

Cellular Connectivity and Noise Map

Relazione

Xia · Tian Cheng

Matricola: 0000975129

Email: tiancheng.xia@studio.unibo.it

Anno accademico

2022 — 2023

Corso di Laboratorio di applicazioni mobili
Alma Mater Studiorum · Università di Bologna

Indice

1	Introduzione	1
1.1	Feature implementate	1
1.2	Screenshot applicazione	1
2	Scelte progettuali	3
2.1	Informazioni generali	3
2.1.1	Organizzazione package	3
2.2	Mappa	3
2.2.1	Generazione cella	3
2.2.2	Generazione griglia	4
2.3	Raccolta dei dati	5
2.3.1	Struttura e memorizzazione delle misurazioni	5
2.3.2	Sampler	5
2.4	Servizi in background	6
2.5	Condivisione dati	6
2.5.1	Esportazione	7
2.5.2	Importazione	7
2.6	Componenti	7
2.6.1	App principale	7
2.6.2	Impostazioni	7
2.6.3	Condivisione	7
3	Problemi noti	7
3.1	Scansione Wi-Fi	7
3.2	Servizio in background	7
3.3	Importazione	7

1 Introduzione

1.1 Feature implementate

Di seguito sono elencate le feature implementate:

- Mappa partizionata in aree non sovrapposte con ridimensionamento automatico delle celle in base al livello dello zoom (Figura 1).
- Range della qualità delle misurazioni calcolato automaticamente, con possibilità di scegliere il numero di classi da creare (Figura 2).
- Misurazione di Wi-Fi, LTE, rumore e Bluetooth con le seguenti modalità:
 - Attiva su comando dell'utente.
 - Passiva dopo un determinato intervallo temporale o durante il movimento.
 - In background durante il movimento.
- Filtro di ricerca per alcune tipologie di misurazioni (Wi-Fi e Bluetooth)
- Notifiche per aree prive di misurazioni recenti.
- Esportazione su file e importazione delle misurazioni (Figura 3).

1.2 Screenshot applicazione

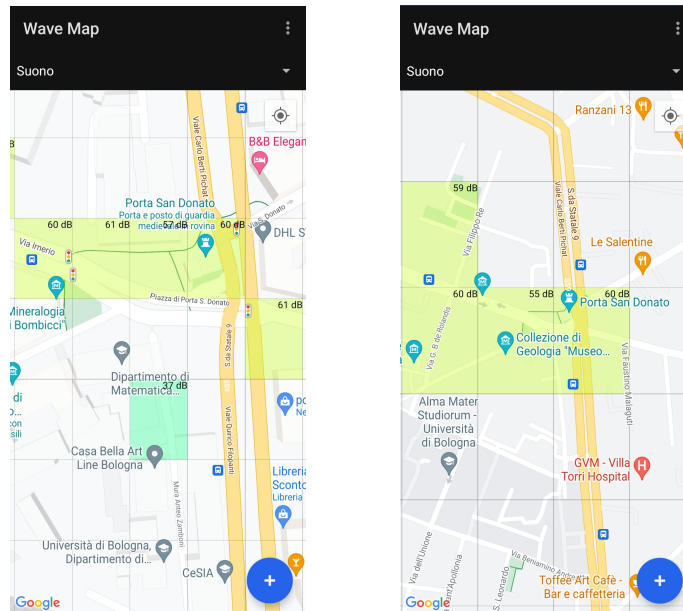
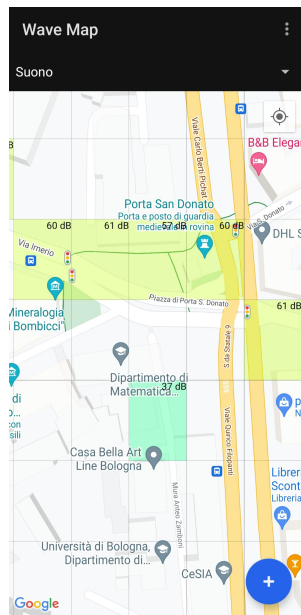
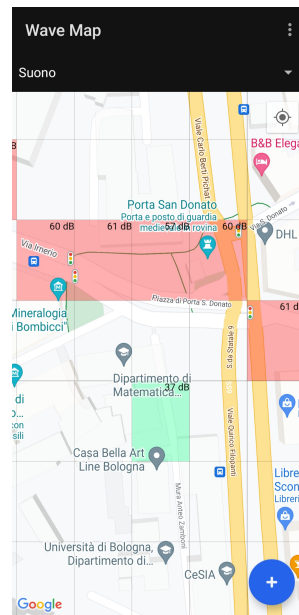


Figura 1: Mappa con celle ridimensionate automaticamente



Suddivisione in 3 range



Suddivisione in 2 range

Figura 2: Range misurazioni calcolati algoritmicamente

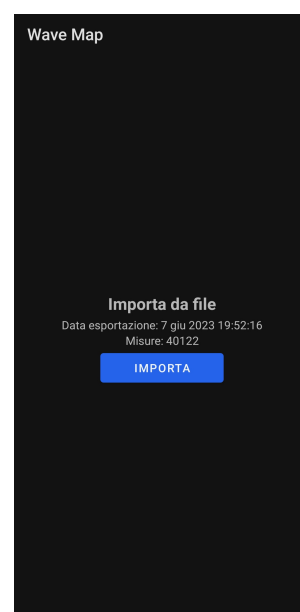
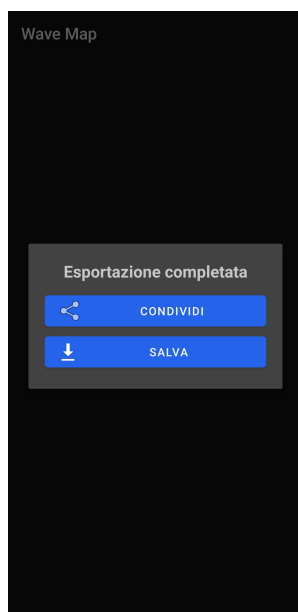
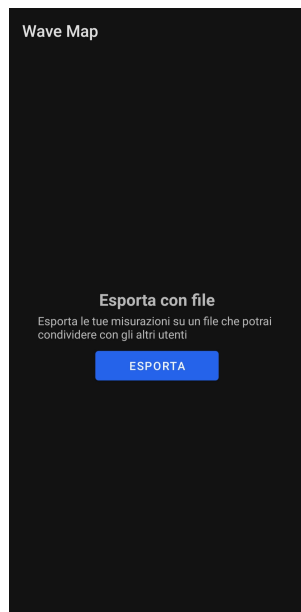


Figura 3: Esportazione e importazione da file

2 Scelte progettuali

2.1 Informazioni generali

Il progetto è stato sviluppato come applicazione nativa utilizzando Kotlin. Come pattern architetturale è stato scelto l'approccio Model-View-ViewModel, mentre per l'interfacciamento con il database locale è stata utilizzata la libreria Room.

Le operazioni asincrone sono state principalmente implementate tramite le `coroutine` e per favorire un codice più "lineare", quando possibile, sono state trasformate le funzioni con callback in funzioni `suspend` utilizzando come wrapper `suspendCoroutine`.

2.1.1 Organizzazione package

In Tabella 1 è descritta l'organizzazione dei package:

<code>db</code>	Classi che implementano la struttura e le operazioni sulle tabelle del database.
<code>dialogs</code>	Classi per istanziare i dialog utilizzati nell'applicazione.
<code>measures</code>	Classi che implementano le operazioni per effettuare le misurazioni.
<code>services</code>	Classi che implementano i servizi in background.
<code>ui</code>	<code>Activity</code> , <code>Fragment</code> e <code>ViewModel</code> dell'applicazione.
<code>utilities</code>	Metodi e variabili di utilità generale.

Tabella 1: Organizzazione package

2.2 Mappa

Per la mappa è stato utilizzato *Google Maps* e l'implementazione è contenuta nel fragment `WaveHeatMapFragment`.

2.2.1 Generazione cella

Una cella della mappa rappresenta la misurazione di un'area quadrata¹ e la dimensione di quest'ultima scala automaticamente in base al livello dello zoom.

Una cella è descritta dalle coordinate del vertice superiore sinistro (nord-ovest) e a partire da questa vengono calcolate le coordinate degli altri convertendo la dimensione della cella (in metri) in un offset da applicare a latitudine e longitudine. Per questioni estetiche, gli offset sono approssimati in modo tale che tutte le righe siano allineate verticalmente (Figura 4).

¹Esclusa la zona equatoriale, le celle appariranno rettangolari



Figura 4: Offset calcolati in maniera precisa (sinistra) e approssimata (destra)

Il colore di una cella è determinato dal valore e dal tipo di misurazione, e viene impostato in formato HSV modificando il valore della tinta (hue). Per ciascuna tipologia di misurazione sono specificati gli estremi del range di qualità e il numero di intervalli in cui suddividerlo. La tinta è quindi determinata collocando il valore della misurazione in uno dei sotto-intervalli del range e selezionando come valore finale l'estremo inferiore (Algoritmo 1).

Algoritmo 1: Tinta di una cella

```

1 fun getHue(measure, measure_range=[a, b], hue_range=[0, 150], n_ranges)
2   hue ← measure scalata da measure_range a hue_range
3   hue ← hue discretizzata in hue_range suddiviso in n_ranges valori
4   return hue
5 end

```

Infine, per ogni cella è presente un'etichetta contenente il valore della misurazione. Poiché *Google Maps* non permette di inserire esplicitamente del testo nella mappa, l'implementazione prevede di generare l'etichetta come un'immagine che viene poi impostata come icona di un marker. Inoltre, la dimensione del testo viene scalata per far in modo che rientri nei limiti della cella.

2.2.2 Generazione griglia

La griglia è composta da celle generate relativamente ad una posizione di riferimento. In particolare, in fase di inizializzazione viene designata come cella di riferimento quella che pone la posizione dell'utente al centro e in base a questa è possibile determinare la posizione di tutte le altre celle della mappa.

Nello specifico, date delle coordinate ($\text{pos}_{\text{lat}}, \text{pos}_{\text{lon}}$), per determinare la cella che la contiene si calcola il numero di celle da saltare rispetto a quella di riferimento:

$$\text{to_skip_tiles}_{\text{lat}} = \lceil \frac{\text{pos}_{\text{lat}} - \text{center_top_left}_{\text{lat}}}{\text{latitudeOffset}(\text{tile_length_meters})} \rceil$$

$$\text{to_skip_tiles}_{\text{lon}} = \lfloor \frac{\text{pos}_{\text{lon}} - \text{center_top_left}_{\text{lon}}}{\text{longitudeOffset}(\text{tile_length_meters})} \rfloor$$

Le coordinate del vertice superiore sinistro della cella che contiene ($\text{pos}_{\text{lat}}, \text{pos}_{\text{lon}}$) sono quindi:

$$\text{tile}_{\text{lat}} = \text{center_top_left}_{\text{lat}} + (\text{to_skip_tiles}_{\text{lat}} \cdot \text{latitudeOffset}(\text{tile_length_meters}))$$

```
tilelon = center_top_leftlon + (to_skip_tileslon · longitudeOffset(tile_length_meters))
```

Con questo approccio, ogni volta che viene spostata la visuale della mappa, la griglia viene generata iterando a partire dalle coordinate dell'angolo nord-ovest visibile dello schermo, fino a raggiungere l'angolo sud-est.

In aggiunta, per maggiore efficienza, si tiene traccia delle celle generate in modo da evitare di ridisegnare una cella già presente. Questo meccanismo viene resettato quando viene cambiato il livello di zoom, in quanto tutte le celle già presenti diventano obsolete e vengono cancellate.

2.3 Raccolta dei dati

2.3.1 Struttura e memorizzazione delle misurazioni

Una misurazione è descritta dall'interfaccia **WaveMeasure** e contiene il valore della misurazione, un timestamp, la posizione e un flag per indicare se si tratta di una misurazione propria o ottenuta tramite condivisione. In aggiunta, è presente un campo per informazioni aggiuntive utile per distinguere alcune tipologie di misurazioni (es. per Wi-Fi e Bluetooth viene salvato il BSSID).

L'interfaccia **WaveMeasure** viene quindi utilizzata per implementare la classe **MeasureTable** che descrive la tabella del database dedicata per memorizzare le misurazioni. Tutte le misurazioni sono salvate nella stessa tabella e sono differenziate da un campo (**type**) aggiunto in fase di salvataggio nel database. Inoltre, ciascuna misura è identificata univocamente da un UUID, utile anche per differenziare le misurazioni provenienti da altri utenti.

In aggiunta, una seconda tabella descritta da **BSSIDTable** contiene la mappatura da BSSID a SSID.

2.3.2 Sampler

Per la raccolta dei dati è stato introdotto il concetto di *sampler* per gestire in maniera modulare le misurazioni. Nello specifico, un *sampler* è descritto dalla classe astratta **WaveSampler** e richiede l'implementazione dei seguenti metodi:

- **sample** per prendere una nuova misurazione.
- **store** per il salvataggio dei dati nel database.
- **retrieve** per la ricerca dei dati note le coordinate dei vertici di una cella della mappa.

Inoltre, sono esposte le seguenti funzioni ausiliarie:

- **average** richiama **retrieve** e restituisce la media dei valori.
- **sampleAndStore** richiama in sequenza **sample** e **store**.

Per maggiore flessibilità, le misure vengono sempre intese come liste di **WaveMeasure**. Ciò permette di gestire misurazioni che per loro natura non generano un'unica misurazione (es. Wi-Fi e Bluetooth).

A partire da **WaveSampler** sono quindi implementati i *sampler* per:

- Wi-Fi (**WiFiSampler**):

- Ottiene la potenza della rete al quale il dispositivo è attualmente connesso tramite il servizio di sistema `ConnectivityManager`. Per versioni inferiori alla API 29, viene invece utilizzato il `WifiManager`.
- Misura la potenza delle reti circostanti registrando un `BroadcastReceiver` con filtro `WifiManager.SCAN_RESULTS_AVAILABLE_ACTION` e richiedendo una scansione completa attraverso il `WifiManager`.
- Bluetooth (`BluetoothSampler`):
 - Misura la potenza dei dispositivi accoppiati mediante il `BluetoothManager`.
 - Misura la potenza dei dispositivi circostanti registrando un `BroadcastReceiver` con filtro `BluetoothDevice.ACTION_FOUND` e richiedendo al `BluetoothManager` una scansione completa.
- LTE (`LTESampler`): ottiene la potenza del segnale LTE tramite il `TelephonyManager`.
- Suono (`NoiseSampler`): viene fatta la media di una serie di campionature effettuate utilizzando un `MediaRecorder`.

2.4 Servizi in background

La classe `BackgroundScanService` estende `Service` e implementa le funzionalità dell'applicazione attive in background.

In fase di avvio del servizio, viene inizializzato un `FusedLocationProviderClient` impostato per fornire periodicamente la posizione del dispositivo. In particolare, gli aggiornamenti vengono notificati dopo che è stata percorsa una distanza minima dalla posizione precedente e hanno una priorità impostata a `PRIORITY_BALANCED_POWER_ACCURACY` per preservare la batteria del dispositivo.

Le operazioni effettuate alla ricezione di un aggiornamento della posizione sono:

1. Verifica se l'area è coperta da misurazioni, in caso negativo viene inviata una notifica. Per evitare di inviare notifiche troppo frequentemente, viene tenuto traccia del momento in cui è stata inviata quella più recente in modo tale da poter ignorare le notifiche successive se dovessero essere create a distanza troppo ravvicinata.
2. Effettua una misurazione completa utilizzando tutti i *sampler*.

2.5 Condivisione dati

Le operazioni per l'esportazione e l'importazione dei dati sono implementate come metodi statici della classe `ShareMeasures`.

La condivisione dei dati avviene tramite la creazione di un file contenente le misurazioni. Per maggiore interoperabilità con eventuali estensioni, i dati vengono salvati in formato JSON e per tale scopo viene utilizzata la libreria `Gson`.

2.5.1 Esportazione

L'esportazione è gestita dal fragment `FileExportFragment` e `ViewModel FileExportViewModel`.

Durante la fase di esportazione, tutte le misurazioni e tutti i BSSID salvati nel database vengono convertiti in una stringa JSON, assieme ad alcuni metadati. Il risultato è quindi temporaneamente salvato nella cache dell'applicazione.

L'utente ha quindi la possibilità di salvare il risultato in un file locale nella cartella *Downloads* oppure di condividerlo attraverso un `Intent` di tipo `ACTION.SEND`.

2.5.2 Importazione

L'importazione è gestita dall'activity `ImportActivity` e `ViewModel ImportViewModel`.

`ImportActivity` ha come `intent-filter` azioni del tipo `VIEW` e `SEND`, e quindi permette di selezionare l'azione di importazione quando l'utente apre o condivide un file.

Una volta aperto e validato un file di misurazioni, l'importazione consiste nell'inserire nel database tutte le misurazioni presenti marcandole come misure ottenute per condivisione. Per evitare duplicati, l'UUID viene usato come discriminante e vengono quindi ignorate tutte le misurazioni già presenti.

Analogamente, la tabella dei BSSID viene importata escludendo le righe già note.

2.6 Componenti

2.6.1 App principale

2.6.2 Impostazioni

2.6.3 Condivisione

3 Problemi noti

3.1 Scansione Wi-Fi

3.2 Servizio in background

3.3 Importazione