ALMA MATER STUDIORUM

REPORT LABORATORIO DI APPLICAZIONI MOBILI

PROGETTO: WIRELESS CONNECTIVITY MAP

Wibo - Applicazione Per La Costruzione di Mappe sulla Connettività Wireless

Studente
MICHELE DI STEFANO

Anno Accademico
2018/2019

January 13, 2019

Contents

Chapter 1

Analisi

In questo capitolo vengono analizzati i requisiti che l'applicazione deve soddisfare e descritti i casi d'uso che riguardano l'interazione dell'utente con l'applicazione.

1.1 Requisiti

1.1.1 Funzionalità 1

Codificare le aree di Google Maps:

- utilizzare l'API di Google Maps
- scegliere una codifica valida per rappresentare aree invece che punti (no solo GPS)
- le aree dovrebbero essere codificate in modo tale che l'intero spazio sia coperto senza la presenza di "buchi" (e.g. l'uso di cerchi non è valido a meno che non si sovrappongano)

1.1.2 Funzionalità 2

Codificare il *Received-Signal-Strength-Indicator* (RSSI):

- visualizzare la potenza della connettività (RSSI) attraverso la colorazione delle area con una scala di colori (e.g. da rosso a verde)

- i dati dovrebbero essere acquisiti di nuovo dopo che sia trascorso un periodo di tempo (selezionabile dall'utente)
- due (o più) misure circa la stessa area devono essere rappresentate in qualche modo (e.g. trasparenza del colore)

1.1.3 Funzionalità 3

Supportare almeno tre tecnologie:

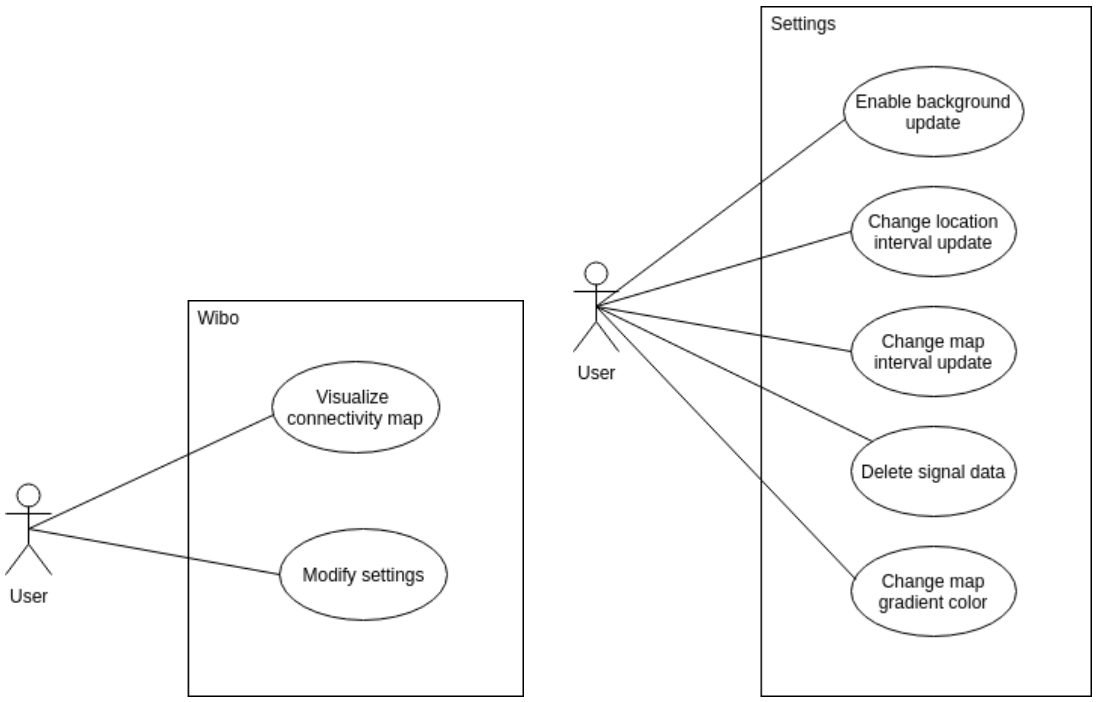
- l'applicazione dovrebbe mostrare almeno tre diverse mappe in base alla tecnologia usata:
  - LTE (4G)
  - UMTS (3G)
  - WIFI

1.2 Casi d'uso

Di seguito sono riportati i casi d'uso relativi all'interazione dell'utente con l'applicazione in generale, e all'interazione dell'utente con le impostazioni (settings) in particolare.

Le principali azioni che producono un comportamento completo e significativo per l'utente sono le seguenti:

- visualizzare la mappa di connettività
- abilitare/disabilitare gli aggiornamenti in background
- modificare l'intervallo degli aggiornamenti di posizione
- modificare l'intervallo degli aggiornamenti della mappa
- cancellare i dati sulle aree memorizzati nel database
- modificare il colore del gradiente utilizzato per colorare le aree della mappa



(a) wibo (b) settings

Figure 1.1: Casi d'uso

Chapter 2

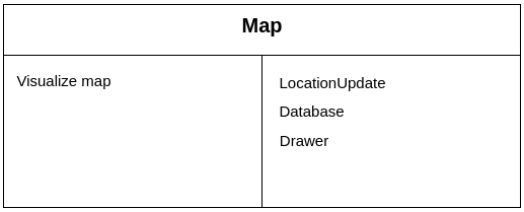
Progettazione

Nella fase di progettazione è stato seguito un approccio Object Oriented, tuttavia tenendo in considerazione le componenti architetturali che riguardano lo sviluppo di applicazioni per Android.

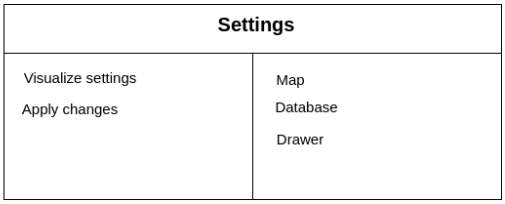
2.1 CRC card

Nel processo di identificazione delle classi principali sono state utilizzate le *Class-Responsibility-Collaboration* (CRC) card, strumento impiegato nella progettazione Object Oriented che permette di astrarsi dai dettagli implementativi e di focalizzarsi sugli elementi essenziali della classe, evitando così un livello di complessità che, in questa fase, potrebbe essere controproducente.

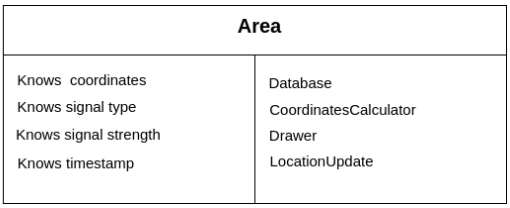
In basso sono rappresentate le classi più importanti con le principali responsabilità e successivamente è riportata una breve descrizione di ognuna.



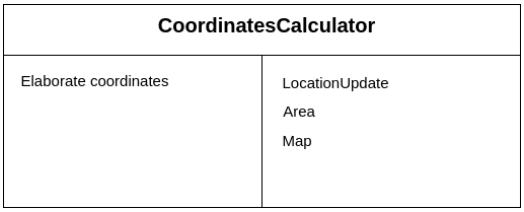
(a) Map



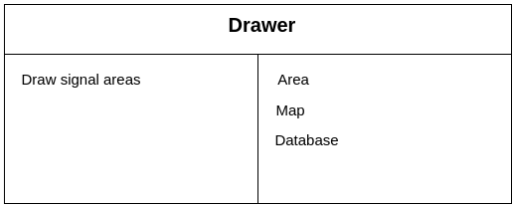
(b) Settings



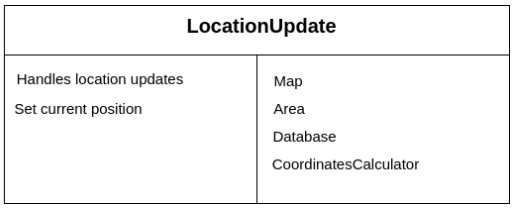
(c) Area



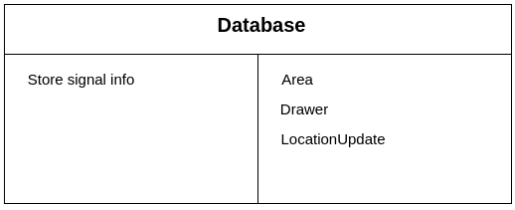
(d) CoordinatesCalculator



(e) Drawer



(f) LocationUpdate



(g) Database

Figure 2.1: CRC card

2.1.1 Descrizione

Map

La responsabilità primaria di questa classe è quella di visualizzare la mappa sulla quale verranno rappresentate opportunamente le aree.

Settings

Le responsabilità principali di questa classe sono: visualizzare le impostazioni modificabili dall'utente e salvare le eventuali modifiche rendendole effettive.

Area

La responsabilità primaria di questa classe è quella di conoscere le informazioni relative ad una determinata area, in particolare: coordinate, tipo di segnale, potenza del segnale, marca temporale.

CoordinatesCalculator

La responsabilità primaria di questa classe è quella di elaborare le coordinate acquisite tramite GPS.

Drawer

La responsabilità primaria di questa classe è quella di disegnare un'area sulla mappa, occupandosi di gestire la scelta del colore e della trasparenza.

LocationUpdate

Le responsabilità principali di questa classe sono: gestire gli aggiornamenti di posizione ed impostare la posizione corrente all'avvio dell'applicazione.

Database

La responsabilità primaria di questa classe è quella di garantire la persistenza delle informazioni rilevate sulle aree.

2.2 Database

Per quanto riguarda la progettazione del database questa è stata suddivisa in più fasi.

2.2.1 Progettazione concettuale

Nella fase di progettazione concettuale è stato fatto ricorso ai diagrammi *Entity-Relationship* (ER) per modellare il dominio che, in questo caso, consisteva nelle informazioni relative ad un'area.

Gli attributi rilevanti identificati nel dominio sono stati i seguenti:

- **mgrsTen:** indica le coordinate dell'area nel formato *Military-Grid-Reference-System* (MGRS)
- **signalStrength:** indica la potenza del segnale in una scala da 0 a 4
- **connectivityType:** indica il tipo di connettività alla quale fa riferimento la misura
- **timestamp:** indica l'istante nel quale sono state rilevate le informazioni e funge da chiave primaria

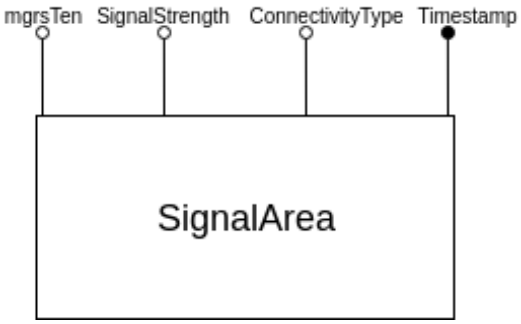


Figure 2.2: Diagramma ER per l'area

2.2.2 Operazioni

Le operazioni, relative al database, ritenute utili ai fini dell'applicazione sono risultate essere le seguenti:

1. inserire rilevamenti relativi ad un'area
2. calcolare la media della potenza del segnale relativa ad un'area per un determinato tipo di connettività

3. calcolare la media della potenza del segnale relativa ad un determinato tipo di connettività, raggruppata per coordinate mgrs
4. cancellare tutti i rilevamenti relativi ad un determinato tipo di connettività
5. ottenere tutti i rilevamenti relativi ad un determinato tipo di connettività e ad una specifica area
6. ottenere il timestamp massimo per un determinato tipo di connettività
7. ottenere tutti i valori di potenza del segnale per un determinato tipo di connettività

2.2.3 Traduzione verso il modello relazionale

Il diagramma ER rappresentato sopra, nel modello relazionale, si traduce nel seguente schema

SignalArea(**timestamp**, mgrsTen, signalStrength, connectivityType)

rappresentabile dalla tabella sottostante

SignalArea

timestamp	signalStrength	connectivityType	mgrsTen

Figure 2.3: tabella SignalArea

2.2.4 Codifica SQL

Di seguito sono riportate le operazioni codificate in *Structured-Query-Language* (SQL). [1]

Operazione 1

Vedi nota[2]

Operazione 2

```
SELECT ROUND(AVG(signalStrength),0) AS signalStrength, mgrsTen,
       COUNT(mgrsTen) AS areaOccurrences FROM signalarea_table
WHERE mgrsTen = :mgrsTenID AND connectivityType = :connectivityTypeID
```

Operazione 3

```
SELECT ROUND(AVG(signalStrength),0) AS signalStrength, mgrsTen,
       COUNT(mgrsTen) AS areaOccurrences FROM signalarea_table
WHERE connectivityType= :connectivityType
GROUP BY mgrsTen
```

Operazione 4

```
DELETE FROM signalarea_table
WHERE connectivityType = :connectivityType
```

Operazione 5

```
SELECT * FROM signalarea_table
WHERE mgrsTen = :mgrsTenID AND connectivityType= :connectivityType
```

[1] `:name` sta ad indicare che `:name` è un parametro passato alla query quando la si invoca tramite DAO.

[2] facendo questo progetto uso della libreria Room, per le operazioni standard (`INSERT`,`DELETE`,`UPDATE`) non è necessario fornire il corrispettivo in SQL, in quanto sono già forniti degli appositi costrutti. Pertanto non è riportata la codifica dell'operazione 1.

Operazione 6

```
SELECT MAX(timestamp) FROM signalarea_table
WHERE mgrsTen = :mgrsTenID AND connectivityType= :connectivityType
```

Operazione 7

```
SELECT signalStrength FROM signalarea_table
WHERE mgrsTen = :mgrsTenID AND connectivityType= :connectivityType
```

Chapter 3

Implementazione

In questo capitolo sarà fornita una visione dal generale al particolare sulla struttura del codice e saranno trattate alcune scelte implementative riguardanti le classi utilizzate.

3.1 Java package

Per motivi di maggior leggibilità, di facilitazione della manutenzione e per raggruppare classi che contribuiscono ad espletare una determinata funzionalità, il codice è stato suddiviso in 5 *package*: activity, coord, draw, persistence, utils (vedi Figure 3.1).

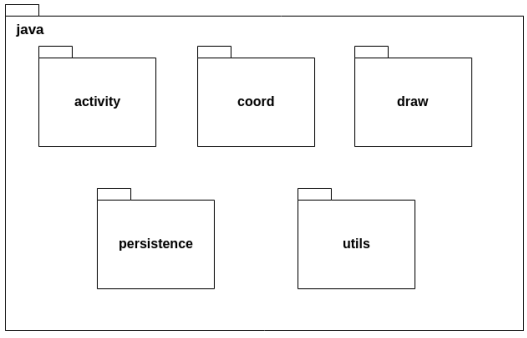


Figure 3.1: java package

Il contenuto di ogni package sarà analizzato ora più nel dettaglio.

activity

In questo package sono contenute le classi `MapsActivity` e `SettingsActivity` (vedi Figure 3.2) che costituiscono le due Activity dell'applicazione.

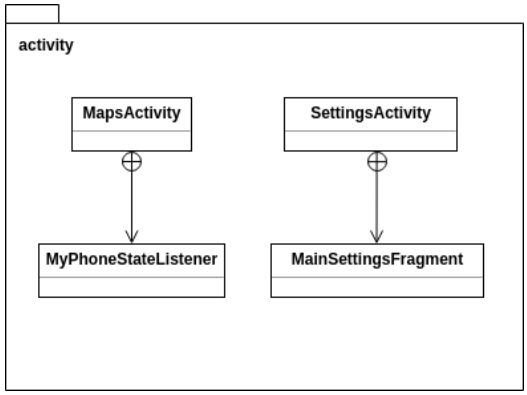


Figure 3.2: activity

`MapsActivity` è l'Activity principale visualizzata dall'utente all'avvio dell'applicazione e che mostra la mappa.
Contiene la classe `MyPhoneStateListener` necessaria per registrare i cambiamenti di segnale delle connettività UMTS, LTE, GSM tramite la callback `onSignalStrengthsChanged()`.
Gestisce altre callback come, ad esempio, quelle che si occupano di rispondere all'utente con le azioni opportune quando clicca le opzioni che figurano nella toolbar o le callback che vengono invocate in base alla presenza, o meno, di determinati permessi richiesti dall'applicazione.
Presenta inoltre alcune funzioni ausiliarie per identificare il tipo di connettività attivo o per cancellare la registrazione di un *receiver/listener*. Queste funzioni sono utili, ad esempio, quando l'Activity invoca `onPause()`, `onResume()`.

`SettingsActivity` è l'Activity che viene lanciata quando l'utente clicca sull'icona delle impostazioni presente nella toolbar.
Contiene la classe `MainSettingsFragment` che ha il compito di visualizzare le preferences tramite le quali l'utente può modificare il comportamento dell'applicazione. A tal fine al suo interno sono definite anche la callback `OnSharedPreferenceChanged()` e le costanti utilizzate come chiave per accedere ai valori salvati nelle `SharedPreferences`.

coord

In questo package è contenuta la classe `CoordinatesCalculator` ed il package `nasalib` (vedi Figure 3.3).

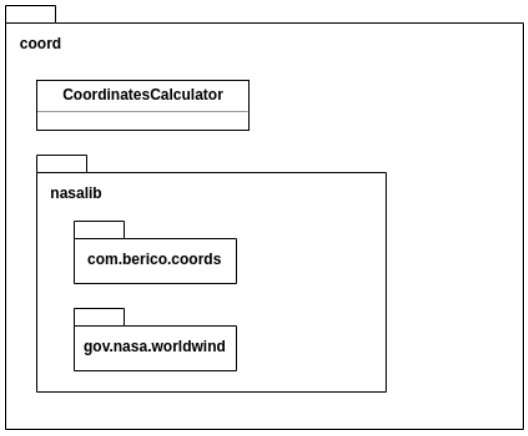


Figure 3.3: coord

Il package `nasalib` contiene a sua volta altri due package che, nel complesso, espletano la funzionalità di conversione tra diversi sistemi di coordinate geospaziali (tra cui MGRS e latitudine/longitudine). I package sono estratti dal progetto NASA World Wind.

`CoordinatesCalculator` tramite l'utilizzo del package sopracitato, gestisce l'elaborazione delle coordinate. Come funzionalità principali, consente la conversione da latitudine e longitudine a MGRS e viceversa, e da MGRS a MGRS approssimato ad aree aventi una lunghezza di 10m per lato.

draw

In questo package è contenuta la classe `AreaDrawer` (vedi Figure 3.4).

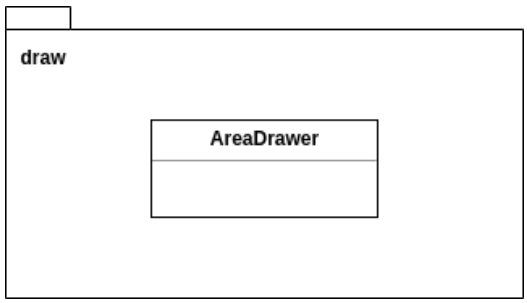


Figure 3.4: draw

`AreaDrawer` è una classe che si occupa di disegnare le aree colorate sulla mappa. Implementa il design pattern *Singleton*.
Contiene una `HashMap` che mappa la potenza del segnale con diversi colori. Il mapping è modificabile tramite il metodo `setGradientColor()`, che invoca una procedura interna per modificare l'`HashMap`.
Presenta il metodo `getOpacityLevel()` per selezionare il livello di opacità del colore ed inoltre il metodo `drawLocalAreaSquare()` che disegna l'area sulla mappa.

persistence

In questo package sono contenuti tutti gli elementi necessari al fine di implementare la persistenza dei dati. I package principali all'interno di persistence sono: dao, database, entity, utils (vedi Figure 3.5).

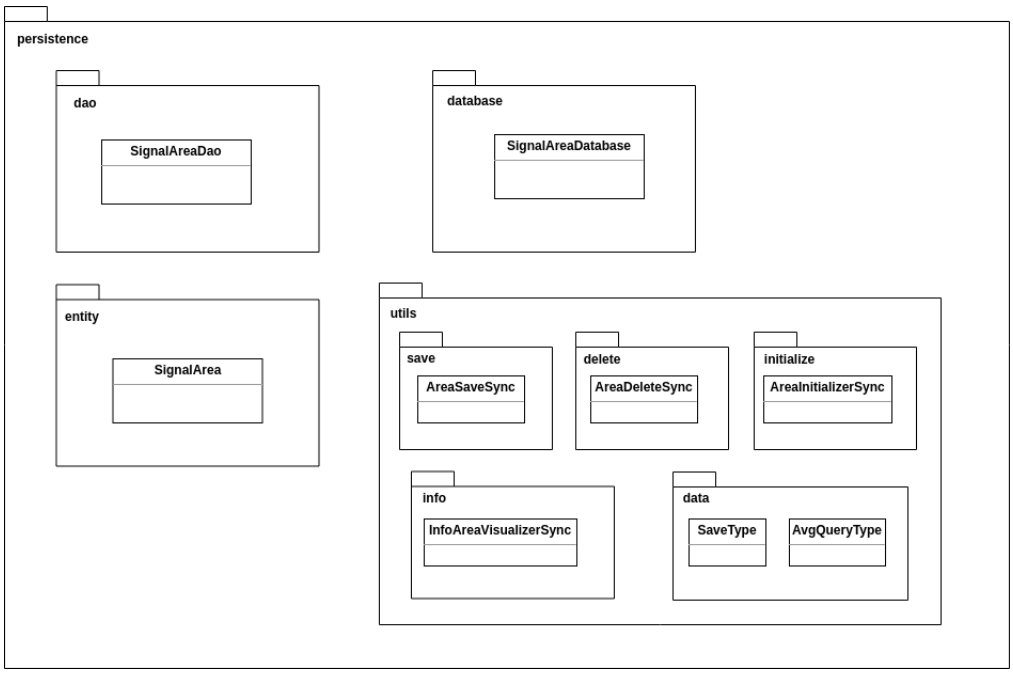


Figure 3.5: persistence

Il package `dao` contiene la classe interfaccia `SignalAreaDao` dove vengono definite le interazioni con il database, ovvero le query (definite nel capitolo 2) di cui avremo bisogno.

Il package `database` contiene la classe astratta `SignalAreaDatabase`, la quale estende la classe `RoomDatabase` ed implementa il design pattern *Singleton*, consigliabile in quanto l'istanziazione di `RoomDatabase` è costosa. Rappresenta il punto di accesso ai dati persistenti.

Il package `entity` contiene la classe `SignalArea` che rappresenta una singola tabella all'interno del database. Come visto nel capitolo 2 questa calsse concretizza il dominio da analizzare (informazioni relative all'area).

Il package `utils` raggruppa package contenenti classi che ricoprono diverse funzioni ausiliarie. La maggior parte rappresentano task asincroni che accedono al database.[1] La loro implementazione è brevemente de-

[1]non è possibile eseguire operazione sul database Room tramite il thread UI.

scritta di seguito:

- **save:** contiene il task asincrono `AreaSaveAsync` che si occupa di accedere al database per salvare i dati. Inoltre se dall'ultimo rilevamento dell'area a quello più recente è trascorso il periodo di tempo selezionato dall'utente per l'aggiornamento della mappa il task disegna l'area sulla mappa [2].

- **delete:** contiene il task asincrono `AreaDeleteAsync` che accede al database per eliminare i dati tramite la query che implementa l'operazione 4.

- **initialize:** contiene il task asincrono `AreaInitializerAsync` che accede al database per recuperare i dati tramite la query che implementa l'operazione 3 e disegnare successivamente sulla mappa le aree relative ad un tipo di connettività.

- **info:** contiene il task asincrono `InfoAreaVisualizerAsync`, invocato quando l'utente clicca a lungo su di un'area della mappa. Accede al database utlizzando la query che implementa l'operazione 2, per occuparsi poi, della visualizzazione della `InfoWindow` relativa all'area in questione.

- **data:** contiene classi che rappresentano due tipi di dati utilizzati per memorizzare il risultato di alcune query.

utils

In questo package sono contenute classi ausiliare che riguardano la la connettività wifi, gli aggiornamenti di posizione e le finestre di informazione (vedi Figure 3.6).

[2] tramite il metodo `onPostExecute()` che torna ad utilizzare il thread UI

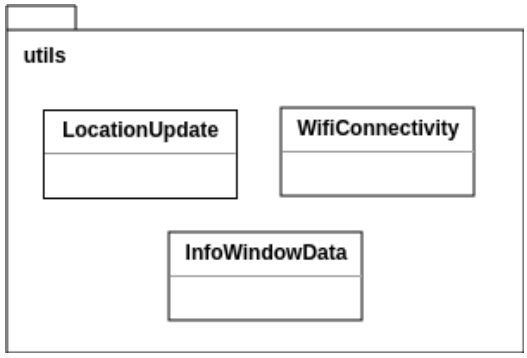


Figure 3.6: utils

`LocationUpdate` è la classe che si occupa di gestire la richiesta, l'avvio, lo stop e la ricezione degli aggiornamenti di posizione. Le informazioni sono reperibili dall'oggetto `Location` che l'applicazione recupera tramite il *fuesed location provider*. Inoltre tramite il metodo `setCurrentPosition()`, all'avvio dell'applicazione, segnala sulla mappa l'ultima posizione conosciuta (corrente).

`WifiConnectivity` presenta due metodi: `checkWifiConnection()` che controlla se il wifi è attivo sfruttando il `ConnectivityManager`.
`scanWifiSignal()` che definisce la callback `onReceive()` la quale gestisce la ricezione dei risultati della scansione ed inoltre avvia la scansione degli hotspot circostanti .

`InfoWindowData` rappresenta un dato utilizzato per creare dei tag da associare ai marker, al fine di utilizzarne il contenuto per customizzare le finestre di informazioni predefinite.