ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI SCIENZE

CORSO DI LAUREA IN SCIENZE E TECNOLOGIE INFORMATICHE

STUDIO E IMPLEMENTAZIONE DI COMPONENTI SOFTWARE PER PIATTAFORMA MOBILE ANDROID

Tesi di Laurea in
MOBILE WEB DESIGN

Relatore:
Dott. Mirko Ravaioli

Presentata da:
Francesco Cecchi

Sessione III
Anno Accademico 2012/2013

Indice

Introduzione

Il mercato degli smartphone è un ambiente che ha visto una crescita rigogliosa negli ultimi 15 anni, passando da dispositivi rudimentali e dalle capacità decisamente limitate a tecnologie man mano più evolute ed in grado di servire a scopi ed utilizzi sempre più complessi, raffinati ed onerosi.

Abbiamo conosciuto dispositivi legati alla telefonia, i cellulari, grazie ai quali era possibile memorizzare numeri di telefono in una rubrica, inviare e ricevere SMS ed effettuare chiamate, e abbiamo anche incontrato altri dispositivi che invece assumevano più il ruolo di "agenda elettronica", come i PDA(*Personal Data Assistant*).

Entrambi sono dispositivi che hanno riscosso un notevole successo a partire dalla seconda metà degli anni novanta, e che da sempre hanno percorso strade destinate a incrociarsi.

Incominciava a sentirsi la necessità di fare tutto su un unico gadget, e verso la fine dello scorso millennio ci furono le prime timide scoperte nel mondo degli smartphone: i produttori di dispositivi iniziarono ad assemblare cellulari con una serie di componenti hardware per arrivare ad offrire servizi nuovi, che invogliassero l'utenza ad aggiornare i propri dispositivi.

Non si tardò molto, quindi, ad ottenere dei device con funzionalità quali registratore audio, fotocamera, riproduttore musicale, rubrica, accesso ad Internet, lettura di e-mail e localizzazione GPS. In tempi più recenti è inoltre stata aggiunta un'altra importante novità: il touch screen.

Per potere gestire tutte queste funzionalità vi era l'esigenza di un sistema operativo da installare sugli smartphone e i software utilizzabili dall'utenza hanno contribuito nella crescita e diffusione degli stessi ambienti mobile per cui tali software erano costruiti. Nel corso del tempo abbiamo visto nascere e crescere una serie di ambienti mobile, ognuno caratterizzato da pregi e difetti.

Questa tesi è volta allo studio del mondo di Android, e si svilupperà nell'ottica dello studio, analisi, implementazione ed esposizione di alcuni dei concetti e delle tecnologie che più si sono rivelate interessanti.

Nel capitolo 1 mi occuperò di fare un veloce riepilogo dei maggiori ambienti mobile presenti oggigiorno. I Capitoli successivi torneranno invece a concentrarsi sul mondo di Android.

Il capitolo 2 tratterà dell'utilizzo delle mappe, introducendo lo sviluppo della tecnologie che le sorregge, per poi spiegare come sfruttarle. Nella parte finale del capitolo non mi occuperò più di come accede agli strumenti di sviluppo, ma mostrerò come è effettivamente possibile interagire con gli elementi del servizio, creando un'app dimostrativa.

Il capitolo 3 sarà orientato allo studio dei servizi di push-notification, anche qui esponendo l'argomento in una breve introduzione. Si proseguirà con l'esporre i metodi per approcciarsi a questa tecnologia ed infine, anche qui verrà introdotta ed esposta un'app, con l'ausilio di codice dimostrativo.

Infine il capitolo 4 sarà dedicato all'analisi ed esposizione della gestione dei sensori, illustrandone differenze ed esponendo quali tipi di sensore sono supportati, per poi finire il capitolo con una veloce guida su come potere interagire con essi.

1. Ambienti Mobile

Nonostante la tesi si concentrerà sullo studio del mondo Android, è doveroso riepilogare la storia e le caratteristiche che sono dietro ai principali sistemi operativi per piattaforme mobile che possiamo trovare oggigiorno.

1.1 Sistema iOS

Sistema operativo della casa di Cupertino, iOS fu presentato per la prima volta nel gennaio 2007 assieme all'iPhone presso il *MacWorld Conference and Expo* di San Francisco, sotto il nome di "iPhoneOS".

Con un passo annuale, questo sistema operativo ha goduto di costanti e importanti aggiornamenti, che l'hanno fatto crescere in termini di stabilità, affidabilità e potenzialità. Ogni update ha portato delle novità, alcune anche suggerite dagli stessi clienti, come per esempio il supporto per MMS e funzionalità di copia-incolla. Nella versione 2.0 venne introdotto l'App Store, tramite cui gli utenti possono installare app prodotte da terzi.

Nel 2010 con la versione iOS 4.0 vengono introdotte funzionalità di multitasking, come per esempio la geolocalizzazione e notifiche push, introdotte da Apple in quanto non si voleva rischiare che app di terzi consumassero troppa batteria nel fare queste procedure. Fu da questa versione che si incomincio a riferirsi al sistema operativo come "iOS", abbandonando iPhone OS.

Successivi aggiornamenti hanno portato man mano novità, fra cui il supporto per la sincronizzazione wireless, l'integrazione con il servizio iCloud, l'assistente vocale Siri e un rinnovamento della gestione delle mappe.

Secondo i più recenti controlli, iOS si posiziona al secondo posto dei sistemi operativi più distribuiti, con un 23% di share del mercato.

1.2 Sistema BlackBerry

BlackBerry OS è il sistema operativo mobile proprietario, sviluppato da RIM (*Research In Motion*) per la linea di prodotti BlackBerry. La prima versione risale al 1999 e lo rende in un certo senso il primo sistema operativo per smartphone. Basato su Java, il mondo BlackBerry è noto per le sue capacità di supporto della posta elettronica aziendale attraverso MIDP (*Mobile Information Device Profile*), che offre funzionalità di attivazione della rete wireless e sincronizzazione di dati quali e-mail, calendario, attività, note e contatti attraverso software quali Lotus Domino e Microsoft Exchange.

I dispositivi BlackBerry furono un'innovazione nell'ambito aziendale: oltre alle funzionalità appena descritte, i BlackBerry inoltre disponevano di una picoola tastiera **QWERTY** che, palesemente migliore per la scrittura di e-mail, venne molto apprezzata dall'utenza. Questi dispositivi inoltre presentano un supporto per trackball, trackpad e touchscreen.

Date le caratteristiche di queste funzionalità e servizi, si può con sicurezza affermare che BlackBerry si è fatto strada nel mondo aziendale, portando le attività e le industrie ad introdurre uno stile di lavoro più serrato grazie alle possibilità comunicative offerte dai dispositivi.

1.3 Sistema Android

Nel 2003, Android Inc. fu fondata da Andy Rubin, Rich Miner, Nick Sears e Chris White. Si vociferava che questa società, che all'apparenza affermava di occuparsi semplicemente di applicazioni mobile, si stesse muovendo per la creazione di un nuovo sistema operativo basato su kernel di Linux. Google intanto stava incominciando a guardarsi attorno, fiutando che nell'aria c'era odore di cambiamento, e si rendeva conto che la sua presenza nel mondo mobile non era adeguata, o comunque, adeguatamente protetta. Nel 2005 quindi, il colosso acquisì Android Inc. e portò avanti il progetto. Nel

2007, assieme ad altre società quali Quallcom, Samsung, HTC e T-Mobile, Google fonda l'OHA(*Open Handset Alliance*), progetto che mira allo sviluppo di dispositivi che presentano un sistema operativo libero. Nello stesso anno viene rilasciata la prima versione dell'SDK, che permette ai developer di iniziare a creare app per rendere appetibile il nuovo sistema operativo.

Come avviene per Linux, ogni versione del sistema operativo segue un ordine alfabetico e dalla versione 1.5 riporta il nome di un dolce: CupCake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Beans ed infine KitKat. Le versioni che si possono considerare "alfa" e "beta", sono in stretto legame col nome del sistema operativo, cioè sono Astro Boy e Bender. La versione 1.1 di Android invece fu chiamata Petit Four.

Oramai alla versione 4.4, Android come altri sistemi operativi è cresciuto, aggiungendo man mano sempre più funzionalità, correggendo bug, e apportando modifiche anche sostanziali. Presenta nativamente un supporto per i servizi di Google ai quali ogni dispositivo si può collegare, supporta widget, account Exchange per la sincronizzazione, e presenta il PlayStore per poter acquistare app, musica e libri tramite il device.

In quanto software open e distribuibile sui dispositivi di più produttori di smartphone, non c'è da stupirsi che il mercato di Android abbia riscosso sempre più successo, terminando il 2013 con il 79% di share del mercato.

2. Mappe

2.1 Introduzione

Le mappe nel mondo di Android sono state introdotte nel 2009 con il rilascio sul dispositivo Motorola Droid del sistema operativo Android 2.0 Eclair.

In questa versione infatti fu rilasciata l'app Google Maps Navigation, da molti considerata una "rivoluzione", in quanto presentava già dalla prima release version delle funzionalità assai interessanti per tutta la possibile utenza, per esempio:

- **Supporto della lingua inglese**, cioè non era necessario dover conoscere l'indirizzo esatto del luogo da cercare.
- **Supporto vocale in lingua inglese.**
- **Due diversi tipi di visualizzazione delle mappe** : satellitare e stradale. Inoltre era anche presente una visualizzazione del traffico in base alla rotta scelta.
- **Modalità alla guida** : attivando questa modalità, eventi come ricezione di chiamate, SMS e semplici sveglie venivano gestite in modo più semplificato, intuitivo e maneggevole.
- **Ricerca di luoghi di interesse sulla rotta**
- **Salvataggio della rotta** : una volta selezionata una rotta, non è più necessaria mantenere attiva la connessione Internet, in quanto il percorso viene memorizzato.

Per non dimenticarsi del fatto che l'app **NON** era a pagamento, garantendo già da subito una posizione favorevole rispetto ad altre app come TomTom, Navigon, MapQuest, ecc.

Con queste premesse, non c'è da stupirsi che Google Maps Navigation sia presto diventata una cosiddetta "app-killer", cioé una app così efficiente da mettere a tappeto la concorrenza.

Google ha quindi proseguito nel tempo ad apportare modifiche, migliorie e il supporto per sempre più lingue.

2.2 Uso delle mappe

Assieme a questa potente app, sono state rilasciate agli sviluppatori delle API(*Application Programming Interface*) per permettere di interfacciarsi con i servizi offerti da Google.

Queste API per Android sono gratuite, a meno che l'app che sfrutti la tecnologia di Google faccia una richiesta superiore ai 25'000 "aggiornamenti" al giorno. In tal caso è necessario passare alla versione "for Business", a pagamento, che permette di superare questo tetto massimo di richieste.

2.2.1 Attivazione del servizio Google

La versione 1 di Google Maps Android API aveva un certo meccanismo di "registrazione", attraverso il quale era possibile far riconoscere il proprio applicativo sui server di Google, e quindi sfruttarne le potenzialità.

Questa versione è oramai deprecata (*3 dicembre 2012*) e dal *18 marzo 2013* non è più possibile registrare nuove app, ma quelle che già erano iscritte ai servizi possono continuare a sfruttare la versione 1 senza problemi.

La versione 2 mantiene lo stesso concetto per la registrazione:

1) Ottenere una chiave SHA-1 dall'app
2) Registrare il progetto su Google APIs Console
3) Inserire nell'app il codice di registrazione

Passo 1:

Per ottenere questa chiave, bisogna eseguire il comando keytool sul file keystore. Il file keystore può essere di due tipi: debug o release.

Viene generato sempre un file debug.keystore quando si procede alla compilazione del codice sorgente durante la fase di sviluppo, mentre la versione release fa parte del processo di firma dell'app, di solito uno degli ultimi passi prima della distribuzione del prodotto.

Da linea di comando quindi, eseguiamo l'operazione, in questo caso sul file debug.keystore:

```
C:\Users\Scr47ch\.android>keytool -list -v -keystore debug.keystore -alias androiddebugkey
-storepass android -keypass android
```

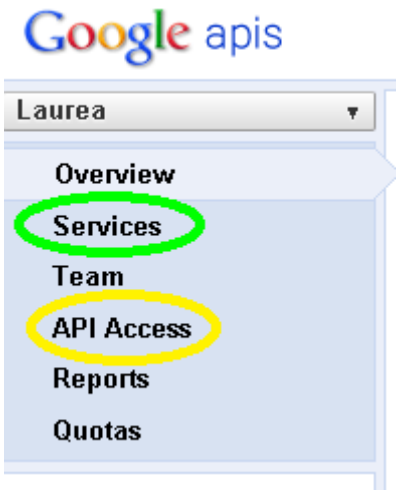
L'esecuzione di questo comando ci restituirà una serie di informazioni, fra le quali, anche la SHA-1 key:

```
SHA1: 2D:67:B3:19:E9:F0:16:6C:67:79:2A:DB:54:01:14:A0:21:54:97:46
```

Passo 2:

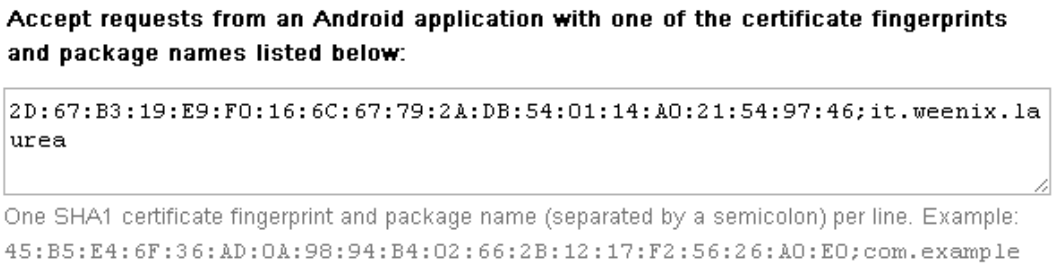
Procediamo ora con la registrazione del progetto sul Google APIs Console, un'interfaccia web che ci permette di personalizzare e gestire le nostre app, decidendo a quali servizi di Google può accedere.

Per prima cosa bisogna fare login sul sito, clickare su **Create Project** e seguire le indicazioni per creare quindi un nuovo progetto associato al nostro account.



 Selezionato il nostro progetto dal menu a tendina, procediamo e clickiamo la voce Services, cerchiamo la voce Google Maps Android API v2 e attiviamola.

 Successivamente torniamo al menu principale e selezioniamo la voce API Access. Da qui selezioniamo la voce **Create new Android key...** e inseriamo la nostra SHA-1 key e il package name della nostra app, separate da un punto e virgola



Una volta confermato, otterremo una chiave di 40 caratteri sotto la voce

Key for Android apps (with certificates)

API key: AIzaSyC4ENCf0p2nLhVq1fB-3mec3YRb5StQVM4

Ora che l'app è registrata, non ci resta che collegare l'app ai servizi mediante questo codice "API key".

Passo 3:

Questa fase dell'attivazione delle mappe riguarda la modifica del file *AndroidManifest.xml* all'interno del nostro progetto.

Per prima cosa aggiungiamo all'interno del tag <*application*>

```
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="API_KEY"/>
```

Sostituendo al posto della voce API_KEY dell'attributo *android:value*,il codice di registrazione ottenuto in precedenza.

A questo punto abbiamo effettivamente stabilito un ponte fra noi e Google, ma la nostra app ancora non può funzionare a dovere senza darle i permessi necessari.

Inseriamo quindi come figlio diretto del tag <*manifest*> una serie di voci definite come:

```
<uses-permission android:name="nome_permesso"/>
```

I permessi minimi per far funzionare l'app sono i seguenti quattro:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVIC
ES"/>
```

- **Accesso ad Internet**, per scaricare le mappe.
- **Accesso allo stato della connettività,** per controllare se una connessione Internet è possibile.
- **Possibilità di scrivere sulla memoria esterna**, necessario per il salvataggio della cache
- **Accedere ai servizi di Google.**

I seguenti due permessi non sono necessari, ma di solito sono usati in app di geolocalizzazione:

```
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Il primo permesso offre all'app la possibilità di ottenere delle coordinate geografiche sulla posizione attuale sfruttando le connessioni WiFi e traffico mobile. Si chiama COARSE in quanto è solo approssimativa, dato che, a differenza del permesso FINE, non viene calcolata utilizzando la tecnologia GPS (*Global Positioning System)*, che offre una precisione maggiore.

Infine, potrebbe rivelarsi utile se non necessario avvisare che la propria app stia sfruttando una tecnologia che necessita di opportune librerie grafiche.

Google Maps Android API v2 infatti ha bisogno di OpenGL ES v2 installate sul dispositivo per riuscire a renderizzare la mappa.

Per avvisare di questa necessità, si può aggiungere accanto ai permessi la seguente voce:

```
<uses-feature
        android:glEsVersion="0x00020000"
        android:required="true"/>
```

Non bisogna però dimenticarsi di una cosa molto importante: bisogna "firmare" il file *AndroidManifest.xml* con la versione dei Google Play Services, l'app di Android che fa effettivamente da ponte fra noi e Google.

Dentro il tag <*application*> aggiungiamo

```
<meta-data
    android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version" />
```

In questo modo si saprà con quale versione di Google Play Services l'app è stata compilata.

Il processo di registrazione e attivazione delle mappe si può ore definire concluso. Abbiamo ora a disposizione tutti gli strumenti per sfruttare le API delle mappe.

Vediamo ora come aggiungere una mappa alla nostra app:

le mappe vengono gestite in Android come Fragment, ossia un "pezzo di interfaccia utente" che può essere inserito nell'Activity (la schermata dell'app).

2.3 Implementazione

Vediamo ora una possibile implementazione delle mappe. Di seguito verrà esposto il codice quantomeno essenziale per utilizzare effettivamente le API di questo servizio.

Verrà mostrato come ottenere una schermata che presenta una mappa e in seguito alcuni dei comandi base da utilizzare.

L'implementazione delle mappe avviene sfruttando due classi:

La classe che definisce come ci sarà presentata la mappa, cioè se la mappa sarà una ***view*** o un ***fragment***. Avremo quindi o un riferimento alla classe *MapView* o uno a *MapFragment*.

La classe che si occupa dell'interazione con la mappa, *GoogleMap*.

Di seguito si è preferito gestire l'app con una struttura basata su fragments, quindi utilizzeremo *MapFragment*. Da far notare, le differenze fra le due classi non si trovano a livelo "espositivo" della mappa, ma a livello di implementazione. Come per ogni fragment, la classe *MapFragment* oltre ad avere il proprio life-cycle sarà anche legata al ciclo di vita dell'activity nella quale viene presentata, mentre la classe *MapView* è da considerare come un'activity, col proprio life-cycle indipendente. Si tratta solo di una scelta di programmazione quindi.

L'implementazione si estende sul fragment e l'activity che lo ospita, cioè i file *MainActivity.java* e *MyMapFragment.java*.

Partiamo dall'activity:

activity_main.xml

```
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:tools="http://schemas.android.com/tools"

    android:id="@+id/container"

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    tools:context="it.weenix.laurea.MainActivity"

    tools:ignore="MergeRootFrame" />
```

Ho deciso di utilizzare un FrameLayout come base per la creazione della grafica dell'activity piuttosto che istanziare staticamente il fragment.

MainActivity.java

```
public class MainActivity extends FragmentActivity {


    @Override

    public void onCreate(Bundle savedInstanceState)

    {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);

        if (savedInstanceState == null)

        {

                Fragment Map = new MyMapFragment();

        getFragmentManager().beginTransaction().add(R.id.container,
Map,"majorMapFragment").commit();

        }

    }

…

}
```

Alla creazione dell'activity viene fatto un controllo sull'oggetto Bundle *savedInstanceState*. Si controlla se il suo valore è equivalente a ***null***, e, se il controllo risulta positivo, si procede con la creazione del fragment.

Questo comportamento viene preso in quanto si vuole gestire un'app che possa cambiare orientamento quando il dispositivo viene girato. Se arbitrariamente si decidesse di creare ed istanziare il fragment senza un controllo di questo genere, ogni volta che il dispositivo cambia orientamento si aggiungerebbe un altro fragment a

quello già esistente. L'activity infatti viene ricreata ogni volta che vi è un evento di "rotazione", ma i dati all'interno di essa vengono preservati, assieme al fragment. Alla ricreazione dell'activity quindi verrebbe istanziato il fragment già esistente e aggiunto uno nuovo dello stesso tipo;*savedInstanceState* ci permette di capire quando un'app è avviata per la prima volta semplicemente controllandone il valore, che è ***null***.

La creazione del fragment avviene prima creando l'oggetto Fragment partendo dal frammento definito dalla classe *MyMapFragment*. Successivamente si ottiene il FragmentManager, si inizia una transazione, si aggiunge a quest'ultima l'oggetto Fragment e si conferma la transazione con *commit()* .

Passiamo ora all'esposizione della mappa vera e propria, partendo dal file XML che definisce il layout del fragment.

map_fragment.xml

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:tools="http://schemas.android.com/tools"

    android:layout_width="match_parent"

    android:layout_height="match_parent" >


    <fragment

        android:id="@+id/map"

        android:layout_width="match_parent"

        android:layout_height="match_parent"

        android:name="com.google.android.gms.maps.MapFragment" />


</RelativeLayout>
```

Notiamo che all'interno del fragment istanziato dalla MainActivity troviamo un altro fragment istanziato staticamente, cioè il fragment della mappa stessa. La dichiarazione viene fatta tramite un tag *<fragment />* generico e solo tramite l'attributo *"android:name="com.google.android.gms.maps.MapFragment />"* avviene un collegamento con il fragment che vogliamo creare. Notiamo anche l'attributo id, "*android:id="@+id/map"*, che ci servirà successivamente.

Definiamo ora alcune variabili ed oggetti che ci serviranno e il metodo *onCreateView* :

```
public class MyMapFragment extends Fragment {


        private View view;

        private Context activityContext;

        private ViewGroup container;

    private GoogleMap map;

        private final static String TAG = "Laurea";

    @Override

    public View onCreateView(LayoutInflater inflater, final ViewGroup
container, Bundle savedInstanceState)

    {

        view = inflater.inflate(R.layout.map_fragment, null, false);

        activityContext = container.getContext();

        this.container = container;

        Fragment thisFrag = (Fragment)
getFragmentManager().findFragmentByTag("majorMapFragment");

        thisFrag.setHasOptionsMenu(true);

        Bundle myInitBundle = getArguments();

        initializeMap();

       …
```

```
    }

  …

}
```

Porgiamo attenzione a due cose, l'oggetto *GoogleMap map*, e il metodo *onCreateView* : l'oggetto è quello definito precedentemente, cioè l'elemento con cui interagiremo per apportare modifiche alla mappa; il metodo invece mostra come istanziare il frammento e "disegnarlo" (tramite *inflater.inflate(R.layout.map_fragment, null, false)*).

Alla creazione del frammento viene inoltre richiamato un altro metodo, cioè *initializeMa*p() *:*

```
        private void initializeMap() {

        if (map == null) {

                map = ((MapFragment)
getFragmentManager().findFragmentById(R.id.map)).getMap();

            if (map == null) {

                Toast.makeText(activityContext, "Errore nella
creazione della mappa", Toast.LENGTH_SHORT).show();

            }

        }

        }
```

Se l'oggetto *GoogleMap map* non esiste già, viene ottenuto ricercando il fragment *MapFragment* all'interno del layout(definito in *map_fragment.xml*), e su questo oggetto *MapFragment* viene subito richiamato il metodo *getMap()*.

Abbiamo così ottenuto un'activity al cui interno viene visualizzata una mappa. Purtroppo allo stato attuale non siamo in grado di compiere alcuna operazione interessante, se non zoomare e spostarci nella mappa.

Vediamo innanzitutto come è possibile avere successo in un'operazione semplicissima: il cambio della mappa.

Al momento sono disponibili quattro diverse mappe da utilizzare:

- Normale
- Ibrida
- Satellitare
- Terrestre

Vi è però da far notare che la classe GoogleMap presenta anche un quinto valore da utilizzare per impostare la mappa, *GoogleMap.NONE*, cioè assenza di mappa.

Per potere cambiare la mappa basta richiamare l'oggetto GoogleMap, istanziato in precedenza, e richiamare il metodo *setMapType*. Il metodo richiede in input un valore integer, che può essere reperito tramite le variabili *public static int* definite nella classe GoogleMap.

```
map.setMapType(GoogleMap.MAP_TYPE_NORMAL);

map.setMapType(GoogleMap.MAP_TYPE_HYBRID);

map.setMapType(GoogleMap.MAP_TYPE_SATELLITE);

map.setMapType(GoogleMap.MAP_TYPE_TERRAIN);

map.setMapType(GoogleMap.MAP_TYPE_NONE);
```

Un comune metodo per la gestione di questi valori, è l'uso del widget Spinner, ossia un menù a tendina, che può essere popolato dinamicamente o staticamente tramite lettura di un file di risorse *XML*.

Passiamo ora ad esporre un semplice comando:

```
map.setBuildingsEnabled(true);
```

Sempre utilizzando l'oggetto GoogleMap, invochiamo il metodo *setBuildingsEnabled* e passiamogli una variabile booleana. Tramite questo metodo si può decidere se far apparire la creazione di solidi 3D sulle mappe. Questi solidi vengono visualizzati quando si fa uno zoom di precisione su alcune città maggiori, come per esempio Milano, Roma, Berlino, New York … Si può decidere quindi se mantenere una visuale "piatta" o meno.

Il prossimo passo è vedere come interagire letteralmente **sulla** mappa.

Per fare questo abbiamo bisogno di definire un listener della classe *OnMapClickListener*. Questo permetterà all'app di capire quando viene effettuata una pressione sulla mappa. Il listener può essere creato staticamente referenziandolo tramite una istanza di oggetto, oppure definire che la nostra Activity implementa il comportamento del listener in questione. Il metodo più comune rimane comunque l'implementazione dinamica.

```
map.setOnMapClickListener(new OnMapClickListener() {

                        @Override

                        public void onMapClick(LatLng point) {

                                Toast.makeText(activityContext, "Lat:
"+point.latitude+"\nLon: "+point.longitude,
Toast.LENGTH_SHORT).show();

                        }

};
```

Richiamando il metodo *setOnMapClickListener* impostiamo il listener, che dovrà presentare come minimo una implementazione del proprio metodo *onMapClick*, invocato automaticamente alla ricezione dell'evento di pressione. In questo esempio ogni qualvolta viene effettuato una pressione, verranno visualizzate le coordinate tramite un Toast.

Così come si può gestire l'evento di click semplice, si può fare lo stesso anche quando si effettua una pressione prolungata sulla mappa. Il metodo che dovremo usare sarà *setOnMapLongClickListener*, e stavolta dovremo istanziare un nuovo oggetto della classe *OnMapLongClickListener* :

```
map.setOnMapLongClickListener(new OnMapLongClickListener() {


                        @Override

                        public void onMapLongClick(LatLng arg0) {

                                MarkerOptions markerCreator = new
MarkerOptions().position(arg0).title("Marker di
prova").infoWindowAnchor((float) 0, 0);

                                Marker marker =
map.addMarker(markerCreator);

                                markers.add(markerCreator);

                                m.add(marker);

                        }

                });
```

In questo caso vediamo come un "long click" permetta di creare un *marker* sulla mappa, ossia un segnalino

L'oggetto *Marker* vero e proprio, cioè quell'icona che ci viene mostrata, non presenta un metodo costruttore. Bensì si parte dalla classe *MarkerOptions*, una classe che gestisce le impostazioni di configurazione del futuro Marker. Una volta finita la dichiarazione di *MarkerOptions*, l'oggetto viene utilizzato come argomento del metodo

add di GoogleMap. Usando questo metodo, la mappa legge le impostazioni di costruzione del Marker e lo disegna, restituendo come risultato l'oggetto Marker.

Si è da sempre capita l'importanza dei marker nell'ambito delle mappe, e quindi dentro la classe GoogleMap non manca certamente la gestione per il click su questi segnalini.

Similmente per come avviene per i click e i long-click, l'evento di pressione su marker deve essere letto da un listener, registrato dall'oggetto GoogleMap. Abbiamo quindi il metodo *setOnMarkerClickListener* e la classe *OnMarkerClickListener*.

```
map.setOnMarkerClickListener(new OnMarkerClickListener() {


                        @Override

                        public boolean onMarkerClick(Marker marker) {

                                LatLng punto = marker.getPosition();

                                Toast.makeText(activityContext, "Il
titolo di questo Marker è \'"+marker.getTitle()+"\' e mi trovo alle
coordinate ( "+punto.latitude+" ; "+punto.longitude" ).
",Toast.LENGTH_SHORT).show();

                });
```

L'oggetto Marker contiene una serie di valori interessanti da controllare, legati sia alle informazioni che restituisce(titolo e snippet, una descrizione aggiuntiva), sia legate all'oggetto in sé(rotazione, id, trasparenza, icona).

Similmente a come vengono creati i Marker, troviamo altri elementi di "decoro", per esempio *Circle* e *Polyline*, ossia linea e cerchio.

Avremo quindi delle classi "builder", *CircleOptions* e *PolylineOptions* rispettivamente, che andranno a definire le caratteristiche degli oggetti da disegnare. Gli oggetti derivati da queste classi, contenente una serie di impostazioni, andranno aggiunti alla mappa tramite metodi della classe GoogleMap.

Vediamo come utilizzare la classe *Circle*.

Come abbiamo detto, serve istanziare un oggetto *CircleOptions*. Le proprietà che definiscono un cerchio sono la posizione del centro, il raggio, il colore della figura, raggio e colore del contorno e l'altezza a cui è posizionato l'oggetto sulla mappa. Mentre questi valori vengono impostati semplicemente attraverso gli attributi di *CircleOptions*, servono dei metodi per modificare un oggetto *Circle* già disegnato.

Per aggiungere il cerchio alla mappa, si utilizza il metodo *addCircle* della classe *GoogleMap*.

```
CircleOptions myCircleOptions = new CircleOptions();

myCircleOptions.center(point);

myCircleOptions.fillColor(Color.argb(100,255,69,0));

myCircleOptions.radius(1000);

myCircleOptions.strokeColor(Color.TRANSPARENT);

Circle myCircleObject = map.addCircle(myCircleOptions);
```

Per *Polyline* avremo *GoogleMap.addPolyline*, mentre per *Circle* sarà *GoogleMap.addCircle*.

PolylineOptions richiede, come valori minimi per potere costruire la *Polyline*, delle coordinate geografiche, sotto la forma di un oggetto *LatLng*. In generale, non vi è limite a quanti "punti geografici" possono essere aggiunti per definire una *Polyline*.

Tramite *PolylineOptions* è inoltre possibile impostare colore e spessore della linea risultante prima che essa venga disegnata, ma anche successivamente sarà possibile intervenire sull'oggetto *Polyline* per modificarne gli attributi a piacimento.

```
PolylineOptions myPolyOptions = new PolylineOptions();

myPolyOptions.add(new LatLng(100.0, -30.5));
```

```
myPolyOptions.add(new LatLng(105.5, -40.0));

myPolyOptions.add(new LatLng(94.5, -40.0));

myPolyOptions.add(new LatLng(100.0, -30.5));

myPolyOptions.width = 10;

Polyline myPolyline = map.addPolyline(myPolyOptions);

myPolyline.setWidth(4);
```

Nel caso volessimo alterare la linea già tracciata, vi sono due opzioni:

1) Rimuovere l'oggetto dalla mappa, tramite il metodo *remove* della classe *Polyline*, e successivamente ricrearne uno nuovo.
2) Utilizzare il metodo *setPoints* di *Polyline*, che permette di ridisegnare la linea.

La classe *Polygon* invece si tratta di un particolare tipo di *Polyline*. La differenza fra le due si trova nel fatto che *Polygon* presuppone che i punti che formano l'oggetto creino una figura chiusa. L'esempio dimostrato prima della *Polyline* (che come si può notare dalla prima e ultima coppia di coordinate, definisce una figura chiusa), può essere quindi riadattato come segue:

```
PolygonOptions myPolygonOpt = new PolygonOptions();

myPolygonOpt.add(      new LatLng(100.0, -30.5),

                       new LatLng(105.5, -40.0),

                       new LatLng(94.5 , -40.0));

Polygon myPolygon = map.addPolygon(myPolygonOpt);
```

Anche *Polygon* può essere modificato tramite il suo metodo *setPoints()*.

Infine vediamo come lavorare e compiere azioni non appena la mappa stesa viene disegnata :

```
        map.setOnMapLoadedCallback(new OnMapLoadedCallback() {


                        @Override

                        public void onMapLoaded() {

                                CameraPosition cameraPosition = new
CameraPosition.Builder().target(new LatLng(44.2666 ,
12.3437)).zoom(12).build();


        map.animateCamera(CameraUpdateFactory.newCameraPosition(cameraP
osition));

                        }

                });
```

Tramite questa semplice operazione, si può dire all'app cosa fare quando la mappa viene caricata. In questo caso si decide di fare uno zoom alle coordinate 44.2666 , 12.3437 con uno zoom che vale 12.

Lo zoom della camera può assumere un valore compreso fra *GoogleMap.getMinZoomLevel()* e *GoogleMap.getMaxZoomLevel()*. Ad uno zoom di 0, il planisfero ha una larghezza di 256 dp. Ogni aumento di livello di zoom raddoppia la larghezza precedente. Quindi a livello 1 il planisfero sarà largo 512 dp per esempio.

3. Google Cloud Messaging

3.1 Introduzione

Un'altra potente tecnologia offerta da Google è Google Cloud Messaging, (*GCM)*, annunciata il 27 giugno 2012 presso il Google I/O tenuto a San Francisco.

In realtà si tratta di una tecnologia che sostituisce il vecchio servizio C2DM (*Cloud To Device Messaging*), in fase beta, rilasciato assieme ad Android 2.2 Froyo il 20 maggio 2010.

Stiamo parlando di push-notification, ossia un sistema tramite il quale è possibile ricevere messaggi da un server.

Il concetto è semplice, e l'applicabilità è vasta:

basti pensare di avere un'app orientata nell'ambito del giornalismo, che ci informa sulle notizie più recenti, o un'app per il meteo, oppure un'app che ci dica quando un negozio che ci piace presenta delle offerte, o quando un locale organizza un evento, con relativi aggiornamenti.

Tutti questi esempi presentano le medesime possibilità, potenzialità e gestibilità.

3.2 Problemi e soluzioni

Un sistema normale non potrebbe fare altro che interrogare il server ogni determinato periodo di tempo e scoprire se ci sono nuove informazioni da scaricare e presentare all'utente.

Il problema di questa tecnica, definita “polling” (cioè una richiesta continua di aggiornamenti, di solito basati su un timer) è che è dispendiosa, sia in termini di connettività sia in termini di batteria.

Un altro problema da non sottovalutare è che le richieste fatte dai client non è detto che restituiscano effettivamente informazioni utili!

Il servizio di GCM risolve i nostri problemi, in quanto non sarà più l’utente a richiedere informazioni, ma sarà il server a informare i client quando vi è necessità.

Una delle peculiarità di GCM è che sfruttando questo servizio, il server non si deve preoccupare che l’utente sia online o comunque raggiungibile nell’immediatezza dell’invio delle informazioni, in quanto i messaggi non saranno spediti attraverso un canale diretto con il device, ma verranno depositati presso server intermedi di Google che si preoccuperanno di farli reperire ai destinatari quando saranno raggiungibili. Google può constatare questo stato di reperibilità in quanto tramite Google Play Services riesce a mantenere sempre aperta e attiva una connessione con i propri device.

3.3 Analisi della metodologia di progetto

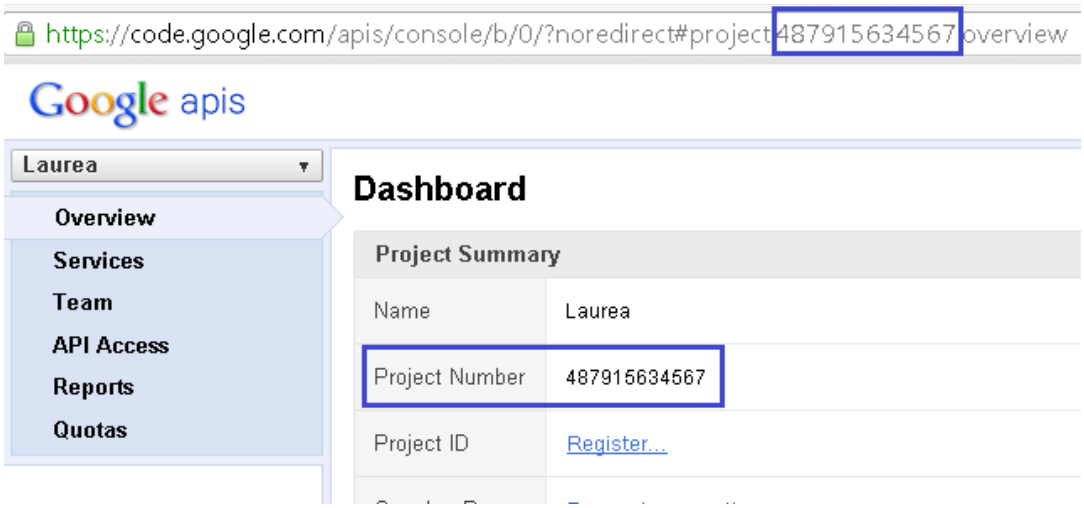
Esistono varie modalità di implementazione per l’architettura della comunicazione lato server. Per esempio si può scegliere se sfruttare un server CCS (*Cloud Connection Server*) che sfrutta dei messaggi XMPP (*Extentensible Messaging and Presence Protocol)*, un insieme di protocolli aperti basati su XML, oppure un server HTTP, che implementa dei messaggi o tramite testo normale o tramite formato JSON.

Ognuna di queste implementazioni ha i suoi pregi e i suoi difetti. Nell’esposizione che segue prenderemo l’esempio di avere a che fare con un server HTTP con messaggi JSON.

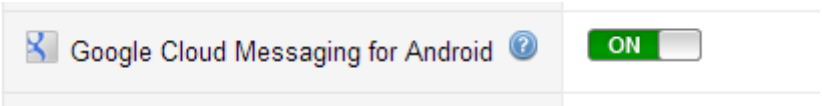
3.3.1 Registrazione

Per sfruttare le push-notification basta registrare sia il server che i client presso i server di Google.

Dopo aver fatto login su Google APIs Console, creiamo un nuovo progetto. Fatto questo, annotiamoci un importante valore che ci servirà nelle fasi successive dello sviluppo, cioè il Project Number (anche detto Sender ID). Questo è un valore numerico, che può essere trovato o nella voce apposita della home del progetto, oppure nel link stesso della pagina del progetto.



Si procede nella sezione Services del menu. Cerchiamo e attiviamo dunque la voce Google Cloud Messaging for Android.



A questo punto spostiamoci sulla voce API Access e clickiamo su **Create new Server key…** . Successivamente ci verrà chiesto di inserire l'IP del server, in modo da aggiungerlo ad una whitelist di IP permessi. Possiamo anche non inserire niente e dare conferma, in modo da permettere a tutti gli IP di essere accettati.

Sotto la voce :

Key for server apps (with IP locking)

API key:

troveremo il codice che ci serve.

Per esempio, :

Key for server apps (with IP locking)

API key: AIzaSyC2FoaePnCfMG4KKMvf7uBI-OB7lGAxOec

Questo sarà il riferimento del progetto per il server, ma analizzeremo questo aspetto più avanti.

3.3.2 Modifica dell'app

Il comportamento "passivo" dei client non significa che l'unica cosa da fare per i device sia installare l'app perché tutto il sistema funzioni. E' comunque necessaria una fase di inizializzazione.

Per prima cosa bisogna aggiungere le librerie di Google Play Services al nostro workspace.

Modifichiamo ora il file *AndroidManifest.xml* per potere aggiungere una serie di permessi e definire l'oggetto BroadcastReceiver che utilizzeremo.

I permessi minimi richiesti sono

- `com.google.android.c2dm.permission.RECEIVE` , in modo che l'app possa registrarsi e ricevere messaggi.
- `android.permission.INTERNET` , per permettere l'utilizzo di Internet
- `android.permission.GET_ACCOUNTS` , per device con una versione di Android inferiore alla 4.0.4, per permettere l'accesso all'account Google richiesto da GCM .
- `applicationPackage + ".permission.C2D_MESSAGE"`, quindi per esempio " com.weenix.laurea.permission.C2D_MESSAGE ". Serve a preservare la nostra app rispetto all'iter di registrazione e comunicazione con GCM, evitando che altre app possano intromettersi.
- `android.permission.WAKE_LOCK` , che da la possibilità all'app di mantenere il processore attivo. Si tratta di una permission opzionale, in quanto non è detto che tutte le app abbiano bisogno di mantenere il processore attivo. Di solito può risultare necessaria se alla ricezione dei messaggi si devono fare calcoli ed operazioni onerose, che potrebbero essere interrotte prima della loro fine.

Può risultare necessario l'inserimento nel Manifest del API level minimo dell'app. Sapendo che originariamente C2DM è stato rilasciato con Froyo, API level 8, avremo questa voce:

```
android:minSdkVersion="8"
```

Ora definiamo l'oggetto BroadcastReceiver.

Dentro il tag <*application*>, inseriamo

```
<receiver
   android:name="it.weenix.laurea.gcm.GcmBroadcastReceiver"
   android:permission="com.google.android.c2dm.permission.SEND">
   <intent-filter>
    <action android:name="com.google.android.c2dm.intent.RECEIVE"/>
    <category android:name="com.weenix.laurea" />
   </intent-filter>
</receiver>
```

Notare che nel campo android:name del tag <category /> deve essere inserito il packageName della nostra app.

Nel caso che ci si voglia approcciare allo sviluppo con i WakeLock accennati in precedenza, per evitare quindi che il processore si rimetta a riposo, il BroadcastReceiver che andremo a definire sarà in realtà un *WakefulBroadcastReceiver*, cioè un'estensione della classe normale che presenta dei metodi particolari. In più, sarà necessario registrare un Service nel Manifest:

```
<service
   android:name="it.weenix.laurea.gcm.GCMIntentService" />
```

Questo Service, che sarà istanziato come estensione della classe IntentService, sarà l'oggetto che eseguirà i lavori durante il periodo di WakeLock. Una volta eseguiti i lavori, rilascierà il lock, e il dispositivo potrà tornare a "riposo".

L'app, sfruttando le API di Google Play Services, esegue una serie di controlli:

primo fra tutti controlla se riesce a reperire in locale un registration_id . Se non ce la fa, sfrutta una serie di API per ottenere dai server GCM un determinato registration_id col quale può essere identificato il dispositivo stesso. Per fare questo viene mandato ai server GCM il Project Number/Sender ID di cui avevamo preso nota in precedenza.Questo codice viene salvato da qualche parte in locale, di solito fra le SharedPreferences.

E' raccomandabile inoltre salvarsi allo stesso modo la versione attuale dell'app: in avvii futuri si dovrebbe sempre controllare se la versione memorizzata coincide con la versione attuale, e nel caso differisca, richiedere un nuovo aggiornamento del registration_id (e salvarlo in locale assieme all'attuale app Version) .

Infine bisogna notificare il nostro server dell'esistenza della registrazione del nostro client su GCM : infatti al momento, l'unica cosa che il server conosce è l'id del progetto su GCM, ma quel valore non identifica nessun device client !

Anche tramite un semplice messaggio HTTP il registration_id viene spedito al server, che provvederà a salvarlo in maniera opportuna.

3.3.3 Funzionamento dell'app

Il BroadcastReceiver che abbiamo dichiarato nel Manifest ha il compito di restare in attesa ed interpretare i messaggi diretti all'app.

Ogni qualvolta il BroadcastReceiver riceverà un messaggio, il contenuto dello stesso viene passato come un oggetto Intent al metodo onReceive(Context, Intent), cioè il metodo che viene richiamato di default alla ricezione di messaggi.

Si può estrarre il contenuto del messaggio dall'Intent sfruttando le API appropriate.

Per esempio, se il messaggio dal server HTTP JSON contiene una chiave-valore come :

```
"messaggio" : "sono un messaggio di prova"
```

potremmo avere una situazione come

```
public void onReceive(Context context, Intent intent) {
        Bundle extra = intent.getExtras();

        String myString = "";
        if (extra.getString("messaggio") != null )

        {

                myString = extra.getString("messaggio");

        }
    }
```

In questo modo abbiamo salvato dentro una variabile myString, il contenuto del messaggio ricevuto.

3.3.4 Funzionamento lato Server

La comunicazione lato server a questo punto è facilmente descrivibile come segue :

viene creato un messaggio, con payload non superiore a 4kb, da spedire ai server GCM HTTP tramite una POST al link

`https://android.googleapis.com/gcm/send`

I messaggi spediti dal server dovranno presentare una certa formattazione, cioè bisognerà rispettare un protocollo, in modo che GCM e l'app lato client sappiano riconoscere e leggere le istruzioni che gli vengono passate.

Un messaggio HTTP è composto in due parti: header e body.

L'header deve contenere i seguenti due tag: Content-Type e Authorization.

- **Content-Type:** indica il tipo del contenuto del messaggio.
  - **JSON:** application/json
  - Testo Normale : application/x-www-form-urlencoded;charset=UTF-8
- **Authorization:** indica il codice del progetto GCM cui fare riferimento, cioè la Server Key ottenuta in precedenza. Essa viene inserito dopo "key=", per esempio come:
  Authorization: key=AIzaSyC2FoaePnCfMG4KKMvf7uBI-OB7lGAxOec

Il body invece contiene una serie di informazioni destinate al client, ma come messaggio minimo deve presentare una voce con chiave `registration_ids` e come valore il `registration_id` ricevuto dal client in fase di registrazione.

```
{ "registration_ids": [ "42" ] }
```

Da notare che `registration_ids` è un JSONArray, in quanto lo stesso messaggio può essere spedito a più di un destinatario.

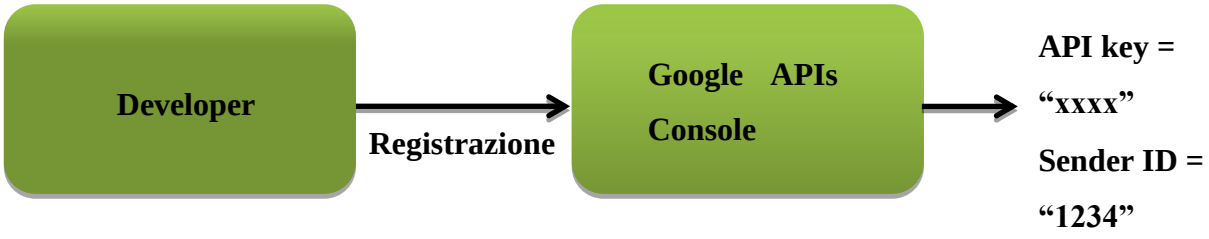
Il campo `data` definisce invece il contenuto effettivo del messaggio. Anche questo è un JSONArray, in quanto potrebbe essere necessario spedire più informazioni nello stesso messaggio.

Bisogna notare che ciò che riceverà il client non conterrà le voci `registration_ids` , `data` o altro. Questi campi vengono letti da GCM, e GCM inoltrerà il contenuto di `data` presso i device identificati da `registration_ids`

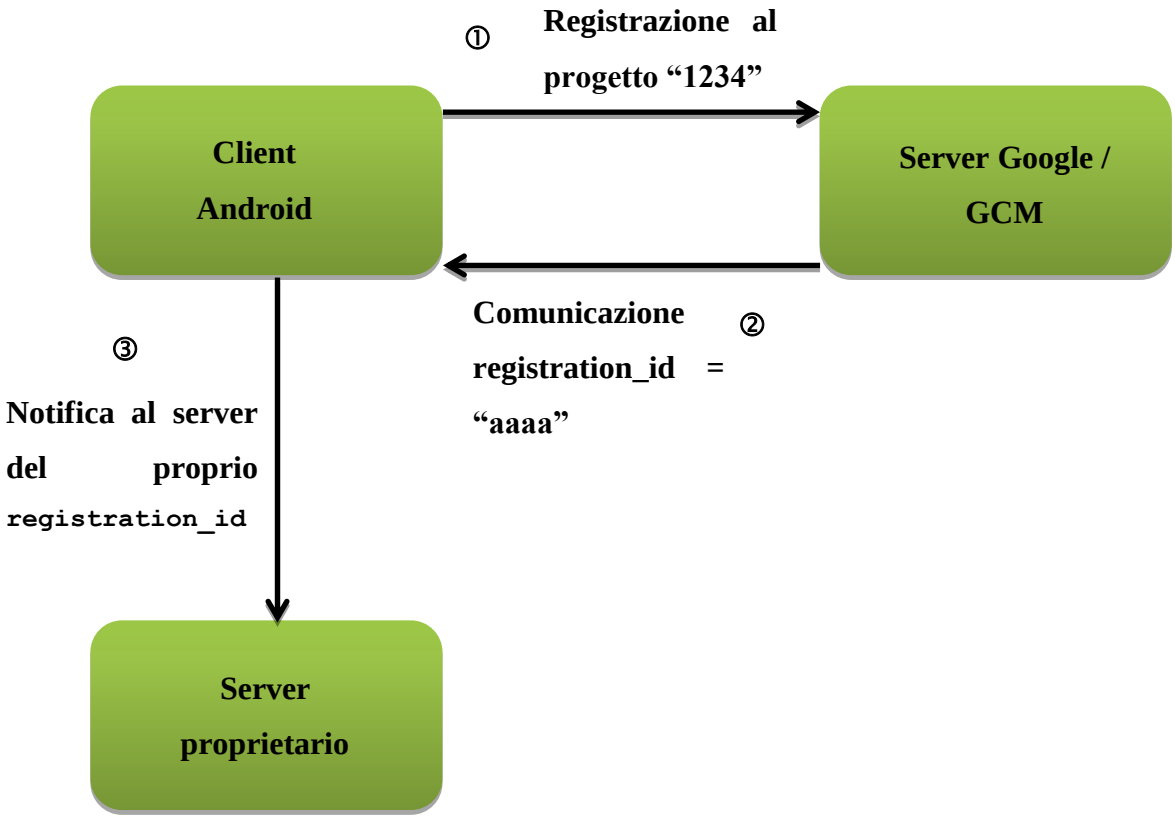
Ricapitolando:

Fase di registrazione :

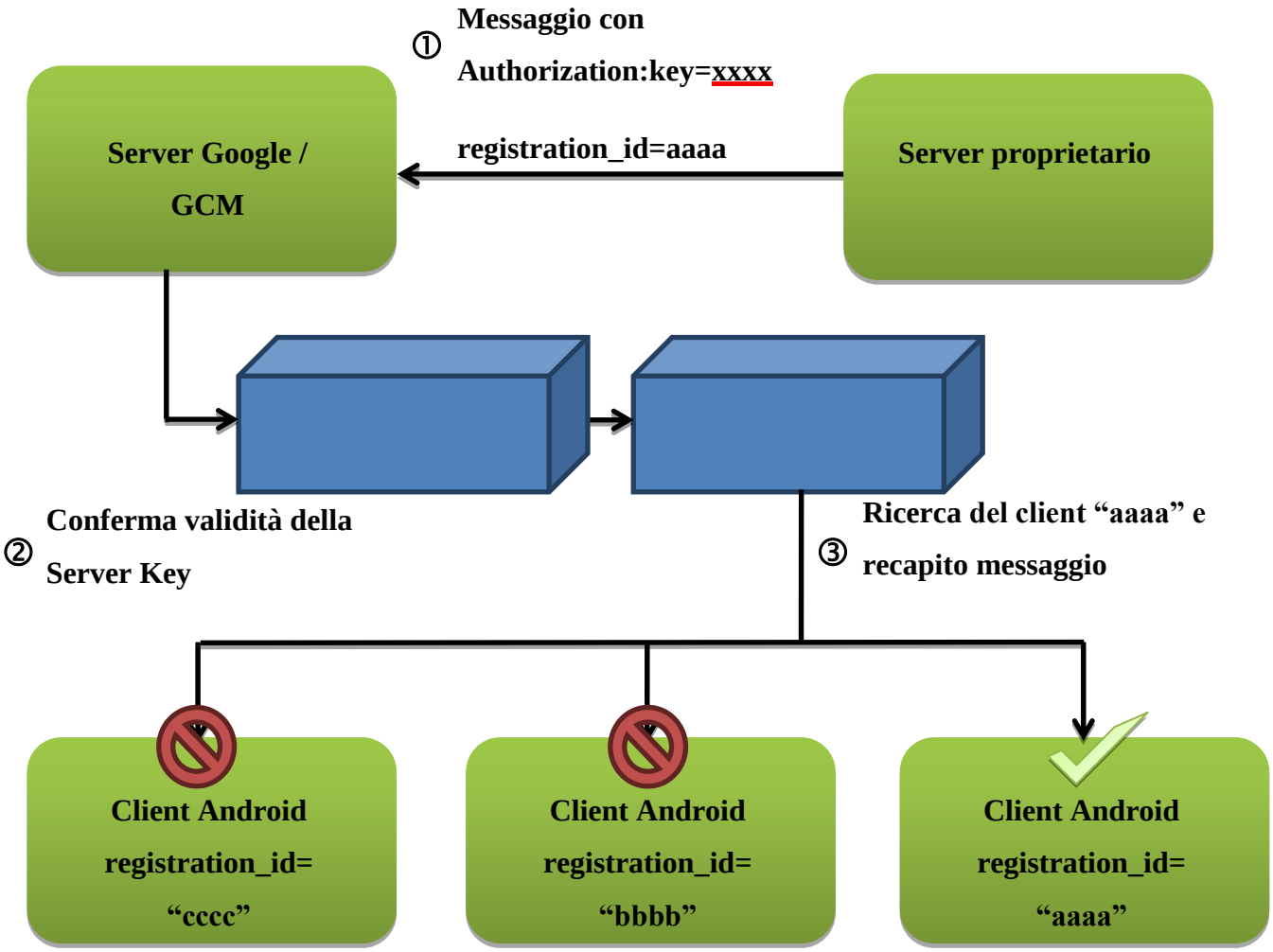
Passo 1)



Passo 2)



La fase di comunicazione invece sarà:



3.4 Implementazione

Mostriamo ora una implementazione reale della tecnologia appena esposta. Nello sviluppo di questa, mi sono preoccupato di gestire sia la parte Server che la parte Android del progetto.

Per quanto riguarda la parte Server ho creato un progetto in PHP supportato da un serverweb Apache. Ho installato e configurato XAMPP, una piattaforma software gratuita costituita da vari componenti, fra cui Apache HTTP Server , il database MySQL ed una serie di strumenti per la programmazione di linguaggi PHP e Perl. Da questa descrizione si può evincere il perchè del nome :

X Cross-Platform

A Apache HTTP Server

M MySQL

P PHP

P Perl

3.4.1 Linguaggio PHP

PHP, acronimo di "*PHP: Hypertext Preprocessor*", (originariamente *Personal Home Page*) è un flessibile linguaggio di programmazione che trova posto come strumento per lo sviluppo di soluzioni web.

Attualmente è infatti usato maggiormente per la creazione di pagine web dinamiche, ma la sua applicabilità si estende anche alla creazione di software con interfaccia grafica e per creare script utilizzabili da riga di comando.

PHP trova i suoi punti di forza nell'essere un linguaggio ***weakly typed***, ossia non vengono fatti controlli sui tipi di variabili utilizzati nelle operazioni. Per esempio, in un linguaggio di questo genere, è possibile utilizzare delle variabili *int* come se fossero delle stringhe, in quanto viene eseguito una conversione implicita. In questo PHP assomiglia molto al linguaggio C.

Un'altra qualità è che PHP è un linguaggio interpretato e cross-platform, rendendolo un ottimo strumento da utilizzare per la creazione di pagine web dinamiche.

Col termine "dinamico" si intende un sito o una pagina web che presentano un contenuto generato da un programma, e non dalla lettura "statica" di un file che presenta lo stesso contenuto fino a quando verrà modificato manualmente.

La piattaforma XAMPP inoltre presenta delle interessanti librerie da integrare con PHP, per esempio *cURL*, usata per il trasferimento di informazioni sfruttando vari protocolli.

3.4.2 Implementazione Server

L'idea per l'implementazione del Server è semplicissima : gestire le caratteristiche di GCM e permettere l'invio di messaggi ai client Android.

Una semplice implementazione si può gestire tramite 3 file PHP, che ho chiamato *index.php* , *sendMessage.php* e *regReceiver.php* .

Partiamo dall'ultimo:

regReceiver.php

```
<?php

        $my_json = $_POST['regid'];
```

```
        $filename = "regids.txt";
        $file_writer = file_put_contents($filename, $my_json);
        if ($file_writer == FALSE )
        echo "Error";
        else echo $my_json;
?>
```

Si tratta di un semplice script che però svolge una funzione molto importante. Esso viene utilizzato dal server per interpretare la notifica da parte del client Android di un registration_id .

Una volta chiamato lo script, il server salva nella variabile $my_json le informazioni riconoscibili dal tag ‘regid’ che sono state ricevute tramite metodo POST.

Successivamente, si definisce il nome di un file e si scrive su di esso le informazioni ricevute, grazie al comando file_put_contents($filename, $my_json).

Ho scelto di salvare questi dati su file in quanto a fini dimostrativi è più semplice ed immediato, mentre una più corretta implementazione richiede altre metodologie, come per esempio l’uso di Database.

I file *index.php* e *sendMessage.php* sono strettamente legati, e si può intuire il loro uso già dal nome. Il primo file è la pagina di benvenuto, in cui è presente la form per inserire i dati, dati che verranno poi passati a *sendMessage.php* , che si preoccuperà di spedirli al dispositivo Android.

Vediamo come è possibile questo:

index.php

```
<!DOCTYPE html>
<html>
```

```
    <head>

        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

        <title></title>

    </head>

    <body>

        <form action="sendMessage.php" method="post">

                Messaggio  : <input id="messaggioSpace"
name="messaggio"/><br>

                Firma      : <input id="firmaSpace" name="firma"/><br>

                <input type="submit" /><br>

        </form>

    </body>

</html>
```

Come si può notare, la pagina creata da questo file non fa altro che presentare una semplice form con due campi di testo in cui potere scrivere. E' inoltre presente un tasto che confermerà l'invio dei dati.

Notiamo come ogni campo di inserimento (definito dal tag *<input />* presenti un attributo *name*, e come l'oggetto *<form />* presenti gli attributi *action* e *method*. Soffermiamoci un attimo a spiegare cosa vediamo:

- ***<form action="sendMessage.php" />*** : si tratta di un attributo che permette l'invio dei dati ad un file o ad un indirizzo Internet.
- ***<form method="post" />*** : definisce in che modo devono essere passati. Questo valore può essere GET, POST, PUT ... cioè i metodi di trasferimento dati che vengono definiti dal protocollo HTTP.
- ***<input name="messaggio" />*** : l'attributo definisce una "etichetta", un identificativo dell'informazione che verrà spedita.

Questa pagina quindi offre una form di compilazione nella quale sono presenti due campi di scrittura e un tasto. Alla pressione del tasto, le informazioni all'interno dei due campi verranno presi, "etichettati" rispettivamente col nome "messaggio" e "firma" e passati alla pagina sendMessage.php tramite metodo POST.

Passiamo ora a illustrare il secondo file:

sendMessage.php

```
<?php
    $messaggio = $_POST["messaggio"];
    $firma = $_POST["firma"];

    echo "Messaggio spedito!<br>$messaggio<br>$firma<br><br><br>";

    $my_regid = file_get_contents("regids.txt");
    if ($my_regid != FALSE)
    {
        $current_date = date("d/m/y H:i:s");

        $data = array (
            'messaggio' => $messaggio,
            'firma' => $firma,
            'data' => $current_date
        );

        $content = array (
            'registration_ids' => array($my_regid),
```

```
    'data' => $data
);


$url = 'https://android.googleapis.com/gcm/send';


$server_key = "AIzaSyC2FoaePnCfMG4KKMvf7uBI-OB7lGAxOec";
$headers = array(
    'Authorization: key=' . $server_key,
    'Content-Type: application/json'
);


$ch = curl_init();

curl_setopt($ch, CURLOPT_URL, $url);

curl_setopt($ch, CURLOPT_POST, true);

curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);

curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);

curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($content));
```

```
        $result = curl_exec($ch);
        if($result === FALSE) {
            die(curl_error($ch));
        }
        curl_close($ch);
    }
?>
```

Tramite il comando $_POST è possibile richiamare i dati ottenuti dal file tramite metodo POST. In particolare quello che ci interessa è ottenere quello che l'utente ha deciso di inviare, quindi riempiremo le nostre variabili grazie ai comandi $_POST["messaggio"] e $_POST["firma"] .

Un'altra importante cosa che serve al file è l'identificativo del dispositivo, valore che verrà recuperato da un file di testo locale tramite il comando file_get_contents("regids.txt") .

Dopo un semplice controllo sull'effettivo reperimento dell'identificativo, si passa a compilare una serie di variabili che assembleranno il messaggio da spedire a GCM.

- **$data** identifica un array che contiene i dati che vogliamo spedire, cioè quello che l'utente ha digitato, assieme alla data attuale.
- **$content** è l'array che si aspetta GCM. Deve contenere l'identificativo e il messaggio da reperire al client Android (**$data**).
- **$url** è l'indirizzo dei server GCM. Questo valore cambia se il server è basato su XMPP (http://gcm.googleapis.com) o HTTP (https://android.googleapis.com/gcm/send)
- **$server_key** è il codice ottenuto da Google API Console

- **$headers** , un array bidimensionale che definisce gli headers necessari alla richiesta

Ora che abbiamo definito le nostre variabili, possiamo procede a creare e spedire il messaggio di richiesta. Per far questo, ho sfruttato la libreria cURL .

Con l'oggetto creato sfruttando cURL, $ch, ho definito una serie di opzioni, come indirizzo, metodo, header, se mi aspetto una risposta dopo l'invio, e i dati da spedire, tutti codificati nello standard JSON.

Infine spedisco il messaggio tramite il comando curl_exec($ch).

3.4.3 Implementazione Android

L'implementazione di GCM su Android invece è più estesa, e spazia dalla registrazione ai server di Google, alla notifica al server privato e infine la parte che riguarda la ricezione (e notifica) dei messaggi ricevuti. L'implementazione che ho seguito prevede l'uso di WakeLock. Nella seguente trattazione si suppone di avere già definito tutti i requisiti necessari nel Manifest.

Elenchiamo e diamo un veloce sguardo alle classi che andremo a spiegare:

- *MainActivity.java* : l'activity principale che viene visualizzata per prima quando si accete all'app.
- *GCMMessage.java* : la classe che definisce l'oggetto ricevuto.
- *GCMReceiver.java* : la classe che definisce il WakefulBroadcastReceiver , oggetto che riceverà i messaggi dal server GCM e farà partire il lavoro di "lettura"
- *GCMIntentService.java* : questa classe espone il comportamento dell'IntentService risvegliato dal GCMReceiver. Si occupa di interpretare il messaggio e di salvarlo in un Database locale.

Partiamo nell'esporre la registrazione ai server GCM.

Questa viene svolta nella MainActivity ogni volta che quest'ultima viene creata.

Dapprima vi è un controllo sull'esistenza e reperibilità di un identificativo che potrebbe essere già stato richiesto e salvato dall'app. Se questa ricerca ha esito negativo, si procede nella richiesta ai server GCM. Se l'esito è positivo, viene controllata la versione dell'app, e se è datata, si aggiorna il registration_id.

```
GoogleCloudMessaging gcm;

        private final static int PLAY_SERVICES_RESOLUTION_REQUEST =
9000;

    public static final String PROPERTY_REG_ID = "registration_id";

    private static final String PROPERTY_APP_VERSION = "appVersion";

private final static String SENDER_ID = "487915634567";


    @Override

    public void onCreate(Bundle savedInstanceState) {

if (checkPlayServices()) {

            gcm = GoogleCloudMessaging.getInstance(this);

            regid = getRegistrationId(context);

            if (regid.isEmpty()) {

                registerInBackground();

            }

        } else {

            Log.i(TAG, "No valid Google Play Services APK found.");

        }

}

        private String getRegistrationId(Context context) {

            final SharedPreferences prefs = getGCMPreferences(context);
```

```
            String registrationId = prefs.getString(PROPERTY_REG_ID,
"");

            if (registrationId.isEmpty()) {

                Log.i(TAG, "Registration not found.");

                return "";

            }

            int registeredVersion = prefs.getInt(PROPERTY_APP_VERSION,
Integer.MIN_VALUE);

            int currentVersion = getAppVersion(context);

            if (registeredVersion != currentVersion) {

                Log.i(TAG, "App version changed.");

                return "";

            }

            return registrationId;

        }
```

Il metodo *registerInBackground* si occupa di creare un'AsyncTask, un oggetto che permette di fare operazioni su Thread secondari e gestire il lavoro in differenti fasi, per esempio si può decidere di far effettuare controlli su database in background, e alla fine del lavoro in background, quando il controllo si sposta di nuovo sull'interfaccia, far comparire un messaggio di notifica. Nel nostro caso abbiamo interesse di gestire soltanto la prima fase, cioè la parte in background. Qui verrà effettuata la richiesta ai server GCM e si notificherà il server privato.

```
private void registerInBackground() {

        new AsyncTask() {

          @Override
```

```
          protected Object doInBackground(Object... params)
          {
            try {
                if (gcm == null) {
                  gcm = GoogleCloudMessaging.getInstance(context);
                }
                regid = gcm.register(SENDER_ID);

               if (sendRegistrationIdToBackend())
               {
                 storeRegistrationId(context, regid);
               }
            } catch (Throwable ex)
            {
                ex.printStackTrace();
            }
            return null;
          }
        }.execute(null,null,null);
}
```

Come si nota, tramite *gcm.register(SENDER_ID)* viene fatta una registrazione al nostro progetto e subito dopo si prova a far reperire al server il nostro nuovo registration_id, col metodo *sendRegistrationIdToBackend()*:

```
private boolean sendRegistrationIdToBackend() {

                try {
```

```
                HttpClient client = new DefaultHttpClient();



                HttpPost request = new
HttpPost("http://87.19.241.51:9090/ServerLaurea/regReceiver.php");



                List<NameValuePair> nameValuePairs = new
ArrayList<NameValuePair>(1);

                nameValuePairs.add(new BasicNameValuePair("regid",
regid));



                request.setEntity(new
UrlEncodedFormEntity(nameValuePairs));




        HttpResponse response = client.execute(request);



        String srvResponse =
EntityUtils.toString(response.getEntity());



        Log.i(TAG,"Server response is : "+srvResponse);



        if (srvResponse.equals("Error"))

        {

                DialogFragment newFragment =
myAlertDialogFragment.newInstance("Errore");

                newFragment.show(getFragmentManager(), "dialog");

                return false;

        }

        else

        {
```

```
                DialogFragment newFragment =
myAlertDialogFragment.newInstance("Configurazione GCM avvenuta
correttamente");

                newFragment.show(getFragmentManager(), "dialog");

        }



                } catch (Throwable ex)

                {

                        ex.printStackTrace();

                        return false;

                }

                return true;
```

Come si vede, creo un oggetto HttpPost indirizzato al server (per configurazione il server rimane in ascolto su porta 9090) e alla pagina regReceiver.php. A questo oggetto aggiungo il registration_id col tag “regid” e spedisco il messaggio. In base alla risposta del server, l’app mostrerà un’appropriata notifica. Se la risposta è positiva, il metodo ritorna *true*, altrimenti ritornerà *false* . Se ritorna true, allora tramite *storeRegistrationId(context, regid)*, mi salverò in locale il registration_id.

```
private void storeRegistrationId(Context context, String regId) {
  final SharedPreferences prefs = getGCMPreferences(context);
  int appVersion = getAppVersion(context);
  Log.i(TAG, "Saving regId on app version " + appVersion);
  SharedPreferences.Editor editor = prefs.edit();
  editor.putString(PROPERTY_REG_ID, regId);
  editor.putInt(PROPERTY_APP_VERSION, appVersion);
  editor.commit();
}
```

Con registrazione a GCM e la notifica al nostro server, l'unica cosa che rimane da illustrare è la ricezione dei messaggi. Questa procedura è molto più semplice di quelle descritte precedentemente, ma si articola in più file e classi.

Per prima cosa definiamo cos'è la classe *GCMMessage*. Gli oggetti creati non sono altro che dei "contenitori" per le informazioni, in modo che queste ultime possano essere gestite più facilmente mantenendo un riferimento all'evento di ricezione che accomuna questi dati. La classe quindi prevede una serie di metodi *get* e *set*, per potere gestire i dati della classe, e un costruttore che può già inizializzare questi valori:

```
public class GCMMessage {

        private String msg;

        private String firma;

        private String date;


        public GCMMessage(String msg, String firma, String date)

        {

                this.msg = msg;

                this.firma = firma;

                this.date = date;

        }

        public String getMsg()

        {

                return this.msg;

        }

        public String getFirma()

        {

                return this.firma;

        }
```

```
        public String getDate()
        {
                return this.date;
        }
        public void setMsg(String msg)
        {
                this.msg = msg;
        }
        public void setFirma(String firma)
        {
                this.firma = firma;
        }
        public void setDate(String date)
        {
                this.date = date;
        }
}
```

Passiamo ora a *GCMReceiver*. La classe estende *WakefulBroadcastReceiver* e si occupa di ricevere i messaggi di GCM e ottenere un WakeLock per far assicurare che il lavoro venga svolto senza interruzioni. Anche questa classe è molto semplice, in quanto l'unico metodo che presenta è un *Override*[1] sul metodo *onReceive*.

[1] una notazione con cui si avverte il compilatore che il metodo successivo non sarà uno nuovo istanziato dal developer, bensì sarà una versione "modificata" del metodo. Applicare un Override a metodi del ciclo di vita di un app è importante, altrimenti le versioni modificate non vengono mai richiamate perchè non

```
public class GCMReceiver extends WakefulBroadcastReceiver {

        @Override

        public void onReceive(Context arg0, Intent arg1) {

                ComponentName comp = new
ComponentName(arg0.getPackageName(),GCMIntentService.class.getName());

                startWakefulService(arg0, (arg1.setComponent(comp)));

                setResultCode(Activity.RESULT_OK);

        }

}
```

Come si può notare, all'interno del metodo onReceive non si fa altro che creare un riferimento, o meglio, un oggetto che si riferisce alla classe *GCMIntentService*, e successivamente si fa avviare quest'ultimo.

Analizziamo ora l'ultima classe, cioè *GCMIntentService* appunto, che permette di salvare le informazioni e di mostrare una notifica all'utente.

```
public class GCMIntentService extends IntentService{


        public static final int NOTIFICATION_ID = 1;

        private NotificationManager mNotificationManager;

        NotificationCompat.Builder builder;


        private final static String TAG = "Laurea";
```

---

riconosciute nel modo giusto. Un Override viene semplicemente definito tramite *@Override* prima della definizione del metodo

```
        public GCMIntentService()

        {

                super("GCMIntentService");

        }


        @Override

        protected void onHandleIntent(Intent intent){

                Bundle extras = intent.getExtras();

                GoogleCloudMessaging gcm =
GoogleCloudMessaging.getInstance(this);



                String messageType = gcm.getMessageType(intent);



                String firma = "";

                String messaggio = "";

                String date = "";



                if (!extras.isEmpty())

                {

                        if
(GoogleCloudMessaging.MESSAGE_TYPE_SEND_ERROR.equals(messageType)) {

                                sendNotification("Send error: " +
extras.toString());

                        } else if
(GoogleCloudMessaging.MESSAGE_TYPE_DELETED.equals(messageType)) {

                                sendNotification("Deleted message on
server: "+extras.toString());

                        } else if
(GoogleCloudMessaging.MESSAGE_TYPE_MESSAGE.equals(messageType))

{
```

```
        firma = extras.getString("firma");

        messaggio = extras.getString("messaggio");

        date = extras.getString("data");

        sendNotification("Ricevuta notifica da :" +firma+"\n\""+
messaggio+"\"\n\nInviato in data: "+date);

}

                }

                GCMReceiver.completeWakefulIntent(intent);

        }
```

```
        private void sendNotification(String msg)

        {

                mNotificationManager = (NotificationManager)
this.getSystemService(Context.NOTIFICATION_SERVICE);



                PendingIntent contentIntent =
PendingIntent.getActivity(this, 0, new
Intent(this,MainActivity.class).addFlags(Intent.FLAG_ACTIVITY_CLEAR_TA
SK).addFlags(Intent.FLAG_ACTIVITY_NEW_TASK).addFlags(Intent.FLAG_ACTIV
ITY_CLEAR_TOP), 0);



                NotificationCompat.Builder mBuilder = new
NotificationCompat.Builder(this)

                .setSmallIcon(drawable.ic_launcher)

                .setContentTitle("GCM Notification")

                .setStyle(new
NotificationCompat.BigTextStyle().bigText(msg))

                .setContentText(msg);

                mBuilder.setContentIntent(contentIntent);
```

```
                Notification myNotification = mBuilder.build();

                myNotification.flags |= Notification.FLAG_AUTO_CANCEL;

                mNotificationManager.notify(NOTIFICATION_ID,
myNotification);

        }

}
```

Il metodo su cui ci vogliamo concentrare è *onHandleIntent* , dove vengono reperite le informazioni ricevute tramite l'oggetto Intent ricevuto in input.

In realtà prima di questo vi è una fase di "interpretazione" del messaggio GCM, per capire se l'invio dei dati da GCM è avvenuto correttamente o meno.

Una volta ottenuti i dati, viene chiamato il metodo *sendNotification*, che si occupa di creare una notifica mostrando il messaggio ricevuto, la firma e la data di invio.

Come si nota, la parte più complessa dell'applicativo Android è la fase di registrazione, che coinvolge numerosi controlli e collegamenti online, mentre la ricezione è molto più semplice e lineare.

4. Sensori

4.1 Introduzione

La maggior parte dei dispositivi su cui Android è installato presenta una gamma di sensori in grado di rilevare, tracciare e restituire delle informazioni al sistema operativo, alle app e attraverso quest'ultime all'utente.

Questi sensori possono essere utilizzati per esempio per monitorare la temperatura, l'umidità, la posizione, possono fungere da bussola, o dare informazioni sul livello di luce presente nell'ambiente, misurare la pressione e la prossimità di oggetti davanti allo schermo.

In generale è quindi giusto dire che i sensori permettono di mettere in collegamento il software col mondo reale.

Al giorno d'oggi un dispositivo Android può contare su circa una dozzina di sensori diversi. Questa non è ovviamente un'affermazione "corretta ed universale", in quanto la presenza effettiva di sensori dipenderà dal produttore del device, dal modello, e altre scelte di mercato.

La prima frase va quindi riformulata, definendo che Android *supporta* una dozzina di sensori. Ma anche in questo caso la frase non è del tutto corretta.

L'uso dei sensori non è stato infatti implementato in una sola volta, anzi, si possono identificare almeno 4 "*milestones*", delle versioni importanti in cui il supporto dei sensori ha subito modifiche valide da notare.

Per esempio, l'uso di sensori è stato introdotto nel 2009, con la release 1.5 Cupcake di Android. Al tempo soltanto 6 sensori erano presenti e funzionanti, mentre altri due (barometro e giroscopio) erano presenti ma dovettero aspettare fine 2010, con l'uscita di Android 2.3 Gingerbread.

Come in tutti i sistemi, elementi vengono implementati e altri vengono rimpiazzati. Fra questi ultimi ricadono due sensori presenti dalla versione 1.5, cioè il sensore per l'orientamento, sostituito con Android 2.2 Froyo, e il sensore per la temperatura, sostituito con Android 4.0 Ice Cream Sandwich.

4.1.1 Elenco dei sensori

I sensori si possono suddividere in base a 2 criteri: cosa viene misurato e come si misura.

Partiamo dal primo criterio, il più immediato da spiegare.

Ogni sensore è particolare perchè si occupa di misurare un determinato "ramo" della realtà. Vale a dire che i sensori si dividono in categorie proprio perchè si occupano di diversi elementi, come per esempio un termometro e un barometro possono essere accomunati in quanto rilevano delle condizioni ambientali, ma non possono essere legati ad un giroscopio, che si lega al mondo del movimento.

Ecco che si definiscono quindi tre categorie di sensori:

- Sensori di **moto** : sensori che rilevano moto di accelerazione e rotazione sui 3 assi x, y e z. In questa categoria ricadono accelerometri, giroscopi,sensori di gravità e rotazione vettoriale.
- Sensori di **rilevazione ambientale** : si occupano di rilevare diversi tipi di parametri riguardanti l'ambiente che circonda il dispositivo come illuminazione, temperatura, umidità e pressione. Questi sensori sono barometri, fotometri e termometri.
- Sensori di **posizione** :legati alla posizione del dispositivo, sono sensori di orientamento e magnetometri

Il secondo criterio è invece più "tecnico". Si tratta del modo in cui i sensori sono implementati ed usati all'interno di un dispositivo. Si dividono quindi in due categorie: Hardware e Software.

I sensori Hardware semplicemente sfruttano dei componenti presenti nel dispositivo, cioè sono implementati fisicamente.

I sensori Software invece semplicemente imitano un sensore Hardware, di solito prendendo informazioni da questi ultimi e rielaborando tali informazioni in maniera autonoma, per poi presentarle.

Invece di presentare singoli esempi per illustrare questo secondo criterio di suddivisione, sfruttiamo l'occasione e illustriamo in tabella tutti i sensori disponibili, definendo di cosa si occupano e quali sono gli usi comuni

Sensore	*Tipo*	*Descrizione*	*Usi comuni*
Accelerometro	Hardware	Misura la forza di accelerazione in m/s^2 applicata su un dispositivo su tutti e 3 gli assi, assieme alla forza di gravità	Determinare il movimento del dispositivo
Temperatura (OLD)	Hardware	Determina la temperatura dell'ambiente in °C, ma dato che presentava implementazioni diverse in base al tipo di dispositivo, è stato sostituito nella versione 4.0	Misurare la temperatura
Temperatura (NEW)	Hardware	Determina la temperatura dell'ambiente in °C.	Misurare la temperatura
Gravità	Software o Hardware	Misura la forza di gravità applicata sul dispositivo sui 3 assi	Determinare il movimento del dispositivo
Giroscopio	Hardware	Misura in rad/s la rotazione del dispositivo sui 3 assi	Determinare eventi di rotazione

Fotometro	Hardware	Misura l'illuminazione dell'ambiente in lx (SI).	Controllare l'illuminazione dello schermo
Accelerazione lineare	Software o Hardware	Misura la forza di accelerazione in m/s^2 applicata al dispositivo su tutti e 3 gli assi, ma non la forza di gravità.	Controllare l'accelerazione su un singolo asse
Campo magnetico	Hardware	Misura il campo geomagnetico dell'ambiente su tutti e 3 gli assi in μT	Uso della bussola
Orientamento	Software	Misura i gradi di rotazione che un dispositivo effettua su tutti e 3 gli assi. Dalla versione 1.5 la matrice di inclinazione e di rotazione possono essere ottenuti usando il sensore di gravità e del campo magnetico, assieme al metodo *getRotationMatrix()*	Determinare la posizione del dispositivo
Barometro	Hardware	Misura la pressione dell'aria in hPa o mbar	Monitorare la pressione ambientale
Prossimità	Hardware	Misura la prossimità di un oggetto allo schermo del dispositivo, in cm.	Determinare in quale posizione è il telefono durante una chiamata, se per esempio è vicino all'orecchio
Umidità relativa	Hardware	Misura l'umidità relativa dell'ambiente in percentuale	Misurare il "punto di rugiada" e l'umidità relativa o assoluta.
Vettore di rotazione	Software o Hardware	Misura la rotazione del dispositivo fornendo i tre elementi del vettore rotazione del dispositivo	Rilevamento di movimento e rotazione

4.2 Sensor Framework

Questo framework ci consente di interfacciarci con i sensori descritti in precedenza. Quattro sono gli elementi cardine di questo framework, cioè:

1. **SensorManager** : una classe "master", un manager appunto, che fornisce al developer tutti gli strumenti necessari per accedere ed elencare i sensori, registrare e de-registrare listeners per gli stessi e accede a delle costanti per lavorare all'interno del framework. E' anche la classe che presenta il metodo *getRotationMatrix()* citato precedentemente riguardo il sensore di orientamento.
2. **Sensor** : la classe che definisce istanze dei sensori veri e propri. Tramite i metodi di questa classe possiamo ricavare informazioni preziose sulle possibilità e impostazioni del sensore che stiamo analizzando
3. **SensorEvent** : la classe i cui singoli oggetti sono le informazioni ricavate dai sensori. Le informazioni dipendono ovviamente dal tipo di sensore, ma fra queste sono anche disponibili l'accuratezza e la data della rilevazione.
4. **SensorEventListener** : interfaccia propria dei sensori, un listener che resta in ascolto di nuove rilevazioni di dati o modifiche sull'accuratezza dei sensori, presentando due metodi per poter agire nelle casistiche descritte

4.3 Come utilizzare i sensori?

Abbiamo descritto tutto ciò che può esserci utile per capire meglio di cosa stiamo parlando e cosa ci può servire.

Passiamo ora a definire come utilizzare quel che abbiamo appreso.

4.3.1 Passo 1: gestire diversi dispositivi

In precedenza abbiamo affermato che Android può si supportare diversi sensori, ma deve essere prima il produttore del dispositivo a decidere di usare questo hardware che ci interessa!

Ma allora, come possiamo costruire un'app che usa sensori, se non abbiamo la certezza dei dispositivi su cui verrà avviata?

Ci sono due metodi ovviare a questa problematica:

1. Controllare run-time la presenza dei sensori
2. Definire nel Manifest che il sensore interessato sia richiesto

Cosa vuol dire questo?

Nel primo caso diamo a tutti i dispositivi la possibilità di installare ed usare l'applicazione. Soltanto quando poi l'applicazione verrà utilizzata, tramite opportuni controlli si verificherà l'effettiva presenza dei sensori, e si potrà decidere cosa fare.

Questo comporta maggiore flessibilità ed estendibilità dell'app, ma ovviamente comporta una maggiore definizione di "regole", cioè come si deve comportare un'app quando manca un certo sensore, se deve essere segnalata la mancanza o semplicemente ignorata, il tutto quindi si tramuta in maggiore tempo di implementazione.

Esempio:

```
//Definiamo il nostro SensorManager
SensorManager manager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);

// controlliamo la presenza del giroscopio
if (manager.getDefaultSensor(Sensor.TYPE_GYROSCOPE) != null)
{
   .. // il sensore è disponibile
```

```
}
else {

 ..   // il sensore NON è disponibile
}
```

Nel secondo caso invece abbiamo un comportamento più rigido, in quanto diamo a Google Play Store l'incaricho di non fare visualizzare la nostra app a dispositivi non supportati.

I benefici sono una più facile implementazione a livello di codice, ma la nostra app risentirà di poca estendibilità, in quanto rischia di essere supportata su una minore gamma di prodotti. A questo punto una soluzione potrebbe essere lo sviluppo di più versioni in parallelo, che presentino metodi diversi di funzionamento in base ai sensori disponibili, ma questa è solo una possibile scelta di implementazione, che non è detto sia applicabile in tutti i casi.

Per esempio, per richiedere che il dispositivo abbia l'accelerometro :

```
<uses-feature android:name="android.hardware.sensor.accelerometer"
android:required="true" />
```

4.3.2 Passo 2: implementare il codice

A livello di codice si usano le classi del SensorFramework descritte in precedenza.

Definiamo prima l'ambiente in cui ci troviamo, una generica activity:

```
public class ActivityProva extends Activity implements
SensorEventListener

{

private SensorManager mSensorManager;

private Sensor light;
```

```
@Override

protected void onCreate(Bundle savedInstanceState)

{

        ..
}
```

Si parte dall'ottenere il SensorManager

```
@Override

protected void onCreate(Bundle savedInstanceState)

{

  mSensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);

  ...

}
```

Successivamente si creano gli oggetti Sensor a cui faremo riferimento come sensori veri e propri.

```
@Override

protected void onCreate(Bundle savedInstanceState)

{

  mSensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);

  ...

  light = mSensorManager.getDefaultSensor(Sensor.LIGHT);

  ...

}
```

Si registra poi un listener per gli oggetti Sensor appena creati, definendo un delay, cioè ogni quanto effettuare una rilevazione dei dati. Registrazione e de-registrazione dei listener si fanno di solito nei metodi onPause() e onResume().

```
@Override

protected void onPause()

{

  super.onPause();

  mSensorManager.unregisterListener(this);

}



@Override

protected void onResume()

{

  super.onResume();

  mSensorManager.registerListener(this, light,
SensorManager.SENSOR_DELAY_NORMAL);

}
```

Notare il metodo *registerListener* del SensorManager : richiede in input 3 valori.

Il primo è il SensorEventListener, il secondo è l'oggetto Sensor definito in precedenza, il terzo è il delay con cui vengono raccolti i dati dal sensore.

Dato che la nostra Activity implementa il SensorEventListener, ci basterà definire i seguenti due metodi e l'app funzionerà a dovere.

```
@Override

public void onAccuracyChanged(Sensor sensor, int accuracy) {

  // metodo che viene chiamato quando si cambia l'accuratezza del
```

```
  // sensore
}

@Override
public void onSensorChanged(SensorEvent event) {
  // il metodo che viene chiamato ogni delay, e offre l'oggetto
  // SensorEvent che ha I dati che ci interessano.
  // è di solito in questo metodo che si applicano le scelte di
  // programmazione, come aggiornare una TextView o modificare la
  // luminosità dello schermo
}
```

L'app così creata registrerà all'avvio l'uso del sensore della luminosità, e ogni volta che delle rilevazioni saranno state fatte, sarà possibile ottenere i dati dall'oggetto SensorEvent del metodo onSensorChanged.

L'oggetto SensorEvent presenta degli attributi molto interessanti :

- `accuracy` , tramite cui otteniamo l'accuratezza della rilevazione
- `timestamp` , la data della rilevazione
- `sensor` , ci restituisce l'oggetto Sensor su cui abbiamo registrato il listener
- `values[]`, un array di float che contiene i dati rilevati dal sensore.

`values[]` è interessante in quanto il modo in cui bisogna leggerlo, o meglio, interpretarlo varia in base al sensore utilizzato. Per esempio, con un fotometro, avremo un array monodimensionale, e accederemo ai dati per esempio come

```
float luce = event.values[0];
```

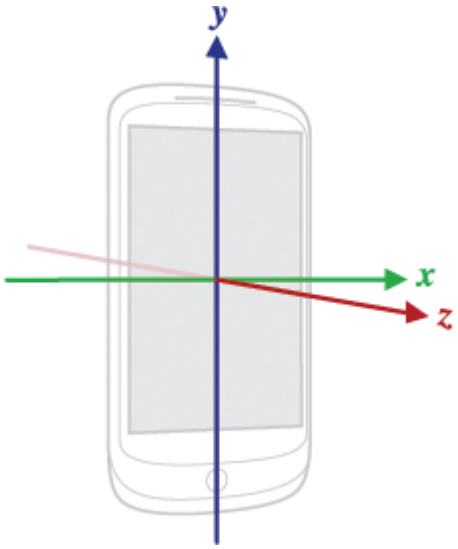
mentre se abbiamo a che fare con un accelerometor, che misura la forza di accelerazione sugli assi x, y e z, potremmo avere il seguente scenario :

```
float x = event.values[0];

float y = event.values[1];

float z = event.values[2];
```

I sensori di **moto** (tranne quello a *Vettore di rotazione*) e il sensore di **posizione** del *Campo magnetico* utilizzano lo stesso metodo di rappresentazione dei valori, cioè




```
values[0] = x

values[1] = y

values[2] = z
```

Il sensore a *Vettore di rotazione* oltre a questi 3 valori, presenta anche un quarto dato, il componente scalare del vettore di rotazione, che però è opzionale

Anche l'ormai obsoleto sensore di **posizione**, quello dell'orientamento, basa i suoi valori sui 3 assi, ma in modo diverso :

```
values[0] = Azimuth ( gradi attorno all'asse z)

values[1] = Inclinazione ( gradi attorno all'asse x)

values[2] = Rotolio ( gradi attorno all'asse y)
```

Tutti gli altri sensori invece riportano soltanto un valore.

Conclusioni

Nel documento presentato abbiamo conosciuto meglio il sistema operativo di Android, mettendo a luce alcuni dei suoi aspetti più interessanti per lo sviluppo di app.

Sono state analizzate le procedure di creazione, implementazione ed utilizzo di tre diversi "usi" di Android, esponendo capacità e potenzialità di ciascuno di essi. Abbiamo osservato e studiato cosa sono queste tecnologie e come si implementano in un possibile contesto.

Senza dubbio Android ha dimostrato di avere le carte in regola per rimanere uno dei sistemi operativi per smartphone preferiti e di maggior successor, ancora per molto tempo.

Le possibilità di miglioramento ci sono, e ogni aggiornamento vengono introdotti nuovi elementi da utilizzare e imparare. Basti pensare all'evoluzione dei sensori, come man mano sono cresciuti di numero per analizzare l'ambiente circostante. Si pensi anche al fatto che Maps e GCM entrambi sono alla loro seconda versione, in quanto vi era bisogno di un miglioramento sostanziale non raggiungibile attraverso normali aggiornamenti.

Le opportunità che riserva il futuro degli smartphone sono vaste. Si potrebbe pensare per esempio a nuovi sensori, come quello della lettura di impronte digitali recentemente aggiunto da Apple, o nuovi modi per effettuare transazioni tramite mobile, per esempio con identificazione tramite lettura della retina.

L'argomento di tesi che ho affrontato è stato sicuramente interessante, e mi ha aiutato ad integrarmi meglio nel mondo Java, dandomi anche la possibilità di imparare sul campo a sfruttare al meglio le risorse e le potenzialità che un ambiente mette a disposizione.

Bibliografia

- [HA09] Sharon P. Hall, Eric Anderson, *"Operating systems for mobile computing"*, pp. 64-69, 2009
- [RKAMK12] , Radhika Rani, A. Praveen Kumar, D. Adarsh, K. Krishna Mohan, K.V.Kiran , *"Location Based Services In Android"*, K L University, vol. 3, pp. 209-220, 2012
- [KK11], Amit Kushwaka, Vineet Kushwaka, *"Location Based Services Using Android Mobile Operating System"* , Department of Information Technology, Indian Institute of Information Technology, vol. 1, pp 14-20, 2011
- [JN13], David Jaramillo, Richard Newhook, *"Cross Platform, Secure Message Delivery for Mobile Devices"*, 2013
- [ST13], Shanmugapriya M, Tamilarasi A ,"*Design and Development of Mobile Assisted Language Learning (MALL) application for English Language using Android Push Notification Services"*, International Journal of Research in Computer and Communication Technology , vol. 2, pp. 329-338, 2013
- [LCLWS13], Penghui Li, Yan Chen, Taoying Li, Renyuan Wang, Junxiong Sun, *"Implementation of Cloud Messaging System Based on GCM Service"*, International Conference on Computational and Information Sciences, pp. 1510-1512, 2013
- [MA09] Michael Arrington, *"Google Redefines GPS Navigation Landscape: Google Maps Navigation For Android 2.0"*, http://techcrunch.com/2009/10/28/google-redefines-car-gps-navigation-google-maps-navigation-android/ , 2009
- http://it.wikipedia.org/wiki/IOS_(Apple)
- http://en.wikipedia.org/wiki/Android_version_history
- http://developer.android.com/index.html