del sistema operativo. [5]

App Store. Nonostante il primato di Android quanto a diffusione e numero di dispositivi disponibili sul mercato, l'App Store di Cupertino continua a macinare molti più utili rispetto a Google Play e rappresenta il vero valore aggiunto del prodotto di casa Apple.

1.2.2 - Windows Phone

Windows Phone è il sistema operativo per smartphone di Microsoft, presentato al Mobile World Congress il 15 febbraio 2010. La versione 7.0, rilasciata alla fine del 2010, rappresentò un decisivo distacco dalle precedenti versioni, non più compatibili, di Windows Mobile sia per quanto riguarda l'interfaccia grafica che per le funzionalità aggiuntive. Windows Phone 8 è l'ultima versione del sistema operativo Microsoft destinato agli smartphone ed è arrivato sul mercato il 29 ottobre 2012. Basato sul kernel Windows NT, lo stesso utilizzato da Windows 8 con cui condivide

anche gran parte della struttura. Da sottolineare una completa compatibilità con i servizi Microsoft. [6]

Il **Windows Phone Store** è veramente troppo acerbo, privo di contenuti e app con prezzi non allineati con la concorrenza, ed anche se in crescita non è paragonabile ai livelli degl'altri due. Questo è dovuto all'entrata con fin troppo in ritardo da parte di Microsoft nel mercato mobile.

Uno dei pochi pro è la possibilità di utilizzare per lo sviluppo linguaggi di programmazione standard come il C++ e C#.

1.2.3 - Android

Android è stato lanciato alla fine del 2007 da Google, insieme alla Open Handset Alliance (OHA). È un sistema open source realizzata allo scopo di migliorare l'esperienza mobile per gli utenti finali. Tra i suoi punti di forza c'è quello di poter essere installato su una vasta gamma di dispositivi, non solo smartphones ma anche tablet, e-book reader e netbook di diversi produttori. Questo ha favorito in breve tempo una rapida distribuzione ma ne ha anche aumentato la frammentazione. Per gli sviluppatori infatti risulta molto complicato realizzare applicazioni compatibili con tutti questi dispositivi che possiedono caratteristiche molto diverse tra loro.

Version	Codename	API	Distribution
2.2	Froyo	8	1.3%
2.3.3 - 2.3.7	Gingerbread	10	21.2%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	16.9%
4.1.x	Jelly Bean	16	35.9%
4.2.x		17	15.4%
4.3		18	7.8%
4.4	KitKat	19	1.4%

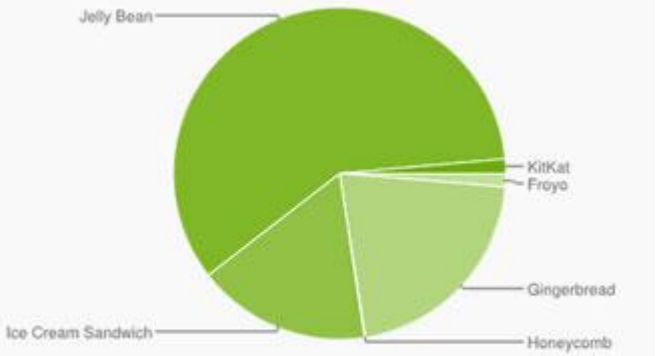


Figura n.2 – Resoconto frammentazione Android [6]

Seppure questo problema si sia con il tempo un ridotto è da sempre uno dei più grandi svantaggi del sistema operativo. La versione di Android attualmente più diffusa è la 4.1 Jelly Bean con una percentuale di installato rispetto al totale dei dispositivi dotati del sistema operativo mobile di Google del 35,9 %. La versione 4.2 raccoglie consensi per il 15,4% mentre la 4.3 raggiunge ora il 7,8%. In totale le tre release raccolte sotto il nome in codice di Jelly Bean riunisce sotto il proprio cappello il 59,1% di tutti i dispositivi del robottino verde. [7]

A livello tecnico, la piattaforma open source Android è di fatto uno stack, ovvero un set di sottosistemi software, basato sul kernel Linux e che è composto da applicazioni Java che vengono eseguite su uno speciale framework, basato anch'esso su Java e orientato agli oggetti, a sua volta eseguito su un nucleo costituito da librerie Java eseguite tramite la macchina virtuale Dalvik, specifica per dispositivi mobili, dotata di compilatore just-in-time (JIT). Infatti tutti gli aspetti del sistema operativo Android, da quelli più superficiali a quelli più critici possono essere modificati liberamente. Questo significa che è consentito ai produttori di dispositivi mobili di adattare il sistema operativo ai propri smartphone (per es. Samsung, HTC,..). Quanto detto vale sia per gli utenti finali che hanno la piena facoltà di personalizzare in ogni aspetto il prodotto acquistato che per gli sviluppatori che hanno la possibilità di rilasciare applicazioni in grado di modificare qualunque caratteristica del sistema.

Il **Google Play Store** è sicuramente il negozio online più fornito, numericamente parlando. Oltre a fornire una grande varietà di servizi, non solo app ma anche libri, noleggio e acquisto di contenuti multimediali e con una media prezzi inferiore ai concorrenti.

1.3 – Nuove Tecnologie

L'incredibile diffusione degli smartphone è stata resa possibile da uno straordinario sviluppo delle tecnologie wireless. Collegamenti dalle capacità sempre più elevate, protocolli sempre più ottimizzati, trasmettitori sempre più potenti, questo è il vero motore di questa rivoluzione tecnologica. Se si pensa che poco più di 20 anni si è passati dai 9,6 kbit/s che possiamo aspettarci da una rete GSM tradizionale alla velocità di trasferimento dati in download di 326 Mb/s che promette l'LTE, il perché di una così grande diffusione dei dispositivi mobile trova un'importante risposta. Ma non solo, oltre alle tecnologie satellitari e alla possibilità di fruire di sempre più dati, un altro settore di sviluppo del wireless è quello che ha come obiettivo la copertura delle aree locali, sia "in-door" sia "outdoor", il Wi-Fi. Un altro settore di sviluppo del wireless, forse il più suggestivo, è quello della "The Internet of Things". I calcolatori diventano così piccoli, potenti ed economici, come ad esempio le etichette intelligenti, RFId (Radio Frequency Identification), da poter essere immersi negli oggetti che ci circondano. Gli oggetti "intelligenti" sono capaci di comunicare col mondo esterno per realizzare vere e proprie reti di sensori. Queste reti immerse nello spazio reale sono a loro volta interconnesse con Internet e vi si può interagire.

L'impiego dei dispositivi mobili nelle proprie azioni quotidiane può essere un vero proprio modo per facilitarne lo svolgimento, oltre che per migliorarlo. Proprio per consentire lo svolgimento di numerose operazioni sui dispositivi l'industria

propone nuove tecnologie, nuovi strumenti da poter utilizzare mediante il software per compiere azioni davvero molto vaste.

1.3.1 - NFC

L'NFC (Near Field Communication) è uno degli ultimi standard in rapida evoluzione, che opera nella banda HF 13,56 MHz (valida in tutto il mondo) e per interazioni a corto raggio (<10 cm) e con velocità massima di trasmissione dati di 424 kBps.[8] La tecnologia NFC si è evoluta da una combinazione d'identificazione senza contatto o RFID (Radio Frequency Identification – Identificazione a Radio Frequenza) e altre tecnologie di connettività. Contrariamente ai più semplici dispositivi RFID, permette una comunicazione bidirezionale: quando due apparecchi NFC vengono accostati entro un raggio di 4 cm, viene creata una rete peer-to-peer tra i due ed entrambi possono inviare e ricevere informazioni. Questa tecnologia presenta enormi potenzialità ed ha trovato applicazione principalmente nel campo del Mobile Payment. I dati che transitano via NFC infatti non sono intercettabili (e quindi violabili) da nessuno che sia al di fuori dei 4 centimetri nei quali avviene la transazione. Il chip NFC è disponibile su una grande quantità di dispositivi (Google ne richiedeva la presenza per i dispositivi uscenti) ma non è presente sui dispositivi di casa Apple.

1.3.2 - Bluetooth Low Energy

Nelle telecomunicazioni Bluetooth è uno standard tecnico-industriale di trasmissione dati per reti personali senza fili (WPAN: Wireless Personal Area Network). Fornisce un metodo standard, economico e sicuro per scambiare informazioni tra dispositivi diversi attraverso una frequenza radio sicura a corto raggio. L'ultima versione del protocollo, la 4.0, viene definita Low Energy per i suoi consumi ridotti e offre opportunità davvero interessanti. I dispositivi che dispongono del supporto BLE, garantiscono un possibile utilizzo del Bluetooth che esula dalla preoccupazione di disattivare il Bluetooth ogni qualvolta si finisce di utilizzarlo. Questa nuova tecnologia, avrà bisogno di tempo per poter funzionare a tutti gli effetti, in quanto ogni produttore dovrà separatamente dare supporto al BLE facendo diventare compatibili tutte le app proprietarie che sfruttano il Bluetooth, sono solo pochi attualmente gli smartphone che sarebbero pronti ad utilizzare questa tecnologia.[9] In questo senso però Apple ha investito in questa tecnologia inserendo nel suo ultimo sistema operativo iBeacon, una tecnologia prevalentemente indirizzata alle attività commerciali ma dal grande potenziale. iBeacon sfrutta la tecnologia Bluetooth Smart introdotta proprio con lo standard 4.0. Questo consente il trasferimento di dati elaborati tramite un posizionamento GPS, ottimizzando ovviamente i consumi energetici su smartphone e tablet. Tramite una connessione avvenuta,

e un'applicazione iBeacon-ready predisposta dall'attività commerciale è possibile ricevere determinati contenuti in base al contesto in cui ci si trova.[10] I futuri utilizzi di iBeacon o di questo genere di approccio al consumatore possono essere davvero tanti, anche se è difficile allontanarsi al momento dal pensiero che questi saranno strettamente correlati all'utilizzo negli esercizi commerciali. Bisogna ricordare inoltre che per questo genere di approccio è necessaria l'installazione di alcune antenne, i contenuti che verranno visualizzati sono infatti all'interno del messaggio inviato dal trasmettitore.

1.3.3 – Wi-Fi Direct

Wi-Fi Direct permette a due dispositivi di stabilire una connessione diretta peer-to-peer in Wi-Fi senza necessità di un router wireless. In questo modo il Wi-Fi diventa un metodo di comunicazione tra dispositivi in modalità wireless, come Bluetooth. Il Wi-Fi Direct è supportato a partire da Android 4 Ice Cream Sandwich ed è uno standard (gestito dalla Wi-Fi Alliance) che non richiede hardware specifico per il suo funzionamento, se non la presenza di un'antenna Wi-Fi standard. La connessione può essere più o meno complessa in base al tipo di utilizzo che se ne farà. Uno degli utilizzi più semplici (e anche più efficaci) è quello dello scambio file, che permette a due dispositivi di scambiarsi file anche molto grossi fino a 300Mbps (in Wi-Fi 802.11n) e si capisce subito come questo sistema possa andare presto a sostituire lo scambio dati tramite Bluetooth, sia in velocità che in semplicità di utilizzo.[11] Ma non ha solo questa caratteristica, infatti al fine di stabilire la connessione offre un servizio di scanning dei dispositivi supportanti questa tecnologia in grado appunto di rilevarne la presenza. Questa funzione può rivelarsi molto utile in un ottica di mobile o di proximity marketing, uno dei mezzi più efficaci per coinvolgere potenziali clienti interessati ai prodotti e servizi perché agganciati direttamente nei pressi dell'attività commerciale. Nonostante lo scarso blasone il Wi-Fi Direct resta sicuramente una caratteristica promettente, inclusa sia nei moderni sistemi operativi Android sia in iOS, che sicuramente in futuro sarà usata da sempre più dispositivi per le connessioni dirette Wi-Fi e non solo. Negl'ultimi dispositivi come per esempio tutti gl'ultimi usciti da casa Samsung, Htc e LG è sufficiente avere il Wi-Fi attivo per poter essere raggiungibili dallo scanning Direct. Questo offre grandi possibilità.

1.3.4 - Geolocalizzazione: Servizi Basati Sulla Posizione

La maggior parte degli smartphone di ultima generazione è dotata di GPS, è possibile sfruttare questo strumento, oltre che alla connessione dati o quella Wi-Fi per geolocalizzare il telefono. La geolocalizzazione è l'identificazione della posizione geografica (latitudine e longitudine) di una persona, di un oggetto in un dato momento. Con la diffusione dei sistemi GPS, degli smartphone e di internet la

geolocalizzazione diventa importante in quanto comunicando la propria posizione è possibile ottenere ad esempio informazioni dalla rete su tutto quello che ci circonda.

- Geolocalizzazione tramite GPS (Global Positioning System)

La localizzazione tramite GPS viene effettuata sfruttando la ricezione di segnali provenienti da vari satelliti che si trovano in orbita: Il principio del GPS è abbastanza semplice: attraverso i segnali ricevuti dai satelliti in orbita (almeno 4) il dispositivo riesce a calcolare la propria posizione conoscendo il tempo di volo del segnali, la mappa dei satelliti e la loro sincronizzazione temporale. Il suo grande vantaggio è la precisione che può arrivare al metro. Lo svantaggio di questa tecnologia è che non può funzionare dentro agli edifici.

- Geolocalizzazione tramite rete GSM

Il processo di localizzazione si basa sulla possibilità del GSM di conoscere la distanza approssimativa dispositivo collegato alla rete GSM dalla stazione radio base con la quale è connesso. Ripetendo in sequenza l'operazione di stima da più basi di trasmissione circostanti, ed effettuando alcune operazioni di triangolazione matematica è possibile stimare con buona precisione la posizione sul territorio del telefonino.

- Geolocalizzazione tramite rete Wi-Fi

Questo sistema di geolocalizzazione sfrutta la presenza di access point WiFi sparsi nel territorio: conoscendo l’esatta ubicazione degli access point ricevuti dal proprio terminale, è possibile determinare la posizione con un’accuratezza di poche decine di metri. Questo sistema è chiamato anche WPS (Wireless Positioning System).

- Geolocalizzazione tramite rete internet

Questo sistema sfrutta il solo indirizzo IP per rilevare la posizione. Ogni provider ha un range di indirizzi IP a sua volta suddiviso in sottogruppi di indirizzi IP che servono determinate regioni e province. Pertanto conoscendo un indirizzo IP possiamo visualizzare l’area approssimativa del dispositivo connesso ad internet.

Capitolo 2. Progettazione

2.1 - Descrizione Generale

Il progetto che si andrà a sviluppare è un applicazione dal nome, non definitivo, "Who Is Around" per smartphone Android che permetta all'utente di interagire con l'ambiente circostante senza dover utilizzare lo smartphone, avendolo semplicemente con sé, in tasca o nella borsa.

Nella fatti specie si cercherà di implementare questo meccanismo con l'obiettivo di condividere i propri interessi e di venire a conoscenza se nelle immediate vicinanze esiste un altro utente con cui ha in comune alcuni di questi.

L'idea che sta dietro all'applicazione quindi è quella che all'utente che la installi e registri il proprio profilo venga notificato la presenza nel luogo in cui si trova di un utente con interessi affini.

Analisi

Il funzionamento di base del sistema prevede quindi l'utilizzo dello smartphone per rintracciare i dispositivi a lui vicini, identificarli in modo univoco ed interfacciarsi ad un server che controlla se questi siano effettivamente registrati come utenti e quindi la presenza di interessi in comune tra i due utenti. Se il controllo ha riscontro positivo il server invia i dati del profilo incontrato allo smartphone che ne salva posizione, orario e informazioni condivise e lo notifica all'utente. Nel dispositivo viene mantenuta traccia degli incontri passati, che vengono suddivisi in "New" e "All", di ogni nuovo utente (con interessi in comune) incontrato. Inoltre salviamo i dati principali di quell'utente e lo aggiungiamo alla lista contatti. Per ogni contatto sarà visualizzabile l'immagine profilo, l'username, l'email, gli interessi in comune e la lista degli incontri precedenti per quel contatto. In ogni incontro sarà visualizzata la posizione su una mappa e visualizzate le informazioni sulla data e l'username dell'utente incontrato. In fase di registrazione un utente sceglierà un username e assocerà il proprio profilo ad un contatto email. Successivamente potrà aggiungere e modificare la propria lista di interessi. Gli interessi possono essere scelti da una lista, che, per poter effettuare il match, deve essere comune a tutti gli utenti. Se non presente un utente può aggiungere un interesse, questo verrà aggiunto alla lista e quindi potrà essere scelto anche dagl'altri utenti.

2.2 - Trovare Gli Utenti: Wifi Direct, Caratteristiche

Per riuscire nell'intento il primo problema affrontato è stato quello di rilevare i dispositivi con cui si veniva a contatto. I moderni smartphone mettono a disposizione

varie alternative. In primo luogo si è presa in esame la possibilità di utilizzare il Bluetooth. Lo scanning Bluetooth infatti permetteva di ricevere la lista dei dispositivi, ottenendo il MAC Address di quelli con la funzione Bluetooth attiva, nel raggio d'azione utile della tecnologia (circa 30 metri). Questo però avveniva soltanto nel caso in cui nel dispositivo fosse stata attivata l'opzione "Rilevabile", stato in cui è possibile passare solo con consenso esplicito dell'utente. Modalità che può avere una durata massima di 5 minuti allo scadere dei quali l'utente avrebbe dovuto riconfermare la volontà di poter esser trovato. Questa limitazione ostacolava l'implementazione di una delle principali caratteristiche dell'applicazione ed è stata per forza di cose abbandonata (la non interazione dell'utente).

Un'altra tecnologia che consente il rilevamento di dispositivi vicini è il Wi-Fi Direct. In realtà il fine ultimo del Wi-Fi Direct è la creazione di reti P2P con la finalità di utilizzare la connettività Wi-Fi per scambiarsi dati, è ovvio allora che per instaurare un canale per lo scambio di dati ci sia bisogno in primo luogo di stabilire una connessione tra i dispositivi [12].

Utilizzeremo quindi la ricerca del modulo Wi-Fi Direct, messa a disposizione dalle Api di Android, per rintracciare gli utenti. Essa permette infatti di rilevare la presenza di altri dispositivi supportanti la stessa tecnologia in un raggio d'azione di un centinaio di metri.

Inoltre con questa tecnologia si è rilevabili nella maggior parte dei casi semplicemente avendo il modulo Wi-Fi attivo.

Questa integrazione e essendo a tutti gli effetti semplicemente un'estensione del Wi-Fi hanno fatto del Wi-Fi Direct la scelta per l'implementazione delle funzionalità dell'applicazione.

Per identificare univocamente un dispositivo e quindi un utente, ho pensato di utilizzare il Mac Address del telefono. Questo infatti si prestava perfettamente per le caratteristiche di univocità e reperibilità ad essere utilizzato come identificativo per l'utente.

Al momento della registrazione l'utente invia oltre ai suoi dati personali che vuole condividere anche il MAC Address della propria scheda Wi-Fi, in questo modo al suo indirizzo sarà associato il suo profilo. Ogni volta che durante una scansione un utente troverà dei dispositivi abilitati e con il Wi-Fi Direct attivo effettuerà una richiesta al server domandando se a quel Mac Address corrisponda un utente registrato.

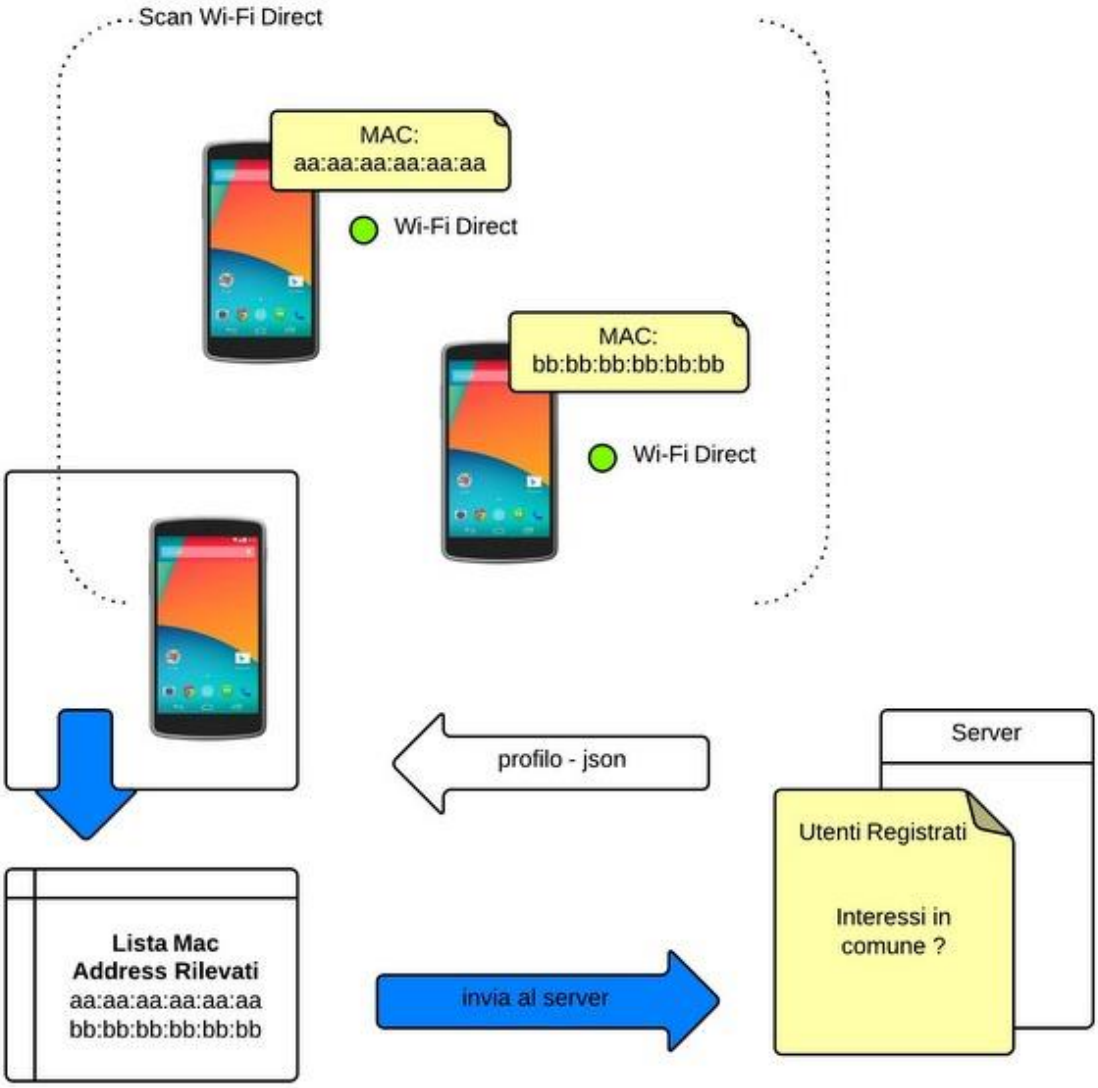


Figura n. 3 - Funzionamento dell'applicazione

2.3 – Logica Parte Client: Applicazione

La parte client ovvero l'applicazione Android si comporrà di una parte che girerà in background, che ogni certo quantitativo di tempo effettuerà uno scanning per verificare la presenza di dispositivi attivi, e di una serie di activity adibite alla navigazione e all'interazione con i dati.

In Background

BroadcastReceiver – WifiDirectBroadcastReceiver

Un Broadcast Receiver permette di ricevere gli intenti trasmessi dal sistema Android, in modo che l'applicazione possa rispondere a eventi ci interessano [13].

Viene registrato dal servizio e ha il compito di controllare i dispositivi rilevabili e di mantenere una lista dei dispositivi disponibili incontrati. Infatti rileva i cambiamenti relativi ai peers disponibili, allo stato del telefono e alla connessione.

Servizio – MyService

Un servizio è un componente di un'applicazione in grado di eseguire operazioni prolungate in background e non fornisce un'interfaccia utente. Un componente dell'applicazione può avviare un servizio e questo continuerà a funzionare in background anche se l'utente passa a un'altra applicazione. Non è quindi legato al fatto che l'applicazione sia in esecuzione.

Caratteristiche perfette per notificare all'utente la presenza di un altro senza che interagisca direttamente con l'applicazione.

Utilizziamo quindi un service che ha il compito di attivarsi ogni minuto e di controllare se sono stati rilevati dispositivi nella lista dei MAC Address dei dispositivi trovati del Broadcast Receiver.

Se ci sono dispositivi, il MyService salva la posizione attuale tramite il GPS o la tipologia di localizzazione attiva al momento (se attiva) e passa la lista ad un processo parallelo che la invia al server tramite connessione http insieme al proprio MAC Address. Se la risposta ha prodotto dei risultati verrà aggiunto un nuovo contatto e un nuovo incontro se l'utente non era già stato incontrato, altrimenti solamente un nuovo incontro. Operazione che viene notificata all'utente.

Quando viene stoppato dall'utente il servizio non sarà più attivo e non verrà riavviato ogni minuto.

Database interno all'applicazione

Nell'applicazione avremo bisogno di mantenere i dati sugl'incontri e sui contatti. I contatti (contacts) sono i dati degli utenti associati al dispositivo rilevato. Gli incontri (meeting) sono invece i dettagli relativi a data e luogo in cui si ha rilevato un contatto.

Operazioni saranno relative a:

- Aggiunta nuovo contatto
- Aggiunta nuovo incontro
- Preleva tutti gli incontri per un contatto
- Preleva i nuovi e tutti gli incontri.

TABLE_CONTACTS (id, id_us, MAC, name, email)

id: identificativo univoco nel database sull'applicazione;

id_us: mantengo l'identificativo che quell'utente ha sul server;

MAC: contiene il MAC address del dispositivo di quell'utente;

name: contiene l'username con cui si è registrato il contatto;

email: contiene l'email con cui si è registrato il contatto;

TABLE_MEETING (id_m, mac_u, name_u, date, latitude, longitude)

id_m: identificativo univoco dell'incontro sul database;

mac_u: contiene il MAC address dell'utente incontrato;

name_u: contiene l'username dell'utente incontrato;

date: mantiene la data e l'ora dell'incontro;

latitude e longitude: mantengono le coordinate di latitudine e longitudine al momento dell'incontro;

Struttura delle pagine

Un'applicazione consiste solitamente un insieme di activity che sono liberamente associati tra loro. Un Activity è un componente di un'applicazione che fornisce una schermata con cui gli utenti possono interagire al fine di fare qualcosa, come comporre il telefono, scattare una foto, inviare una mail o visualizzare una mappa.

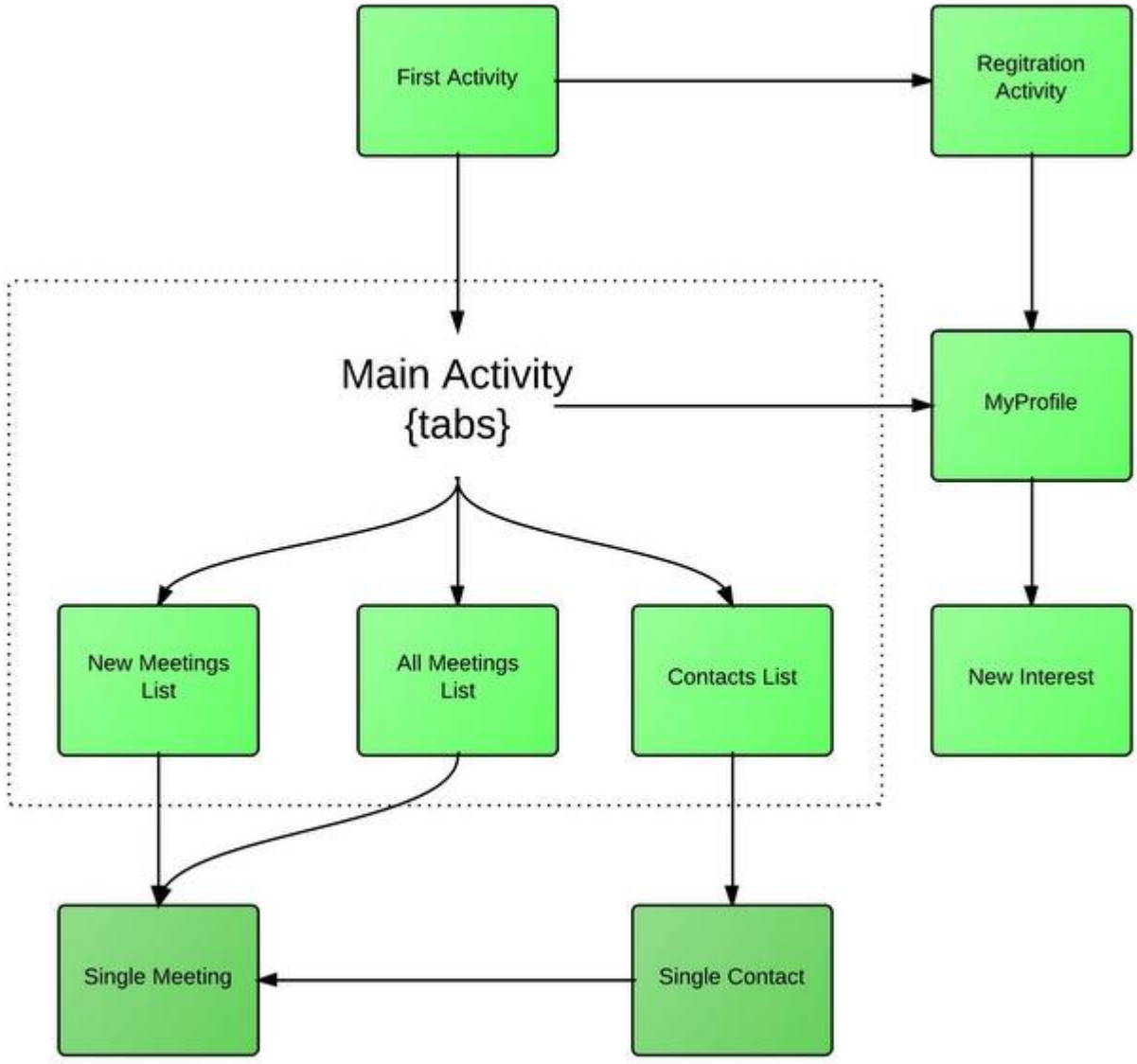


Figura n.4 – Struttura delle pagine dell'applicazione

First Activity: L'activity incaricata di controllare se l'utente è registrato o meno all'apertura dell'applicazione. Se l'utente è già registrato sarà indirizzato verso la Pagina Principale (Main Activity) altrimenti alla pagina di Registrazione.

Pagina Principale. Conterrà le 3 liste relative ai nuovi incontri, storico degli incontri e la lista dei contatti, oltre alla possibilità di accedere alla pagina di gestione del proprio profilo e di attivare/disattivare il service di ricerca.

Pagina di Registrazione: l'utente non ancora registrato può inserire i suoi dati per registrarsi, username e email, verrà poi indirizzato verso la pagina del suo profilo per aggiungere interessi e modificare i dati.

Lista Nuovi Incontri (New): qui dentro verranno visualizzati i nuovi incontri. Saranno presenti in questa lista solo gli ultimi incontri non ancora visualizzati, dopo di che saranno visualizzati nella lista "Tutti gli incontri" (All).

Lista Tutti gli incontri (All): qui saranno visualizzato lo storico degli incontri avvenuti, selezionando un incontro sarà possibile visualizzarne i dettagli (Dettagli Incontro).

Lista Contatti (Contacts): qui sarà visualizzata la lista dei contatti, cliccando su un contatto sarà possibile visualizzarne i dettagli (Dettagli Contatto).

Dettagli Incontro (Single Meeting): activity dove saranno visualizzati i dettagli di un singolo incontro, la posizione, la data e l'ora e l'utente incontrato.

Dettagli Contatto (Single Contact): activity dove saranno visualizzati i dettagli di un singolo contatto, l'username, l'email, eventuali altri dati, gli interessi in comune e la lista degli incontri passati, cliccando su un incontro sarà possibile accedere ai dettagli di quell'incontro.

Gestione Profilo (MyProfile): in questa activity sarà possibile modificare il proprio profilo e modificare la lista dei propri interessi, da qui è possibile raggiungere la pagina di aggiunta e modifica degli interessi.

Nuovo Interesse (NewInterest): in questa activity sarà possibile aggiungere interesse alla propria lista e crearne se non disponibili nella lista di quelli selezionabili.

2.4 - Logica Parte Server: Server Web

Per gestire la parte del riconoscimento del MacAddress e associarlo ad un utente utilizzeremo un Web Server che dovrà prima tutto organizzare i dati dei vari utenti registrati per poi gestire le richieste di dati provenienti dalle applicazioni.

La logica lato server si basa su un servizio web, sviluppato appositamente per l'applicazione, collegato alla gestione di un database, attraverso il quale sono ottenute le informazioni aggiornate.

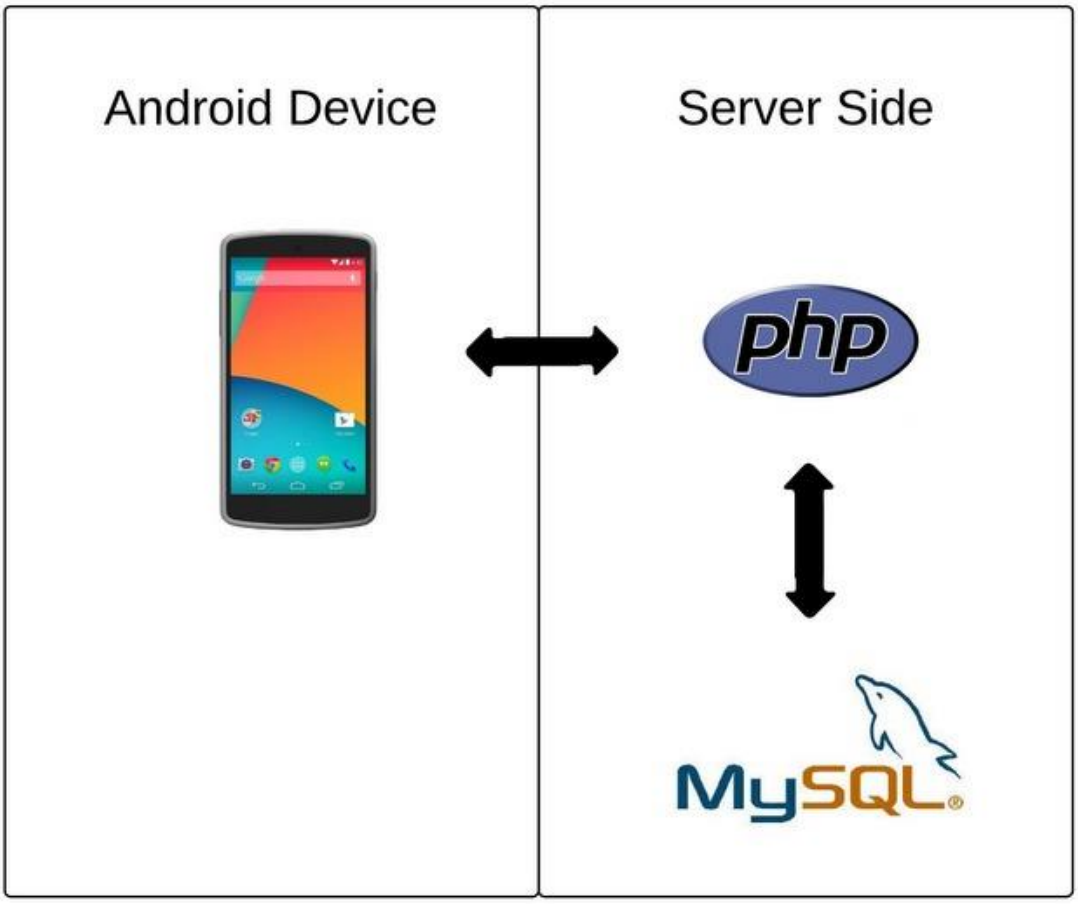


Figura n.5 – Logica Server Web

Le richieste al server serviranno per:

- prelevare i dati di un singolo utente (sia del proprio profilo che di quello dei propri contatti);
- controllare se esiste affinità tra l'utente che invia e i MAC address riscontrati;
- registrarsi e cancellarsi;
- modificare la propria lista interessi e aggiungerne di nuovi.

Il dispositivo effettua una richiesta al server remoto; il server in base alla tipologia di richiesta apre una nuova connessione al database ed esegue su di esso una specifica query; il server riceve i dati dal database e li ritorna al client.

I dati sono ritrasmessi in file di tipo JSON (Javascript Object Notation) che è un semplice formato fra i più utilizzati per quanto riguarda lo scambio dati in applicazioni Client-Server.

Standard JSON

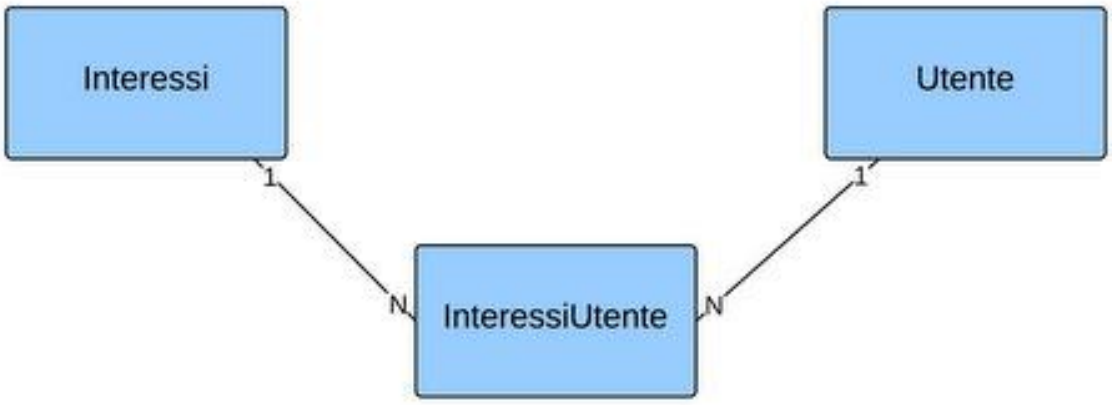
JSON è uno standard aperto basato sul testo e adatto per lo scambio di dati in applicazioni client-server. Infatti è un efficiente formato di codifica dati che consente scambi rapidi di piccole quantità di dati tra client e servizi web[14]. Ovviamente JSON è completamente indipendente dal tipo di linguaggio di programmazione utilizzato. Inoltre è ampliamente supportato dalla maggior parte dei linguaggi di programmazione grazie alla disponibilità di numerose librerie. Quindi può essere usato come standard per la comunicazione tra software sviluppati su linguaggi e piattaforme differenti fra loro.

Rispetto ai formati binari ha il vantaggio di essere facilmente comprensibile, quindi rende più facile lo sviluppo ed il debug dei protocolli di comunicazione che lo utilizzano. Rispetto ad altri formati di rappresentazione dei dati text-based come XML, ha il vantaggio di essere molto più "leggero", cioè di rappresentare gli stessi dati con un numero inferiore di caratteri, in modo da essere più leggibile e meno pesante da essere trasferito via rete [15].

Le informazioni inviate dal server al dispositivo hanno come struttura un oggetto composto da un messaggio di esito (“successo” o “errore”) e un contenuto che contiene le coppie chiave/valore relative ai dati se l’esito è positivo, oppure una stringa corrispondente alla descrizione dell’errore in caso di esito negativo.

```
{
"found":1,

"contacts":[
       {"id_us":"6",
       "MAC":"aa:27:65:dc:b6:a8",
       "name":"Username",
       "email":"Email@email.it"},
       "interessi":["interesse1","interesse2"]

       ]
}
```

Sul database del server abbiamo bisogno delle informazioni per mantenere gli interessi che gli utenti possono scegliere, i dati relativi agli utenti e quali interessi hanno.



TABLE_INTERESSI (**id_i**, i_nome)

id_i: identificativo univoco dell'interesse;

id_nome: descrizione dell'interesse .

TABLE_UTENTI (**id_u**, MAC, username, email)

id_u: identificativo dell'utente sul server;

MAC: identificativo del cellulare dell'utente ;

username: nome con cui si registra l'utente;

email: contatto email con la quale l'utente si iscrive e si pensa che sia raggiungibile.

TABLE_ INTERESSIUTENTE (**id_i, id_u**, nomeinteresse)

id_i: codice identificativo dell'interesse;

id_u: codice identificativo dell'utente;

nomeinteresse: descrizione dell'interessi.

Capitolo 3. Implementazione

3.1 - Perché Android?

Per la realizzazione di questa applicazione è stato scelto il sistema operativo Android sia per il suo notevole progresso nel mercato mobile negli ultimi anni sia per la maggiore accessibilità del suo ambiente di sviluppo. Nel web ormai si è creata una comunità di sviluppatori di notevoli dimensioni che è spesso molto utile per risolvere alcuni inevitabili problemi durante la realizzazione del progetto. A differenza di altri sistemi operativi, Android offre inoltre una serie di tools di sviluppo completamente open-source, utilissimi per lo sviluppo dell'applicazione.

3.2 - App/Client: Introduzione

In particolare per lo sviluppo dell'applicazione è stato utilizzato l'IDE Eclipse sfruttando l'integrazione che questo software ha con Android installando l'ADT Plugin che permette di avere un ambiente di sviluppo professionale ma allo stesso tempo completamente free e opensource.

3.2.1 - Database interno all'app: SQLite

SQLite è un leggerissimo database engine transazionale che occupa poco spazio in memoria e sul disco, pertanto è la tecnologia perfetta per creare e gestire database in un ambiente come quello degli applicativi mobile, dove le risorse sono molto limitate e dunque è molto importante ottimizzarne l'utilizzo[16].

Un database SQLite è, nella pratica, un file: è possibile spostarlo, copiarlo in un altro sistema e continuerà a funzionare tutto regolarmente. Android memorizza i file seguendo uno schema preciso; il file SQLite del database di ogni applicazione viene infatti memorizzato in una directory il cui percorso è: /data/data/packagename/databases dove "packagename" è il nome del package del corrispondente applicativo. Il codice SQL per la creazione delle tabelle:

```
String CREATE_CONTACT_TABLE = "CREATE TABLE CONTACTS (
          Id_c  INTEGER PRIMARY KEY,
          Id_us INTEGER,
          MAC TEXT UNIQUE,
          Username TEXT,
          Email TEXT UNIQUE )";

String CREATE_MEETING_TABLE = "CREATE TABLE MEETINGS (
        Id_m INTEGER PRIMARY KEY,
        Id_us INTEGER,
        Username TEXT,
        MacU TEXT,
        Date LONG,
        Latitude DOUBLE,
        Longitude DOUBLE);
```

Per le interazioni con il database è stata creata la classe DatabaseHandler estendendo la classe *SQLiteOpenHelper* disponibile nel package *android.database.sqlite* con il fine di utilizzare i suoi metodi per la gestione del database.

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

db.execSQL(CREATE_MEETING_TABLE);
db.execSQL(CREATE_CONTACTS_TABLE);
}
```

DBHelper.java

3.2.2 - Gestione dei collegamenti con il Server

Per la gestione della connessione con il server e per inviare e ricevere dati è stata creata la classe JSONParser. Essa utilizza le librerie offerte da Android e apache per l'interfacciamento con la propria tipologia di server.

La classe JSONParser si occupa quindi di inviare al server una lista di coppie/valore e di gestirne la risposta restituendo un oggetto JSON.

```
public JSONObject getJSONFromUrl(String URL, List<NameValuePair> params) {

        // Richiesta HTTP al server
        try {

            DefaultHttpClient httpClient = new DefaultHttpClient();
            HttpPost httpPost = new HttpPost(URL);
            httpPost.setEntity(new UrlEncodedFormEntity(params));
            HttpResponse httpResponse = httpClient.execute(httpPost);
            HttpEntity httpEntity = httpResponse.getEntity();
            is = httpEntity.getContent();

             ...
             (analisi dell'inputstream che viene inserito nella Stringa json )
             ...

        // Parsing della stringa nell'oggetto JSON
        try {
            jObj = new JSONObject(json);
        } catch (JSONException e) {
            Log.e("JSON Parser", "Error parsing data " + e.toString());
        }

        //restituisce l'oggetto JSON
        return jObj;
}
```

JSONParser.java

Il download sarà gestito dal metodo *getJSONFromUrl* che prenderà in ingresso l'URL della pagina php specifica per la funzione richiesta e la lista di coppie chiave/valore con i dati di input necessari al server per estrapolare le informazioni dal database, per esempio i MAC Address appena rilevati saranno inviati alla pagina php "check.php" all'indirizzo (http://whoisaround.altervista.org/pEServerTesi/check.php).

3.2.3 - Acquisizione della posizione: Geolocalizzazione

Al momento, i dispositivi Android sono in grado di acquisire le informazioni relative alla posizione attraverso due tipi di provider: GPS o basato sulla rete. I primi utilizzano il segnale proveniente dai satelliti, mentre i secondi calcolano la posizione attraverso triangolazioni relative a posizioni note [17].

Nell'applicazione la posizione viene salvata quando si è appena riscontrato la presenza di un utente registrato e con interessi in comune. Le informazioni relative alla posizione possono essere ottenute LocationManager in modi diversi a seconda dei LocationProvider disponibili, per ottenere le coordinate di Latitudine e Longitudine del punto in cui si è incontrato l'utente utilizziamo la classe GPSTracker ottenuta implementando appunto l'interfaccia LocationListener.

```
public class GPSTracker extends Service implements LocationListener {


	// Distanza minima tra gli aggiornamenti in metri
	private static final long MIN_DISTANCE_CHANGE_FOR_UPDATES = 10; // 10 meters

	// Tempo minimo tra gli aggiornamenti in millisecondi
	private static final long MIN_TIME_BW_UPDATES = 1000 * 60 * 1; // 1 minute

	// Dichiarazione Location Manager
	protected LocationManager locationManager;

	...

try {
	locationManager = (LocationManager) mContext
				.getSystemService(LOCATION_SERVICE);

	// ottengo lo stato del GPS
	isGPSEnabled = locationManager
				.isProviderEnabled(LocationManager.GPS_PROVIDER);

	// ottengo lo stato del network
	isNetworkEnabled = locationManager
				.isProviderEnabled(LocationManager.NETWORK_PROVIDER);

	if (!isGPSEnabled && !isNetworkEnabled) {
		// nessun network provider abilitato
			} else {
```

GPSTracker.java

GPSTracker.java

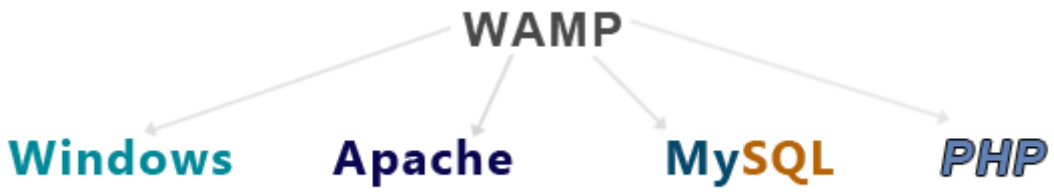
```
} else {
            this.canGetLocation = true;
            if (isNetworkEnabled) {
                  locationManager.requestLocationUpdates(
                                    LocationManager.NETWORK_PROVIDER,
                                    MIN_TIME_BW_UPDATES,
                                    MIN_DISTANCE_CHANGE_FOR_UPDATES, this);
                  if (locationManager != null) {
                        location = locationManager
                        .getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
                        if (location != null) {
                              latitude = location.getLatitude();
                              longitude = location.getLongitude();
                        }
                  }
            }
            // Se il GPS è abilitato ottieni lat/long tramite il servizio GPS
            if (isGPSEnabled) {
                  if (location == null) {
                        locationManager.requestLocationUpdates(
                                          LocationManager.GPS_PROVIDER,
                                          MIN_TIME_BW_UPDATES,
                                          MIN_DISTANCE_CHANGE_FOR_UPDATES, this);
                        if (locationManager != null) {
                              location = locationManager
                              .getLastKnownLocation(LocationManager.GPS_PROVIDER);
                              if (location != null) {
                                    latitude = location.getLatitude();
                                    longitude = location.getLongitude();
                                          }
                              }
                        }

                  ...

            //Restituisco la posizione
            return location;
      }
```

L'utilizzo del LocationManager è molto semplice, in quanto, per essere notificati delle variazioni di posizione, è sufficiente creare un'implementazione dell'interfaccia LocationListener e registrarsi a essa come ascoltatori attraverso il seguente metodo: *public void requestLocationUpdates (String provider, long minTime, float minDistance, LocationListener listener)*. Si noti come il primo parametro sia l'identificativo del provider che si intende utilizzare. Questo significa che è possibile registrarsi come ricevitore degli eventi di posizione solo con un provider. I parametri *minTime* e *minDistance* permettono di specificare rispettivamente una distanza ed un tempo minimo di notifica; questo permette di non generare troppi eventi di notifica ravvicinati sia nel tempo, sia nello spazio, in modo da preservare le risorse del dispositivo.

3.3 - Parte Server: Introduzione

Per la parte server è stato costruito un server web basato su piattaforma Apache utilizzando il linguaggio PHP quindi collegandolo ad un database MySQL.

Le prove sono state effettuate il localhost ma ora il sistema è funzionante e raggiungibile dalla rete.



La funzione del server è quindi ricevere la richiesta dal client (applicazione android) elaborare le coppie di valori, collegarsi al database, prelevare i dati richiesti e inviare una risposta in formato JSON.

3.3.1 - Creazione del database

Il database per la gestione degli utenti e degli interessi viene creato in MySQL.

Il codice SQL per la creazione delle tabelle in questione è il seguente:

```
CREATE TABLE IF NOT EXISTS `utenti` (
  `id_utente` int(11) NOT NULL AUTO_INCREMENT,
  `MAC` varchar(50) NOT NULL,
  `username` varchar(40) NOT NULL,
  `email` varchar(40) NOT NULL,
  PRIMARY KEY (`id_utente`),
  UNIQUE KEY `MAC` (`MAC`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS `interessiutente` (
  `id_iu` int(11) NOT NULL AUTO_INCREMENT,
  `id_i` int(11) NOT NULL,
  `id_u` int(11) NOT NULL,
  `nomeinteresse` varchar(30) NOT NULL,
  PRIMARY KEY (`id_iu`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS `interessi` (
  `id_interesse` int(11) NOT NULL AUTO_INCREMENT,
  `nome_i` varchar(30) NOT NULL,
  PRIMARY KEY (`id_interesse`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8;
```

Una volta creato il database questo può essere collegato e interrogato dal server. Il collegamento al database avviene nella pagina php, nel quale basta creare un nuovo oggetto MySQLi passando come parametri: nome dell'host, user, password e il nome del database selezionato.

dbconfig.php

Gestione della connesione al DB sul server

```
<?php
$host="localhost:3306";
$user="root";
$pass="";
$db="who_is_around";
?>
```

page.php

```
include 'dbconfig.php';
$dbmng= new mysqli($host, $user, $pass, $db);

If($this->connection->connect_error){

   throw new RuntimeException(mysqli_connect_error(),
mysqli_connect_errno());

    }else   { …
```

Dopo aver estrapolato i dati questo vengono formattati in formato JSON e inviati al Client in modo che possa utilizzarli e analizzarli. Questo avviene grazie alla funzione json_encode($array), messa a disposizione da PHP, e all'opportuna formattazione dell'array risposta $response passato come parametro.

Formattazione risposta JSON

```
// keeping response header to json

    header('Content-Type: application/json');

// echoing json result

    echo json_encode($response);
```

3.4 - Funzionalita' App – Parte Background

3.4.1 - Trovare gli utenti

Al fine di trovare gli altri utenti registrati come abbiamo già detto utilizzeremo le API che mette a disposizione Android per l'utilizzo del Wi-Fi Direct o come definito da Google, Wi-Fi P2P (Peer-to-peer).

*"Using these APIs, you can **discover** and connect to other devices when each device supports Wi-Fi P2P"[18]*

Utilizzeremo quindi la procedura che mette a disposizione Android per il rilevamento e connessione tra dispositivi che supportino questo standard, senza però intraprendere una connessione peer-to-peer ma fermandoci alla fase di "*discover*".

GOOGLE API : WI-FI P2P

Il Wi-Fi P2P API sono costituiti dalle seguenti parti principali:

- I **metodi** che permettono di scoprire, richiedere, e di connettersi ai peers sono definiti nella classe *WifiP2pManager* [19].
- I **Listeners** che consentono di venire a conoscenza del successo o del fallimento delle chiamate ai metodi del WifiP2pManager. Quando vengono richiamati i metodi del WifiP2pManager, ogni metodo può avere un ascoltatore specifico, passato come parametro.
- e gl'**Intent** che notificano il rilevamento di eventi specifici dal framework Wi-Fi P2P, come ad esempio l'interruzione della connessione o la scoperta di un nuovo peer.

Queste componenti verranno utilizzate insieme.

Il WifiP2pManager fornisce i metodi per consentire di interagire con l'hardware Wi-Fi del dispositivo, per operazioni come scoprire e connettersi ai peer.

Utilizzeremo i suoi metodi:

Metodo	Descrizione
`initialize ()`	Registra l'applicazione con il framework Wi-Fi. Questo metodo deve essere chiamato prima di chiamare qualsiasi altro metodo P2P Wi-Fi.
`discoverPeers ()`	Avvia la ricerca peers
`requestPeers ()`	Chiede l'elenco corrente dei peer scoperti.

A questi metodi è consentito passare come parametro un Listener, in questo modo il framework Wi-Fi P2P può notificare lo stato della chiamata. Le interfacce listener disponibili e le corrispondenti chiamate ai metodi del WifiP2pManager sono descritte nella tabella seguente:

Interfaccia Listener	Azioni associate (utilizzate)
`WifiP2pManager.ActionListener`	`discoverPeers ()`
`WifiP2pManager.ChannelListener`	`initialize ()`
`WifiP2pManager.PeerListListener`	`requestPeers ()`

Le API Wi-Fi P2P definiscono inoltre gli Intent che vengono inviati in broadcast quando accadono determinati eventi associati al Wi-Fi P2P, come quando un nuovo peer viene scoperto o quando avvengono cambiamenti allo stato della connessione Wi-Fi di un dispositivo. È possibile registrare l’applicazione perché reagisca a questi eventi mediante la creazione di un BroadcastReceiver che gestisca questi Intent:

Intent	Descrizione
`WIFI_P2P_CONNECTION_CHANGED_ACTION`	Trasmesso quando lo stato della connessione Wi-Fi del dispositivo cambia.
`WIFI_P2P_PEERS_CHANGED_ACTION`	Trasmesso quando si chiama `discoverPeers ()` . Sarà seguito dalla chiamata al metodo `requestPeers ()` per ottenere un elenco aggiornato dei peers.
`WIFI_P2P_STATE_CHANGED_ACTION`	Trasmesso quando Wi-Fi P2P viene abilitato o disabilitato sul dispositivo.
`WIFI_P2P_THIS_DEVICE_CHANGED_ACTION`	Trasmesso quando cambiano i dettagli di un dispositivo, come per esempio il nome del dispositivo.

Un BroadcastReceiver permette di ricevere gli Intent trasmessi dal sistema Android, in modo che l'applicazione possa rispondere a eventi a cui si è interessati.

Per la creazione di un Broadcast Receiver per gestire intenti P2P Wi-Fi:

- Creare una classe che estenda quella BroadcastReceiver. Al costruttore della classe passiamo come parametri il WifiP2pManager, il WifiP2pManager.Channel e l'Activity (o Service) in cui questo receiver verrà registrato, in modo da permettere al BroadcastReceiver di inviare gli aggiornamenti all'Activity (nel nostro caso un Service) e consentendogli allo stesso tempo l'accesso all'hardware Wi-Fi e al canale di comunicazione.
- Implementare le reazioni che si desiderano per i rispettivi Intent nel metodo OnReceive () del BroadcastReceiver. Ad esempio, se si riceve un Intent WIFI_P2P_PEERS_CHANGED_ACTION, è possibile chiamare il metodo requestPeers() per ottenere un elenco dei pari attualmente scoperti.

3.4.2 - Implementazione di 'Trova gli Utenti'

Prima di utilizzare le API P2P Wi-Fi, è necessario assicurarsi che l'applicazione possa accedere all'hardware e che il dispositivo supporti il protocollo P2P Wi-Fi. Se il Wi-

Fi P2P è supportato, è possibile ottenere un'istanza di WifiP2pManager , creare e registrare il Broadcast Receiver, e iniziare a utilizzare le API del Wi-Fi P2P.

Impostiamo la richiesta di autorizzazione a utilizzare l'hardware Wi-Fi e anche la versione minima di Api richiesta nel Manifest dell'applicazione. (Per l'utilizzo del Wi-Fi Direct la versione minima di Sdk richiesta è la 14).

Creiamo le classi:

MyService che estende la classe Service, è quindi il servizio che si occuperà rimanendo attivo in background di prelevare i Mac Address dei dispositivi incontrati e inviarli al server tramite un processo parallelo, *HttpGetTask*.

WiFiDirectBroadcastReceiver che estende la classe Broadcast Receiver si occuperà quindi di intercettare e gestire gli Intent lanciati dal sistema al fine di redigere la lista di dispositivi incontrati e reagire ai cambiamenti di stato del dispositivo.

Dopo aver verificato che il Wi-Fi P2P sia acceso e supportato, un buon posto per farlo è nel Broadcast Receiver quando si riceve l'Intent WIFI_P2P_STATE_CHANGED_ACTION (cioè quando il Wi-Fi viene acceso o spento), notifichiamo al MyService lo stato del Wi-Fi P2P se è attivo faremo partire il meccanismo di ricerca altrimenti attenderemo che l'utente lo attivi.

```
@Override
public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();

        if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
            // Determina se la modalità Wifi P2P mode è abilitata o no
            int state = intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE, -1);
            if (state == WifiP2pManager.WIFI_P2P_STATE_ENABLED) {
                    // Abilitato
              }
```

WiFiDirectBroadcastReceiver.java

Nel metodo *onCreate()* del MyService, si ottiene un'istanza di WifiP2pManager e chiamato il metodo *initialize()* per registrare l'applicazione con il framework Wi-Fi P2P. Questo metodo restituisce un *WifiP2pManager.Channel* , che viene utilizzato per connettere l'applicazione al framework P2P Wi-Fi. È stato inoltre necessario istanziare un *WifiDirectBroadcastReceiver* passandogli gli oggetti *WifiP2pManager* e *WifiP2pManager.Channel* insieme al riferimento del nostro MyService. Questo permette al Receiver di comunicare con il Service.

È stato creato un Intent Filter impostato con gli stessi Intent che il WiFiDirectBroadcastReceiver deve intercettare, ovvero il cambiamento della lista dei dispositivi presenti nel nostro raggio (WIFI_P2P_PEERS_CHANGED_ACTION) e dello stato della connessione Wi-Fi (WIFI_P2P_STATE_CHANGED_ACTION).

MyService.java

```
@Override
      public void onCreate() {
         super.onCreate();

          //Indica un cambiamento nello stato del Wi-Fi P2P.
          intentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION);

          //Indica un cambiamento nella lista dei peers.
          intentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION);

          mManager = (WifiP2pManager) getSystemService(Context.WIFI_P2P_SERVICE);
          mChannel = mManager.initialize(this, getMainLooper(), null);
          receiver = new WiFiDirectBroadcastReceiver(mManager, mChannel, this);
          registerReceiver(receiver, intentFilter);

          ...

         mManager.discoverPeers(mChannel, new WifiP2pManager.ActionListener() {
                       @Override
                       public void onSuccess() { . . . }
                       @Override
                       public void onFailure(int reasonCode) { . . . }
         });
```

Per una miglior gestione delle risorse l'istanza del WiFiDirectBroadcastReceiver viene registrata nel metodo onResume () del MyService e deregistrata in quello onDestroy().

Una volta ottenuto un WifiP2pManager.Channel e impostato un Broadcast Receiver l'applicazione può effettuare chiamate di metodo Wi-Fi P2P e ricevere intenti Wi-Fi P2P e richiamare i metodi inWifiP2pManager.

```
public class WiFiDirectBroadcastReceiver extends BroadcastReceiver {


    private WifiP2pManager mManager;
    private Channel mChannel;
    private MyService mService;

       ...

    public WiFiDirectBroadcastReceiver(WifiP2pManager manager, Channel channel,
            MyService service) {
        super();

        this.mManager = manager;
        this.mChannel = channel;
        this.mService = service;
    }


       @Override
    public void onReceive(Context context, Intent intent) {

        String action = intent.getAction();

        ...

        } else if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {


             // Richiesta lista dei peers disponibili al wifi p2p manager.
            if (mManager != null) {
                mManager.requestPeers(mChannel, peerListListener);
            }

       ...
```

WiFiDirectBroadcastReceiver.java

Per rilevare i peer disponibili nel raggio d'azione del Wi-Fi Direct si effettua una chiamata al metodo discoverPeers (). La chiamata a questa funzione è asincrona e il successo o il fallimento viene comunicato all'applicazione attraverso i metodi onSuccess () e onFailure () del WifiP2pManager.ActionListener . Il metodo onSuccess () inoltre segnala solo che il processo di rilevamento è riuscito ma non fornisce alcuna informazione circa i peer che sono stati scoperti.

Quando il processo di rilevamento ha esito positivo e rileva dispositivi, il sistema trasmette un Intent WIFI_P2P_PEERS_CHANGED_ACTION, che viene intercettato grazie al nostro WiFiDirectBroadcastReceiver.

Solo a questo punto è possibile richiedere un elenco dei peer scoperti con requestPeers() che notifica quando una lista di peer è disponibile attraverso il metodo onPeersAvailable(), definito nell'interfaccia *WifiP2pManager.PeerListListener*, che fornisce una WifiP2pDeviceList dalla quale si estraggono i Mac Address dei dispositivi rilevati.

```
private PeerListListener peerListListener = new PeerListListener() {
    @Override
    public void onPeersAvailable(WifiP2pDeviceList peerList) {

        peers.clear();

        peers.addAll(peerList.getDeviceList());

  if (peers.size() == 0) {
                    //Non ho trovato dispositivi
                    return;
        }else{

             //salvo tutti i MACAddress dei dispositivi trovati
             for(WifiP2pDevice Device : peers){
                    if(!MacList.contains(Device.deviceAddress)){
                    //aggiungo i Mac Address dei peer alla lista MacList
                     MacList.add(Device.deviceAddress);

                    }
      ...
```

WiFiDirectBroadcastReceiver.java

In questo modo si ottiene una lista dei dispositivi con cui si è interagito all'interno del raggio di azione del Wi-Fi Direct. Per verificare se questi siano utenti registrati e la presenza o meno di interessi in comune il MyService preleva la lista e la invia al server servendosi di un AsyncTask.

3.4.3 - Collegamento con il Server

HttpGetTask

Una volta ottenuto la lista di MAC Address per verificare vi sono associati utenti sul server utilizziamo quindi un AsyncTask [20].
La classe AsyncTask definisce tre generics: AsyncTask<Params, Progress, Result>. I generics servono ad aumentare la "sicurezza in compilazione" ed a evitare ClassCastException runtime.

Questi parametri vengono utilizzati per i metodi principali di AsyncTask:

- onPreExecute: eseguito sul thread principale, contiene il codice di inizializzazione dell'interfaccia grafica (per esempio l'inabilitazione di un button);
- doInBackground: eseguito in un thread in background si occupa di eseguire il task vero e proprio. Accetta un parametro di tipo Params (il primo generic definito) e ritorna un oggetto di tipo Result;
- onPostExecute: eseguito nel thread principale e si occupa di aggiornare l'interfaccia dopo l'esecuzione per mostrare i dati scaricati o calcolati del task che vengono passati come parametro.

Per il progetto è stata creata una classe HttpGetTask che estende la classe AsyncTask e che accetta in ingresso parametri di tipo String.
Il task viene chiamato dal MyService se ci sono Mac Address nella lista del WiFiDirectBroadcastReceiver.

MyService.java

```
...
if(!receiver.MacList.isEmpty()){

	...

	int i=0;
	int disptrovati= receiver.MacList.size();
	String[] ArrayMac = new String[disptrovati];
	for(String mac : receiver.MacList){
		//Aggiugo i Mac address dalla lista del receiver.
		ArrayMac[i]=mac;
		i++;
	}
	receiver.MacList.clear();


	HttpGetTask task = new HttpGetTask(ArrayMac);
	task.execute();
}
...
```

Nel metodo doInBackground controlla che dei MAC Address alcuni non siano già presenti nei nostri contatti. Quelli che non lo sono vengono inseriti in un lista di coppie valore ed insieme all'URL della pagina di controllo affinità vengono passati all'oggetto JSONParser che li invia al Server insieme al Mac Address del dispositivo mittente.

Al server saranno inviate quindi le seguenti coppie valori: MyMac contenente il valore del Mac Address del dispositivo mittente, “qta” che contiene il numero di

```
public class HttpGetTask extends AsyncTask<String,String,String>  {

static final String CheckAffinityURL = "http://"+URLSERVER+"/pEServerTesi/check.php";
...

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

             //Gestione dell'invio della lista al Server
             JSONParser jsonParser = new JSONParser();
             List<NameValuePair> params2 = new ArrayList<NameValuePair>();
             params2.add(new BasicNameValuePair("qta",num));
             for(int j=0;j<qta;j++){

                    String ind= "MAC"+Integer.toString(j);
                    params2.add(new BasicNameValuePair(ind,MacdaInviare[j]));

             }

             params2.add(new BasicNameValuePair("MyMac",MyService.MyMacAdd));
             JSONObject json = jsonParser.getJSONFromUrl(CheckAffinityURL, params2);
```

dispositivi trovati e tante coppie (“MAC+i”, “valore MAC”) quante indicate dal parametro “qta”.

```
$interessiincomune = $dbmng->query("SELECT nomeinteresse FROM interessiutente where
id_u IN ($id_A,$id_B) GROUP BY nomeinteresse HAVING COUNT(nomeinteresse) > 1");

if($interessiincomune->num_rows > 0){
    $response["found"]=1;
     $result = $dbmng->query("SELECT * FROM utenti where MAC='$mac'");
     $interessiincomune->free();
     if($result->num_rows > 0){
     while($row = $result->fetch_assoc()){

       // array temporaneo per creare single contacts
       $tmp = array();
       $tmp["id_us"] = $row["id_utente"];
       $tmp["MAC"] = $row["MAC"];
       $tmp["name"] = $row["username"];
       $tmp["email"] = $row["email"];

       // inserise il contatto nel json array
       array_push($response["contacts"], $tmp);
      ...
    // keeping response header to json
    header('Content-Type: application/json');

    // trasforma la risposta in un json result
    echo json_encode($response);
```

check.php

La pagina *check. php* sul server si occupa quindi di collegarsi al database controllare se tra i MacAddress ricevuti ci siano o meno utenti registrati e che abbiano almeno un interesse in comune con l'utente associato al MacAddress mittente. E di inviare al

AsyncTask HttpGetTask

```
//Gestione della risposta Json del server

...

try {
      if (json.getString(KEY_SUCCESS) != null) {

            String res = json.getString(KEY_SUCCESS);

            if(Integer.parseInt(res) == 1){
                  // Trovato Qualcuno!

                  //prendo l'array Contatti  dal JSONdata
                  contacts = json.getJSONArray(TAG_CONTACTS);

              for (int i = 0; i < contacts.length(); i++) {
                  JSONObject json_user = contacts.getJSONObject(i);

                  // Se è un nuovo utente aggiungiamo nella tabella CONTACTS
                  DatabaseHandler db = new DatabaseHandler(getApplicationContext());
                  String UserMac= json_user.getString(KEY_MAC);
                  if(!db.isInDb(UserMac))
                  {
                     db.addUser(json_user.getInt(KEY_IDUS),
                     json_user.getString(KEY_MAC), json_user.getString(KEY_NAME),
                     json_user.getString(KEY_EMAIL));

                  }

                  if(db.notifymeeting(UserMac)){
                     quanti++;
                     //Aggiungere l'incontro nella tabella MEETINGS!!
                     db.addMeeting(json_user.getInt(KEY_IDUS),
                     json_user.getString(KEY_NAME),
                     json_user.getString(KEY_MAC), Latitude, Longitude);

                  }

                  if(quanti>0){
                        creanotifica(quanti);
                  }

                  ...
```

client (app) i contatti in formato JSON. La risposta viene gestita sempre all'interno del metodo *doInBackground*, il quale estrapola la risposta dall'oggetto JSON restituito dal parser e se sono stati trovati Utenti, aggiunge il relativo incontro nella tabella Meetings e il nuovo contatto nella tabella Contacts. Per i Mac Address già presenti nella lista contatti viene aggiunto solo il relativo incontro. Per evitare di

inviare al server MAC già incontrati oppure inviare continuamente lo stesso perché presente nel nostro raggio più di un minuto, vengono effettuati controlli incrociati sulla posizione e la tempistica di incontro.

Infatti la funzione *notifymeeting()* controlla che non vengano considerati, inseriti e quindi notificati gli incontri relativi allo stesso utente che avvengano nello stesso posto e nell'arco temporale di un ora.

La funzione *creanotifica()* invia una notifica al sistema contenente l'Intent con il compito di aprire l'applicazione nella MainActivity visualizzando la lista "NEW" con i nuovi Incontri effettuati.

Esempio di risposta di check.php

```
{
"found":1,

"contacts":[

       {"id_us":"6",
       "MAC":"aa:27:65:dc:b6:a8",
       "name":"Matteo54",
       "email":"Matteo@ro.it"},

       {"id_us":"3",
        "MAC":"bb:a7:22:61:6a:2e",
        "name":"Chiara12",
        "email":"Chiar@ch.it"}
       ]
}
```

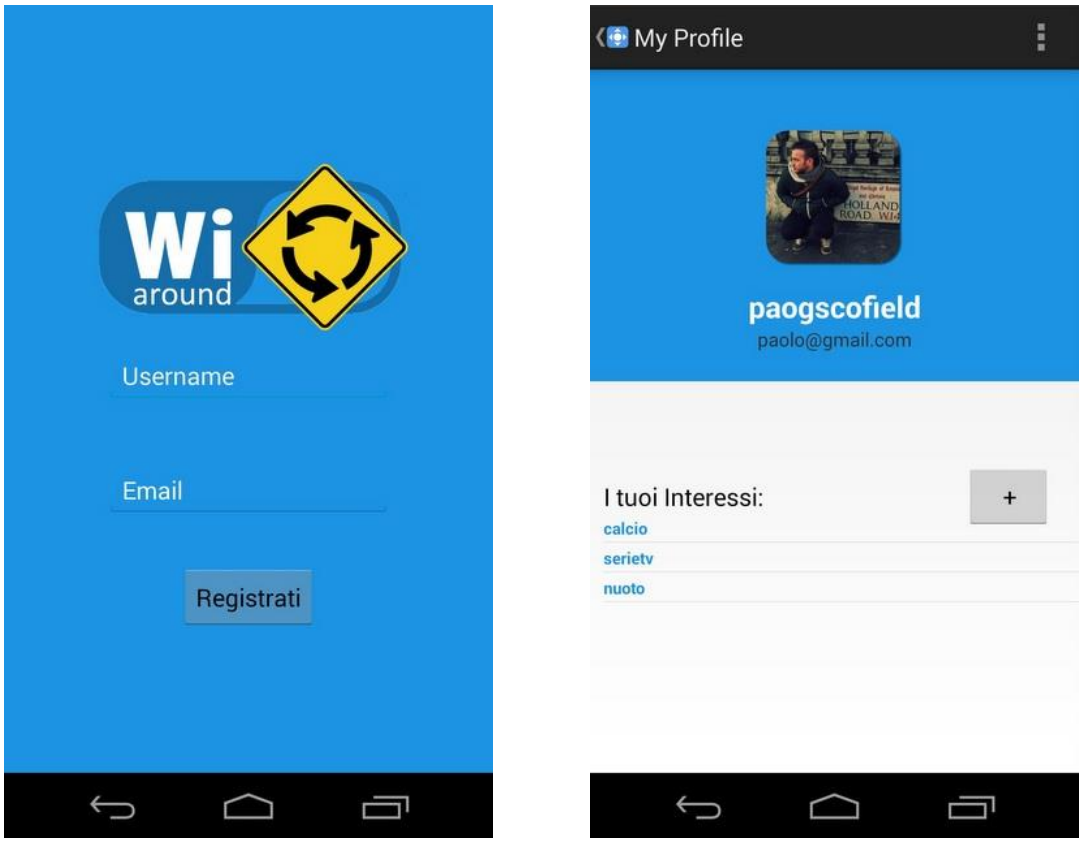
Il MyService viene avviato la prima volta dall'utente premendo utilizzando lo switch posto nell'ActionBar della MainActivity attraverso il metedo *startService(Intent i)*. Quando viene attivato il servizio servendosi dell'*AlarmManager* imposta un timer che ogni 60 secondi richiami il servizio, avviandone il metodo onStartCommand().

La pressione del tasto *off* dello switch richiamerà la funzione *stopService()* che distruggerà il service. Nell'implementazione del metodo *onDestroy()* del Service le impostazioni del timer vengono eliminate.

3.5 - Interazione Utente: Activities

3.5.1 - Registrazione e Gestione del Profilo

La prima installazione. Una volta che l'utente ha scaricato e installato l'applicazione la pagina che gli si mostrerà all'apertura è quella di registrazione una volta registrato all'apertura dell'app si avvierà la pagina relativa all'interazione con i propri contatti e con gli incontri passati. Questo grazie all'activity FirstActivity che effettua un controllo sul database interno. Se sono presenti dati dell'utente invia un Intent e visualizza l'Activity principale altrimenti quella di registrazione. Nel caso in cui l'utente sia già registrato sul server ma per qualsiasi motivo il database dell'applicazione non sia più disponibile sul dispositivo il proprio profilo potrà essere recuperato dalla pagina di registrazione.



Figg. n. 6,7 – Pagine di registrazione e del Profilo

Una volta registrato il proprio profilo con Username e Email, l'utente viene indirizzato verso la pagina di gestione del proprio profilo (MyProfile) dalla quale può visualizzare e modificare i propri dati e dove può aggiornare la lista dei propri interessi.

Per aggiungere un interesse alla lista basta effettuare un tap sul bottone '+' e un Intent ci farà visualizzare l'activity "New Interest" dove possiamo selezionare un interesse

dallo Spinner [21], riempito con gli interessi provenienti dal server, e aggiungerlo alla lista oppure creare un nuovo interesse se questo non dovesse essere già presente.

Per rimuoverne un Interesse basterà toccare sull'interesse presente in lista.

In fase di registrazione i dati sono inviati al server servendosi del JSONParser che invierà i dati alla pagina server *registrazione.php* che controlla che i dati siano conformi alle specifiche e invia la risposta al client (app).

Fig. n. 8 – Activity di modifica degli interessi

3.5.2 - Gestione e visualizzazione degli incontri e dei contatti

MainActivity

La MainActivity è la l'activity principale di navigazione. Al suo interno infatti sono caricati e visualizzati i Fragment relativi alle liste "NEW","ALL" e "CONTACTS" che rappresentano il fulcro dell'interattività dell'utente. Un Fragment rappresenta una funzione o una parte di interfaccia utente di una Activity [22]. E' possibile combinare più Fragment in una singola Activity così da realizzare una interfaccia utente multi-pannello e poter riutilizzare un Fragment in più Activity.

La MainActivity si compone quindi di 3 FragmentList visualizzati in un TabLayout, NewMeetings, AllMeetings, Contacts.

Nell'ActionBar della Main Activity inoltre sono presenti i collegamenti per Attivare/Stoppare il Servizio e per accedere al MyProfile.

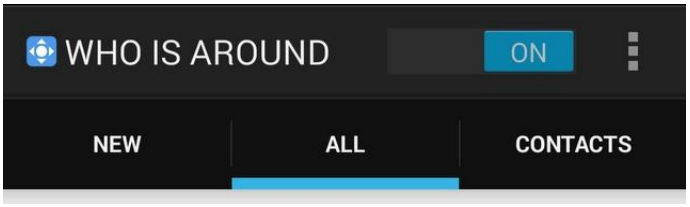


Fig. n. 9 – Action Bar

3.5.3 - Incontri

In caso di riscontro interessi positivo con un Utente incontrato, l'incontro (username, data e posizione) verrà salvato un record nella tabella incontri (TABLE_MEETINGS) nel DB interno all'applicazione.

La lista "ALL" non è altro che l'estrazione, da parte di un AsynTask di tutti i record degli incontri effettuati presenti nel DB interno, all'interno di una ListView.

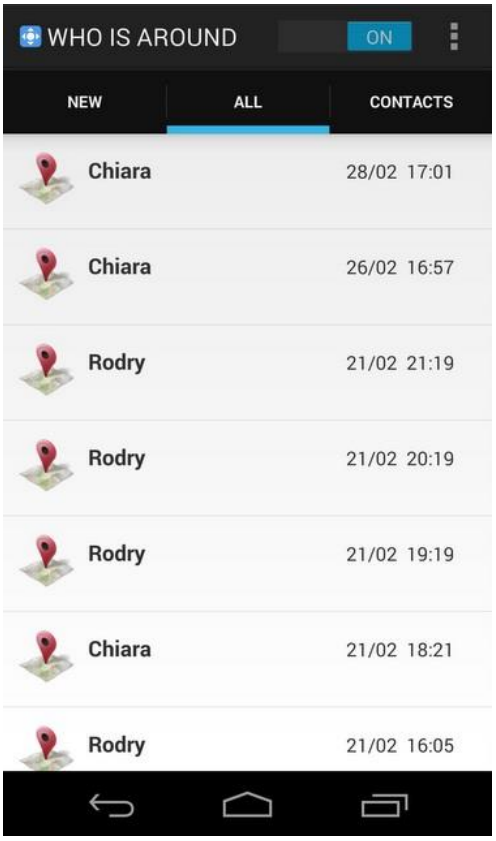


Fig. n. 10 – Lista Incontri

Per ogni elemento della ListView è presente un onClickListener. L'implementazione del suo metodo onClick permette poi di aprire la visualizzazione specifica per quell'incontro (Activity SingleMeeting).

Visualizzazione di un singolo incontro

Quando si apre l'activity Single Meeting si può vedere la posizione sulla mappa relativa a quell'incontro, l'username del contatto incontrato e le informazioni di data e ora.

Fig. n.11 – Singolo Incontro

GOOGLE MAPS API V2

Per visualizzare sulla mappa il luogo di un incontro si è fatto uso delle Google Maps Api V2 che mette a disposizione Android.

Il primo passo indispensabile consiste nell'includere nel progetto le API di Google attraverso pochi semplici passaggi. Una volta importate le mappe questa può essere richiamata e inserita in un MapFragment e richiamato all'interno dell'activity.

Un MapFragment è il modo più semplice per inserire una mappa in un'applicazione. Si tratta di un contenitore per la visualizzazione di una mappa, al fine di gestirne automaticamente le esigenze del ciclo di vita. Essendo un Fragment, questo componente può essere aggiunto al file di layout dell'activity semplicemente con l'XML sottostante [23].

```
<fragment
   class = "com.google.android.gms.maps.MapFragment"
   android:layout_width = "match_parent"
   android:layout_height = "match_parent" />
```

Per inserire il segnaposto relativo alla posizione dell'incontro basterà utilizzare le coordinate di Latitudine e Longitudine salvate al momento dell'incontro.

Per aggiungere un marker è necessario creare un oggetto MarkerOptions passandogli le coordinate e sarà sufficiente utilizzare la funzione messa a disposizione dalle API *addMarker()* per visualizzarlo sulla mappa.

Per aggiungere l'animazione che centri la mappa sul luogo dell'incontro si può facilmente utilizzare il metodo *animateCamera* passandogli l'oggetto CameraPosition istanziato con le coordinate di Latitudine e Longitudine relative.

Mentre per le informazioni relative al contatto e alla data vengono estratte dal database.

Single Meeting.java

```
// create marker
MarkerOptions marker = new MarkerOptions().
                        position(new LatLng(latitude, longitude)).title(name);
googleMap.addMarker(marker);

//
CameraPosition cameraPosition = new CameraPosition.Builder()
      .target(new LatLng(latitude,longitude)).zoom(15).build();

googleMap.animateCamera(CameraUpdateFactory.newCameraPosition(cameraPosition));
```

3.5.4 - Contatti

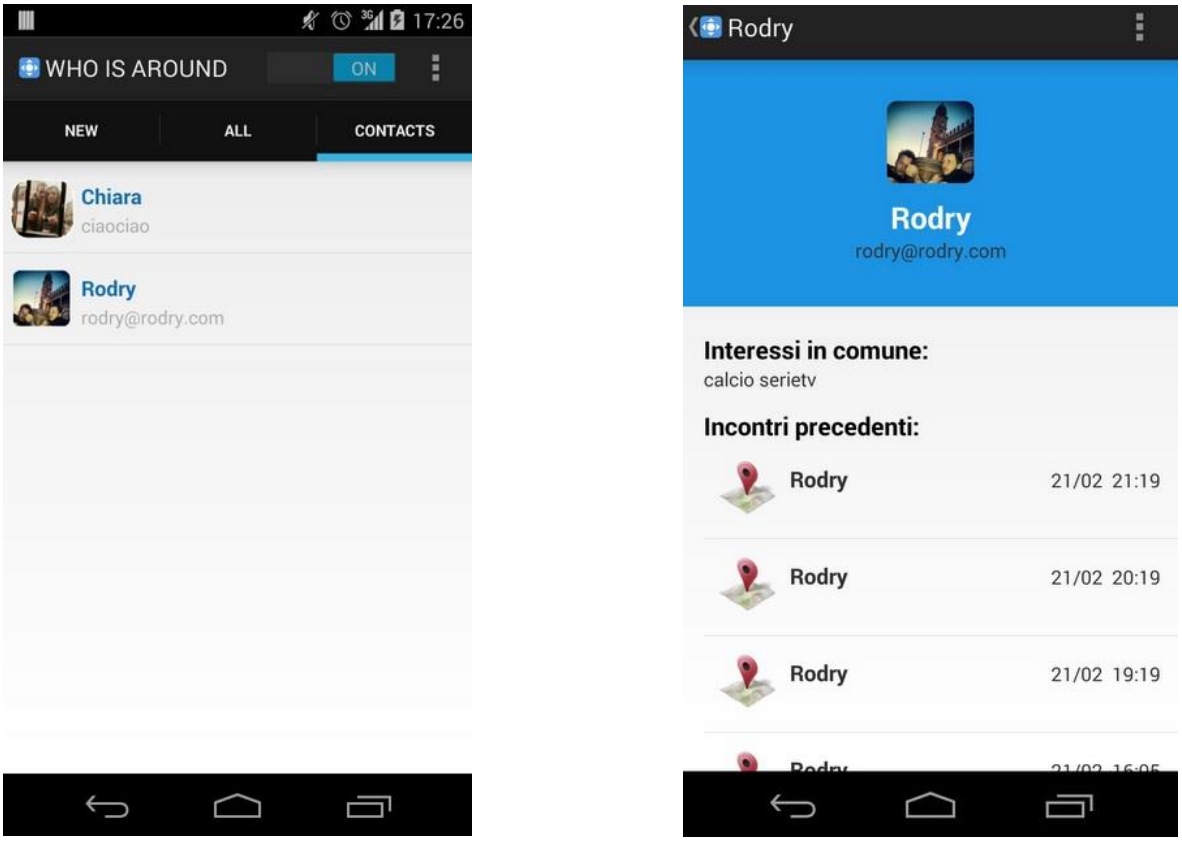
Quando incontriamo un utente che non abbiamo mai incontrato lo salviamo nei contatti. Il ListFragment Contacts non è altro che l'estrazione di tutti i contatti dalla tabella Contacts del DB interno in una ListView.

```
public List<Contact> getAllContacts() {
		List<Contact> contactList = new ArrayList<Contact>();
		// Select All Query
		String selectQuery = "SELECT  * FROM " + TABLE_USER +" ORDER BY "+
KEY_NAME;

		SQLiteDatabase db = this.getWritableDatabase();
		Cursor cursor = db.rawQuery(selectQuery, null);

		// looping through all rows and adding to list
		if (cursor.moveToFirst()) {
			do {
				Contact contact = new Contact();
				contact.setID(Integer.parseInt(cursor.getString(0)));
				contact.setIDUS(Integer.parseInt(cursor.getString(1)));
				contact.setMAC(cursor.getString(2));
				contact.setName(cursor.getString(3));
				contact.setEmail(cursor.getString(4));
				// Adding contact to list
				contactList.add(contact);
			} while (cursor.moveToNext());
		}

		// return contact list
		return contactList;
	}
```

DBHandler.java

Selezionando un Contatto verrà inviato un Intent che farà in modo che il sistema ci visualizzi l'Activity Single Contact la quale contiene il dettaglio di un contatto.

Visualizzazione di un singolo contatto



Figg. n.12,13 – Visualizzazione dei contatti (lista e singolo)

I dettagli relativi mostrano username, contatto email, gli interessi in comune e lo storico degli Incontri per l'utente. Facendo tap su in Incontro della lista è possibile visualizzarne i dettagli.

Nel Database interno però non sono salvate anche le informazione relative agli interessi in comune, quelle vengono prelevate dal Server quando viene richiesto il profilo di un singolo contatto. Questo avviene utilizzando l'oggetto JSONPARSER al quale vengono forniti l'identificatore dell'utente e l'URL della pagina server *profilo.php*, ricevendo come risposta l'oggetto JSON Contact contenente le informazioni relative agli interessi in comune con quell'utente.

```
{
"success":1,
"interests":["calcio","serietv"]
}
```

risposta di profilo.php

Capitolo 4. Conclusioni e Sviluppi Futuri

L'applicazione ha raggiunto gli obiettivi prefissati: permette di rilevare un contatto con altri dispositivi all'interno di un raggio limitato senza che richiedere l'interazione degli utenti. Per farlo è stato utilizzata una tecnologia in espansione quale il Wif-FiDirect che sfrutta un modulo presente su praticamente ogni dispositivo e che necessita solo di un aggiornamento software per poter essere utilizzato.

L'applicazione in sé è completa e funzionante ed è collegata ad un Serverweb raggiungibile. Sicuramente può essere migliorata: per esempio un integrazione con un profilo social eviterebbe all'utente di creare un nuovo profilo. Si potrebbe lavorare inoltre per ridurre il consumo della batteria e raffinare l'algoritmo che calcola l'affinità tra gli utenti, aumentando magari le categorie di input rendendo più completo il profilo o applicando un sistema di pesi agli interessi.

L'intenzione è stata comunque quella di sviluppare un meccanismo che applicasse e che permettesse questa interazione tra dispositivi. Tutta la complessità di questo sistema è delegata al server. I dispositivi hanno solo il compito di rintracciarsi, questo rende questa metodologia incredibilmente duttile e soprattutto a costo zero.

In questo progetto è stato applicato al fine di un interazione con altri utenti ma offre grandi possibilità. Una delle più produttive applicazioni di questo sarebbe, come accennato nell'introduzione, l'utilizzo per aumentare le possibilità di penetrazione delle attività commerciali.

L'integrazione con un profilo di cui sono noti gli interessi e generalità infatti permetterebbe non solo di effettuare una proposta di offerte più specifiche in termini di Proximity Marketing ma anche di coinvolgere nuovi clienti.Questo genere di approccio al marketing è solo un risultato della crescente integrazione e direzione che ha preso il mondo della tecnologia verso l'Internet of Things.

Apple ha investito molto in questo genere di approccio applicandolo però utilizzando il Bluetooth e in maniera un po' differente. Il prodotto di casa Apple si basa sulla tecnologia Bluetooth Low Energy introdotta con lo standard 4.0, che permette il trasferimento di dati elaborati tramite localizzazione GPS ottimizzando il consumo energetico del dispositivo, ma necessità dell'installazione di antenne che inviino al dispositivo i dati di cui necessita, che hanno un costo.

Lo svantaggio procurato dal dover disporre di altri dispositivi per dover funzionare e di un raggio d'azione più ridotto si produce però in un vantaggio quando si parla di

localizzazione, infatti questa modalità consente di stabilire con precisione la posizione dei dispositivi intercettati.

Sono due diversi tipi di approccio con i propri vantaggi e svantaggi quello che rimane una certezza è l'aumento costante dei dispositivi mobili in circolazione. Le nuove tecnologie applicate al proximity marketing stanno rivoluzionando il concetto di acquisto all'interno dei negozi e di integrazione tra smart-object e persone nella vita quotidiana e questo è sicuramente un settore in cui investire.

Bibliografia

[i] Statistiche sviluppo internet e social Italia: http://wearesocial.it

[1] Dati Principali Store Mobile : http://www.ipsos.com/mediact/

[2]Statistiche Mondo Android e Mobile, googleourplanet: http://www.thinkwithgoogle.com/mobileplanet/it/

[3] Statistiche relative alle diffusioni dei sistemi operative e smartphone: http://idcitalia.com/ita/

[4]iOS vs Android: ecco la situazione attuale sulla frammentazione dei sistemi operativi http://www.dimt.it/2014/02/13/smartphone-android-e-ios-occupano-il-957-del-mercato/

[5] iOS: http://www.ispazio.net/453817/ios-vs-android-ecco-la-situazione-attuale-sulla-frammentazione-dei-sistemi-operativi ,

[6]WindosPhone Situazione attuale http://www.tomshw.it/cont/news/applicazioni-android-anche-su-windows-8-e-windows-phone/53349/1.html

[7] Android Italy: “Cos’è Android? La storia del sistema operativo mobile di Google” - http://www.androiditaly.com/articoli/speciali/189-cose-android-la-storia-el-sistema-operativo-mobile-di-google.html,

[8, 9] NFC, Bluetooth,WifiDirect: http://www.phonegurureviews.com/bluetooth-nfc-wifi-direct/ , http://mobileworldcapital.com/en/article/239 , http://www.nfcitaliaworld.it/

[10] iBeacon: https://www.passbeemedia.com/ibeacon/, http://www.ridble.com/ibeacon/, http://support.apple.com/kb/HT6048

[11]Wi-Fi Direct: http://www.wi-fi.org/discover-wi-fi/wi-fi-direct

[12] Wi-Fi Peer-to-Peer API: http://developer.android.com/guide/topics/connectivity/wifip2p.html

[13]Componenti Architettura Android: http://developer.android.com/guide/components/index.html

[14] Introducing JSON - http://www.json.org , http://www.json.org/xml.html

[15] Developer Android: package org.json - http://developer.android.com/ reference/org/json/package-summary.html

[16] Developer Android: SQLiteOpenHelper – http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html

[18] [19] Developer Android APIs: http://developer.android.com/guide/topics/connectivity/wifip2p.html

[20] AsyncTask: http://developer.android.com/training/articles/perf-anr.html

[21] Spinner: http://developer.android.com/guide/topics/ui/controls/spinner.html

- Grant Allen, “Beginnig of Android 4” - Ravaioli Mirko, Dispense del corso di Mobile Web Design,