Alma Mater Studiorum · Università di Bologna

**SCUOLA DI SCIENZE**

**Corso di Laurea in Informatica per il Management**

# SVILUPPO E INTEGRAZIONE DI UN SISTEMA PER LA CREAZIONE DI SERVIZI DECENTRATI SU SMARTPHONE

**Relatore:**
**Dott.**
**FEDERICO MONTORI**

**Presentata da:**
**MICHELE BONINI**

**II Sessione**
**Anno Accademico 2019/2020**

**Sommario**

Al giorno d'oggi, l'attenzione verso la situazione climatica del nostro pianeta si sta facendo sempre più forte. Il nostro stile di vita sta danneggiando noi e l'ambiente circostante. Per esempio, la grandissima quantità di veicoli in circolazione nelle città è fonte di inquinamento il quale è in grado di causare problemi alla salute dell'essere umano e alla natura. Essere informati maggiormente e costantemente su queste tematiche potrebbe sensibilizzare le persone a limitare e moderare i propri comportamenti al fine di evitare ulteriori catastrofi climatiche. Inoltre, l'emergenza Covid-19 che stiamo fronteggiando in questi mesi è un'altra situazione critica per noi e la nostra salute. In questo lavoro di tesi verrà discussa l'applicazione mobile da me realizzata e denominata GeoMonitor, la quale permetterà agli utenti di essere più consapevoli del contesto nel quale essi sono immersi. Infatti, grazie a GeoMonitor sarà possibile monitorare costantemente l'ambiente circostante e magari rapportarlo con il proprio stato di salute oltre ad essere informati sul numero di nuovi positivi da coronavirus giornalieri in determinate zone. GeoMonitor si può considerare come un'estensione della piattaforma IoT collaborativa SenSquare la quale permette di costruire dei servizi individuali o utili alla comunità, utilizzando fonti di dati eterogenei e campagne di crowdsensing. Tali servizi saranno visibili ed utilizzabili anche dalla mia applicazione ed, in aggiunta, se ne potranno creare di nuovi utilizzando anche i parametri locali inerenti alla salute nel caso l'utente abbia necessità particolari, senza condividerli con SenSquare.

# 1

All we have to do is decide what to do with the time that is given to us.

*The Lord of the Rings*

# Indice

# Elenco delle figure

# Introduzione

Al giorno d'oggi gli smartphone stanno diventando sempre più sofisticati e potenti grazie al progredire della tecnologia, e l'avvento dell'Internet of Things ha permesso di estendere ulteriormente le loro funzionalità. Infatti, l'IoT permette alle persone ed agli oggetti di essere sempre connessi tramite la rete Internet. Da ciò possono derivare innumerevoli vantaggi. Per esempio, la transizione alla quale stiamo assistendo negli ultimi anni che consiste nel passaggio da semplici case ad abitazioni definite "intelligenti" o *smart home*, in alcuni casi porta dei vantaggi anche dal punto di vista economico per via del risparmio energetico. Oltre alla domotica, altri possibili contesti di questa tecnologia sono le smart cities, smart environment o l'agricoltura smart ed il loro utilizzo è finalizzato al miglioramento della qualità della vita, ad uno sviluppo delle città più sostenibile e, come anticipato prima, vi è anche un aspetto economico. Nell'Internet of Things i dispositivi come gli smartphone, i computer, le televisioni, i tablet, i sensori e tutti gli oggetti "intelligenti" possono essere collegati tra loro consentendogli di comunicare. Alcuni di essi, grazie all'utilizzo del context-awareness, sono in grado di comprendere ed analizzare il contesto in cui si trovano per poi adeguarsi al suo cambiamento offrendo determinati servizi all'utente. L'applicazione proposta da questo lavoro di tesi è un'estensione del progetto SenSquare il quale consiste in una piattaforma IoT collaborativa che, utilizzando dati eterogenei, è in grado di offrire determinati servizi agli utenti. Si potranno utilizzare sia fonti di dati affidabili/inaffidabili sia compagne di crowdsensing, tuttavia in questa tesi non verrà descritto come SenSquare recupera tali dati. I servizi offerti da questa piattaforma consistono nella creazione di template e nella possibilità di istanziarli in determinate zone. Un template non è altro che lo scheletro delle istanze che verrà caricato ed eseguito con i dati necessari durante il processo di istanziazione.

Tali dati consistono nel valore delle misurazioni dei parametri presenti nella struttura del template e in una determinata area geografica. Infatti, il processo di istanziazione di un template permetterà agli utenti di monitorare i valori di specifiche aree geografiche. In fase di costruzione di un template, si potranno utilizzare sia dati ambientali (per esempio PM10, temperatura, umidità, ...) sia dati inerenti al numero di nuovi positivi da coronavirus associati alle regioni italiane. GeoMonitor, l'applicazione Android proposta da questa tesi, permetterà all'utente di visionare ed istanziare rispettivamente i template e le istanze presenti su SenSquare. Tuttavia offrirà anche la possibilità di costruire dei propri template i quali, oltre ai dati ambientali e covid-19, potranno utilizzare anche parametri inerenti la salute dell'utente utilizzatore. Tali valori verranno aggiunti manualmente dall'utente in una parte dedicata dell'applicazione (realizzata per il corso di applicazioni mobili e migliorata nel corso di questa tesi). I template costruiti ed istanziati dal dispositivo non verranno in alcun modo condivisi in rete, pertanto i dati sensibili inerenti la salute dell'utente rimarranno in locale garantendo il rispetto della privacy. Questa tesi è stata suddivisa in 5 capitoli:

- Il capitolo 1, lo stato dell'arte, analizzerà il contesto nel quale si trova l'applicazione GeoMonitor;
- Il capitolo 2 mostrerà l'architettura di SenSquare prima e dopo i miei contributi, elencando le principali funzionalità di GeoMonitor;
- Il capitolo 3 descriverà quali chiamate API sono state utilizzate e/o implementate;
- Il capitolo 4 dettaglierà l'implementazione dell'applicazione GeoMonitor;
- Il capitolo 5 mostrerà l'interfaccia grafica dell'applicazione;

Successivamente verranno presentate le mie conclusioni dove proporrò alcune possibili applicazioni future ed evidenzierò le difficoltà riscontrate in fase di sviluppo di questo progetto. Infine, è presente un'appendice la quale descriverà il comportamento della sezione adibita alla raccolta delle informazioni riguardanti lo stato di salute dell'utente (la quale non è stata approfondita eccessivamente dal momento che la maggior parte del carico di lavoro è stato svolto durante il progetto di applicazioni mobili).

# Capitolo 1

# Stato dell'Arte

## 1.1 Internet of Things

Come anticipato in precedenza, questo lavoro di tesi estende l'applicativo SenSquare, il quale si occupa di proporre determinate funzionalità o servizi all'utente sulla base di dati provenienti da fonti governative o campagne di Crowdsensing attraverso l'utilizzo dell'Internet of Things Collaborativo.

### 1.1.1 Che cos'è l'Internet of Things?

L'Internet of Things (o Internet delle Cose) è un paradigma che coinvolge vari rami della tecnologia, dall'hardware all'utilizzo di sensori e sistemi di comunicazione. Esso permette di collegare e fare comunicare tra loro oggetti "intelligenti" [3] attraverso una rete Internet. Il termine Internet of Things fu proposto da Kevin Ashton [4] nel 1999, sostenendo che l'IoT avrebbe avuto il potenziale di cambiare il mondo come fece l'invenzione di Internet, se non ancora di più. Come sostenuto da [5], non vi è ancora una definizione ben precisa di Internet of Things. Tuttavia potremmo accettare la seguente, proposta da [6], in quanto riassume in linea generale il concetto di IoT: "*L'Internet of Things permette alle persone e agli oggetti di essere sempre connessi, in qualsiasi posto con qualsiasi oggetto e qualunque persona, usando idealmente un percorso/rete e qualsiasi tipo di servizio*". Le applicazioni di questa tecnologia variano dall'industria aerospaziale fino ad arrivare all'utilizzo personale per esempio con la domotica. Asin

e Gascon hanno suggerito di organizzare tali applicazioni all'interno di dodici categorie quali per esempio smart cities, smart environment, agricoltura smart o home automation [7]. Le smart cities, per esempio, puntano a sfruttare l'IoT per la creazione di tecnologie di comunicazione e servizi in grado di supportare l'amministrazione della città ed i cittadini . Alcune applicazioni le possiamo ritrovare nell'automazione del monitoraggio delle condizioni strutturali degli edifici (riducendo in questo modo lo sforzo umano), nella rilevazione della qualità dell'aria o nella gestione del traffico dei veicoli [8].

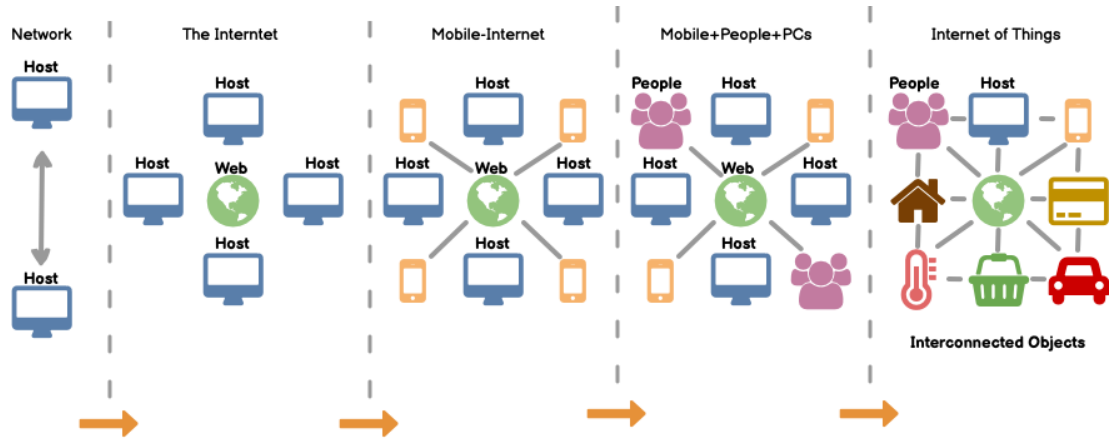



Figura 1.1: L'evoluzione di Internet in cinque fasi, dalla connessione di due computer tra loro fino ad arrivare all'Internet of Things odierno.

### 1.1.2 Il ruolo dei sensori

Un ruolo fondamentale all'interno di questa tecnologia è quello giocato dalle reti di sensori. Quando parliamo di sensori ci riferiamo a quei dispositivi elettronici in grado di rilevare una grandezza in una determinata unità di misura (interagendo con essa) e convertirla in una grandezza di natura diversa [9]. Al giorno d'oggi abbiamo una vasta gamma di tipologie diverse di sensori. Il loro mercato risulta essere in rapida crescita ed il loro utilizzo sta permettendo sempre di più il riconoscimento in modo pervasivo del contesto individuale e sociale nel quale gli utenti dei dispositivi sono immersi. Una rete di sensori è composta da uno o più sensori connessi tra loro attraverso differenti modalità per esempio Internet (la maggior parte di essi utilizzano una rete wireless). Secondo [5] vi sono tre tipi di architettura delle reti di sensori:

- Reti con architettura piatta;
- Reti con architettura a due livelli;
- Reti con architettura a tre livelli (utilizzando la rete Internet).

L'Internet of Things si basa sull'architettura a tre livelli.

### 1.1.3 IoT Collaborativo e Crowdsensing

Che cos'è l'Internet of Things Collaborativo? L'Internet of Things collaborativo è un paradigma sul quale, come anticipato precedentemente, è basato l'applicativo SenSquare. L'idea alla base di questa tecnologia consiste nel coinvolgere e fare cooperare tra loro gruppi di persone nella raccolta di dati e nella condivisione dei servizi, come sostenuto da [10]. In particolare, si parla di monitoraggio basato sulla comunità (Community Based Monitoring o CBM) che viene definito come un "*processo in cui i cittadini interessati, le agenzie governative, l'industria, il mondo accademico, i gruppi, della comunità e le istituzioni locali collaborano per monitorare, tracciare e rispondere a questioni di comune interesse ambientale della comunità*" [11]. Ciò che emerge è che l'utente finale non è più considerato come un semplice consumatore ma bensì parte fondamentale nella raccolta dei dati che verranno successivamente condivisi con la comunità. Si parla di Mobile Crowdsensing quando vengono utilizzati i dispositivi mobili delle persone per collaborare nella raccolta dei dati. Esistono tre categorie di applicazioni Mobile Crowdsensing [12]:

Figura 1.2: Esperimento di Simon Weckert [1].

- Ambientale;
- Infrastrutturale;
- Sociale.

Ad esempio, Google Maps è un'applicazione di tipo infrastrutturale ed è in grado di monitorare la posizione e la velocità degli utenti al fine di misurare i livelli di congestione del traffico. Nel mese di Febbraio del 2020, Simon Weckert, un'artista berlinese, fece un esperimento provando ad "ingannare" gli algoritmi di Google inserendo 99 smarthpones, con l'applicazione di Google Maps avviata, all'interno di un carrellino trascinandolo a piedi per alcune strade della città. Google Maps indicò quelle strade come congestionate dal traffico [13].

## 1.2 Context-Awareness

### 1.2.1 Contesto e consapevolezza

Uno dei primi lavori ad introdurre il concetto di context-aware fu quello proposto da Schilit e Theimer nel 1994 [14], associando il contesto all'ambiente circostante oppure a determinate situazioni. In particolare, i due autori si riferirono al contesto inteso come posizione, persone/oggetti nelle circostanze oppure alle interazioni che tra essi sarebbero potute avvenire. Sempre secondo gli autori, per capire meglio il tipo di contesto bisognerebbe farsi quattro domande: dove ti trovi, chi sei, chi è vicino a te e quali oggetti hai vicino. Qualche anno più tardi, nel 1997, la tecnologia stava avanzando molto rapidamente, per esempio i telefonini stavano diventando programmabili ed i sensori iniziavano a permettere a dispositivi mobili di tenere traccia del contesto. In quegli anni, quindi, vi era un interesse crescente per dispositivi che potessero permettere l'utilizzo di applicazioni context-aware. Fu in quel periodo che Brown e Bovey definirono il contesto inteso come le informazioni relative ad una posizione, per esempio l'ora del giorno, la stagione dell'anno o la temperatura [15]. Tuttavia questa definizione, secondo Schilit, non fu particolarmente corretta e negli anni avvenire si decise per definire il contesto come "*ogni informazione che può essere caratterizzata da una situazione o da un'entità*" [16], dove per entità si intende un luogo, una persona o un oggetto. Ancora, il contesto si definisce come "*le condizioni correlate in cui qualcosa esiste o si verifica*" in una data situazione in cui è presente un attore o un evento [17]. Ma che cos'è effettivamente il contesto e a che cosa serve? Il contesto è una fonte di dati che può essere presa ed analizzata per caratterizzare la situazione o un'entità. Quando gli uomini interagiscono tra loro, sono

in grado di utilizzare il contesto per estrarne informazioni ed arricchire la conversazione. Ma quando l'uomo interagisce con le macchine, non si riesce ad identificare quali elementi siano effettivamente importanti all'interno del contesto. In particolare, gli utenti dispongono di un meccanismo impoverito per fornire input alle macchine, e ciò non permette di sfruttare le potenzialità dovute ad una interazione uomo-macchina che utilizza il contesto. Le applicazioni context-aware, fanno riferimento a sistemi che sono "consapevoli" del contesto ovvero sono in grado di comprendere l'ambiente in cui si trovano per poi successivamente adattarsi al suo continuo cambiamento. L'utilizzo di questa tecnologia permette di fornire agli utenti servizi ad hoc sulla base delle loro richieste utilizzando il contesto in cui si trovano. L'idea chiave alla base delle applicazioni context-aware è quello di incoraggiare gli utenti a raccogliere e condividere i dati favorendo la creazione di una rete in grado di prendere decisioni autonome sempre più precise ed offrire servizi migliori. Le categorie di applicazioni context-aware sono due: quelle di tipo personale e quelle di gruppo o comunitarie.

### 1.2.2 Vantaggi e punti deboli

Uno dei vantaggi dell'utilizzo del context-awareness potrebbe derivare dall'automazione di determinati servizi riducendo l'intervento umano. I punti deboli del context-awareness applicato a dispositivi mobili, consistono nelle limitate risorse in termini di batteria e potenza di calcolo se confrontati con potenzialità di un server. In particolare la gestione della batteria è una delle problematiche principali che vincola l'utilizzo di tale tecnologia. Una possibile soluzione sarebbe quella di spegnere o disattivare i sensori che non stiamo effettivamente utilizzando. Tuttavia alcuni ricercatori stanno cercando di trovare soluzioni che possano garantire un buon compromesso tra il consumo di energia e l'accuratezza dei sensori.

## 1.3 Geo-fencing

Una delle applicazioni del context-awareness è il Geo-fencing. Quando parliamo di Geo-fencing stiamo facendo riferimento all'utilizzo di geo-fences (o recinti virtua-

li) all'interno di applicazioni mobili. Un geo-fence non è altro che un confine virtuale posizionato in una determinata area geografica del mondo reale attorno al dispositivo. Tuttavia ogni dispositivo non può avere più di 100 geo-fences attivi contemporaneamente. Per stabilere un geo-fence devono essere impostate una latitudine, una longitudine ed un raggio [18]. Le applicazioni che utilizzano il Geo-fencing, si occupano di recuperare la posizione dell'utente in maniera continuativa al fine di determinare dove si trova. Successivamente potrà essere predisposto un sistema di notifiche che avviserà l'utilizzatore qualora sia entrato nell'area geo-fence o ne sia uscito. Inoltre, sempre grazie al Geo-fencing, è possibile recuperare il periodo passato dall'utente in una determinata area.


Figura 1.3: Esempio di un'area Geo-fence [2].

## 1.4 Covid-19

Oltre ai vari dati ambientali che possono essere monitorati dalla applicazione sulla quale si basa questo lavoro di tesi, vi sono anche il numero di nuovi positivi (giornalieri) contagiati dal Covid-19 per ogni regione d'Italia.

### 1.4.1 Che cos'è il Covid-19?

Il Covid-19 (detto anche Coronavirus) è una malattia respiratoria infettiva che si è presentata a partire da dicembre 2019. I primi casi riscontrati di coronavirus (i quali sintomi sono simili a quelli di una polmonite) provenivano dalla città cinese di Wuhan. Al giorno 9 novembre 2020 sono stati confermati 50.517.420 contagi con 1.257.922 decessi e 33.083.920 guarigioni [19].

## 1.5 Altre applicazioni

Come argomentato precedentemente, vi sono numerose applicazioni che utilizzano la proprietà del context-awareness. Sebbene sul mercato non esistano applicazioni con le funzionalità che sono state ideate ed implementate in questo lavoro di tesi, in questa sezione verranno introdotte quelle più simili.

### 1.5.1 Monitoraggio della salute e Ambientale

Dall'AppStore e dal Google Play Store è possibile scaricare applicazioni in grado di comunicare la temperatura della nostra città o le previsioni meteorologiche. Inoltre, sempre su questi store digitali, sono presenti app in grado di monitorare l'inquinamento ambientale o i livelli di polline, per esempio **AirMatters** [20] e **BreezoMeter** [21] (disponibili entrambe per i sistemi operativi Android e iOS). Tuttavia esse non dispongono di tutte le funzionalità offerte da GeoMonitor, l'app sulla quale è basata questa tesi. Un'ulteriore applicazione collaborativa riguardante questo tema è **Creek Watch** [22] la quale permette di monitorare i corsi d'acqua attraverso la partecipazione degli utenti. In particolare, quando un'utente passerà vicino ad un corso d'acqua, potrà segnalare (sull'applicazione) la quantità dell'acqua, la sua velocità ed il numero di rifiuti nelle vicinanze. Ciò potrà essere fatto inserendo dei dati manualmente oppure scattando delle foto. All'interno dell'app è presente una mappa che evidenzierà le zone nelle quali gli utenti hanno contribuito al monitoraggio. I dati raccolti saranno utili a tener traccia dell'inquinamento in queste zone ed aiuteranno i ricercatori o scienziati. Gli autori *Yajun Lu* e *J. Cecil*, con la pubblicazione del loro articolo [23], hanno proposto un framework basato sull'Internet of Things collaborativo per le aziende manifatturiere. Come sostenuto dall'articolo, questo framework permetterebbe la cooperazione da remoto di utenti/aziende distribuiti/e nelle attività di ingegneria dei processi produttivi. Nell'articolo [24] si parla di *droni* che utilizzano l'Internet of Things collaborativo al fine di raccogliere dati e supportare le attività delle smart cities e degli smart environment, per esempio effettuando dei servizi di tipo meteorologico, monitorare le congestioni del traffico oppure concentrarsi su aspetti ambientali. Un'altra funzionalità offerta da GeoMonitor, che verrà trattata nei capitoli successivi, è il monitoraggio della salute. Anche

in questo caso è possibile reperire numerose app dagli store online con tali funzionalità. Il sistema operativo iOS ha deciso di integrare nei propri iPhone l'applicazione Salute, la quale permette di monitorare giornalmente diversi parametri come il battito cardiaco o il livello di ossigeno nel sangue (attraverso werable quali gli smartwatch)[25].

### 1.5.2 Immuni

Immuni è un'applicazione senza scopo di lucro che notifica gli utenti qualora siano entrati in contatto con persone considerate positive al Covid-19. Infatti, utilizzando il Bluetooth registra la presenza degli smartphone nelle circostanze inviando e ricevendo codici casuali [26]. Purchè funzioni, i dispositivi non devono essere a più di due metri di distanza e rimanere vicini per almeno 15 minuti.

Figura 1.4: Logo Immuni.

## 1.6 Motivazioni

La scelta del realizzare questa applicazione verte sulle tematiche del monitoraggio ambientale e del proprio stato di salute. L'obiettivo è quello di informare l'utente sui valori di certi parametri ambientali rilevati in una determinata zona, magari mettendoli in relazione con le proprie condizioni di salute. Inoltre è possibile essere informati sul numero di nuovi positivi da coronavirus nella propria regione. In particolare i punti di forza consistono nel costruire dei template i quali possono utilizzare sia dati ambientali che dati inerenti alla salute dell'utente, per poi istanziarli in un'area specifica. L'esecuzione delle istanze produrrà in output dei valori sulla base del template costruito e dei dati recuperati in quella zona specifica. Una delle funzionalità più importanti è l'utilizzo del **geo-fencing**, grazie al quale l'utente potrà essere aggiornato sui valori della zona solo nel caso vi si trovi all'interno (supponendo che vi siano i sensori che rilevino i parametri indicati in fase di costruzione del template). GeoMonitor garantisce il rispetto della privacy dei suoi utenti senza limitare le funzionalità dei template istanziati. Infatti, le informazioni sullo stato di salute dell'utente o sulla posizione del dispositivo che verranno raccolte ed utilizzate dallo smartphone non saranno condivise con la rete, perciò rimarranno in locale.

# Capitolo 2

# Architettura

In questa sezione sarà descritta la piattaforma SenSquare e le sue principali funzionalità. Successivamente verranno introdotte le modifiche da me apportate ed i contributi del mio lavoro.

## 2.1 SenSquare

SenSquare è una piattaforma IoT Collaborativa per Smart Cities volta alla raccolta di dati i quali potranno essere utilizzati per comporre servizi utili alla comunità. Il processo di costruzione di questi servizi, che su SenSquare vengono definiti Template, si avvale di un editor che sfrutta la programmazione visiva: Blockly [27]. Un template può essere definito come lo scheletro, o la struttura di un'istanza e determina le "linee guida" che devono essere seguite durante la sua esecuzione. In questa applicazione i template vengono suddivisi in due categorie: **template remoti** e **template locali**. I template remoti sono quelli costruiti sulla piattaforma SenSquare mediante la libreria Blockly.


Figura 2.1: Logo di SenSquare.

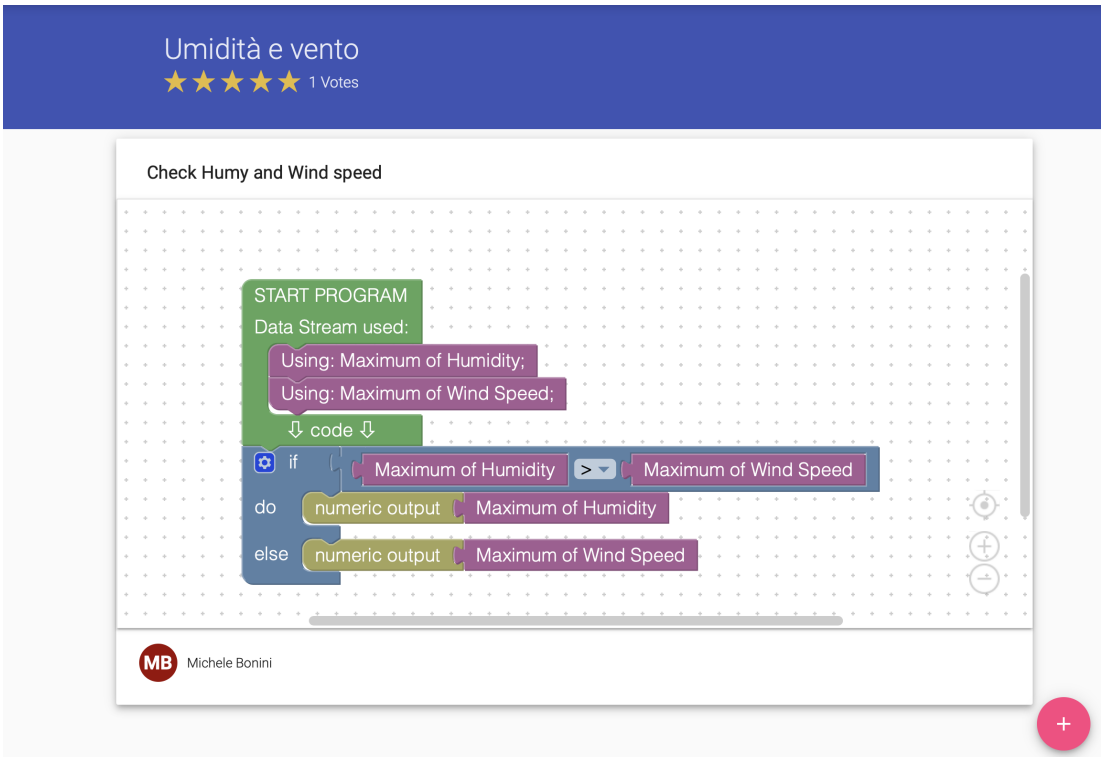



Figura 2.2: Costruzione di un template remoto sulla piattaforma SenSquare

Invece, i template locali, sono costruiti localmente e hanno un potere espressivo minore rispetto a quelli remoti perchè non viene utilizzata nessuna libreria di supporto. Tuttavia, al contrario dei template remoti, quelli locali possono essere formati sia da parametri remoti (per esempio *PM10* o *temperatura*) sia parametri locali come ad esempio *battito cardiaco* o *temperatura corporea*. Un'istanza è il posizionamento del template in una zona specifica sulla mappa. Sono presenti due categorie di istanze: le **istanze remote** e le **istanze locali**. Le istanze remote, ovvero quei template istanziati su SenSquare, permettono di eseguire il template recuperando i valori delle misurazioni sulla base dei parametri presenti nella struttura del template.

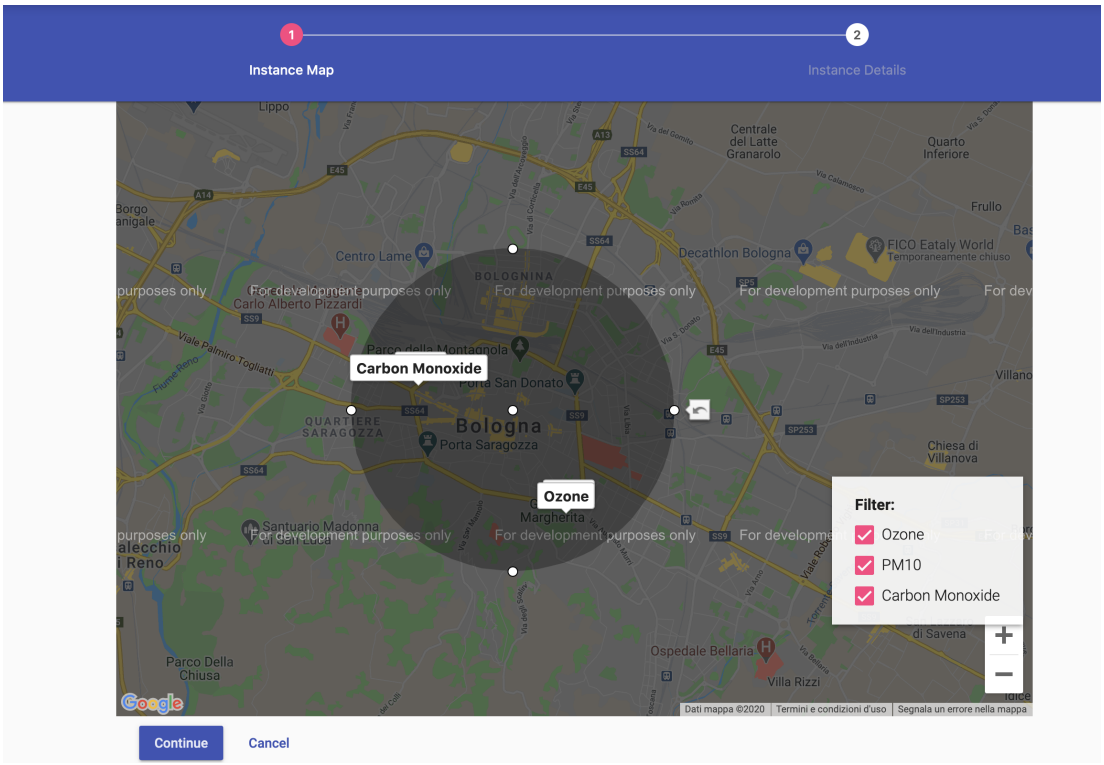



Figura 2.3: Istanziare un template sulla piattaforma SenSquare

Anche le istanze locali, ovvero quei template istanziati localmente, permettono, se attivate, di recuperare i valori delle misurazioni trattati dal template. L'esecuzione dei template remoti può produrre come output il valore di un determinato parametro o valori numerici scollegati dai data class. Quando parliamo di data class ci riferiamo al parametro associato ai valori misurati attraverso i sensori ad esempio i livelli di ozono o di monossido di carbonio rispettivamente indicati con le sigle *O3OZ* e *COPR*. L'esecuzione di template locali, invece, può restituire solo un valore booleano (*vero* o *falso*) a seconda che la condizione espressa dal template sia veritiera o meno.

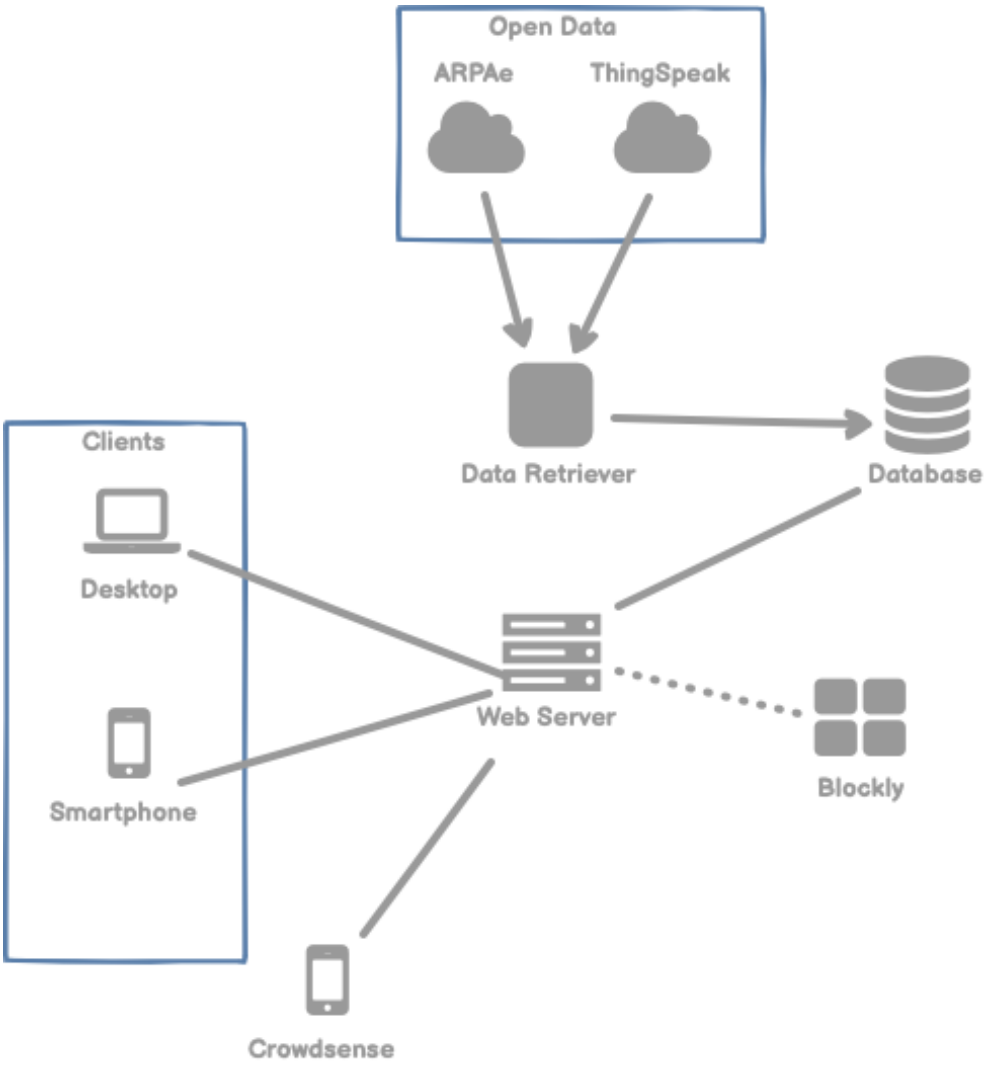



Figura 2.4: Architettura SenSquare.

### 2.1.1 Open Data: dati affidabili e inaffidabili

Una delle potenzialità di SenSquare consiste nell'utilizzo di dati eterogenei tra loro, recuperati periodicamente attraverso il modulo Data Retriever [10]. SenSquare, infatti, sfrutta gli Open Data (dati liberamente accessibili in un formato leggibile) i quali sono suddivisi in due categorie: quelli affidabili, provenienti per esempio da fonti governative come l'Arpae [28], e quelli inaffidabili, prodotti grazie alla collaborazione degli utenti [29]. I dati appartenti alla seconda categoria, possono essere raccolti da diverse piatta-

forme (ad esempio ThingSpeak [30]) e successivamente resi accessibili alla collettività. Tuttavia, sono considerati inaffidabili perchè la loro veridicità non può essere garantita dal momento che provengono dagli utenti.

### 2.1.2 Contributi

Come anticipato precedentemente, il progetto discusso da questa tesi non è altro che un'estensione di SenSquare. In particolare, l'applicazione GeoMonitor, offre la possibilità di visualizzare i template e le istanze presenti sulla piattaforma SenSquare. Tuttavia GeoMonitor permette anche la costruzione di template direttamente dallo smartphone e la loro istanziazione. Dal momento che il processo di costruzione dei template sugli smartphone è stato gestito in maniera diversa da come avviene su SenSquare (per esempio non è stata utilizzata la libreria Blockly), non è possibile comporre template troppo complessi. Ad ogni modo, sia le istanze provenienti da SenSquare sia quelle provenienti dall'istanziazione di template costruiti in locale, possono essere attivate. L'attivazione di tali istanze permetterà all'utente di mandare in esecuzione i template e di visualizzarne il risultato, esattamente come avviene su SenSquare. Tuttavia i risultati verranno prodotti solamente se lo smartphone si troverà all'interno dell'area stabilita dall'istanza. Un altro punto in cui GeoMonitor differisce da SenSquare è quello di permettere all'utente di costruire template che, oltre ad usare i dati provenienti da remoto (*inquinamento*, *umidità* ...), offrano la possibilità di sfruttare dati inerenti alla salute. Ciò è possibile perchè GeoMonitor consente ai suoi utenti di inserire report giornalieri contenenti parametri inerenti al loro stato fisico (per esempio *frequenza cardiaca* e *temperatura corporea*). Ancora, il modulo del *Data Retriever* è stato arricchito in modo tale che potesse recuperare ulteriori dati ambientali da *Open Weather* [31] e dati inerenti ai nuovi positivi da coronavirus dalla *repository GitHub* [32]. Infine, sono state apportate delle modifiche lato server di SenSquare col fine di aggiungere ulteriori API necessarie per reperire determinate informazioni.

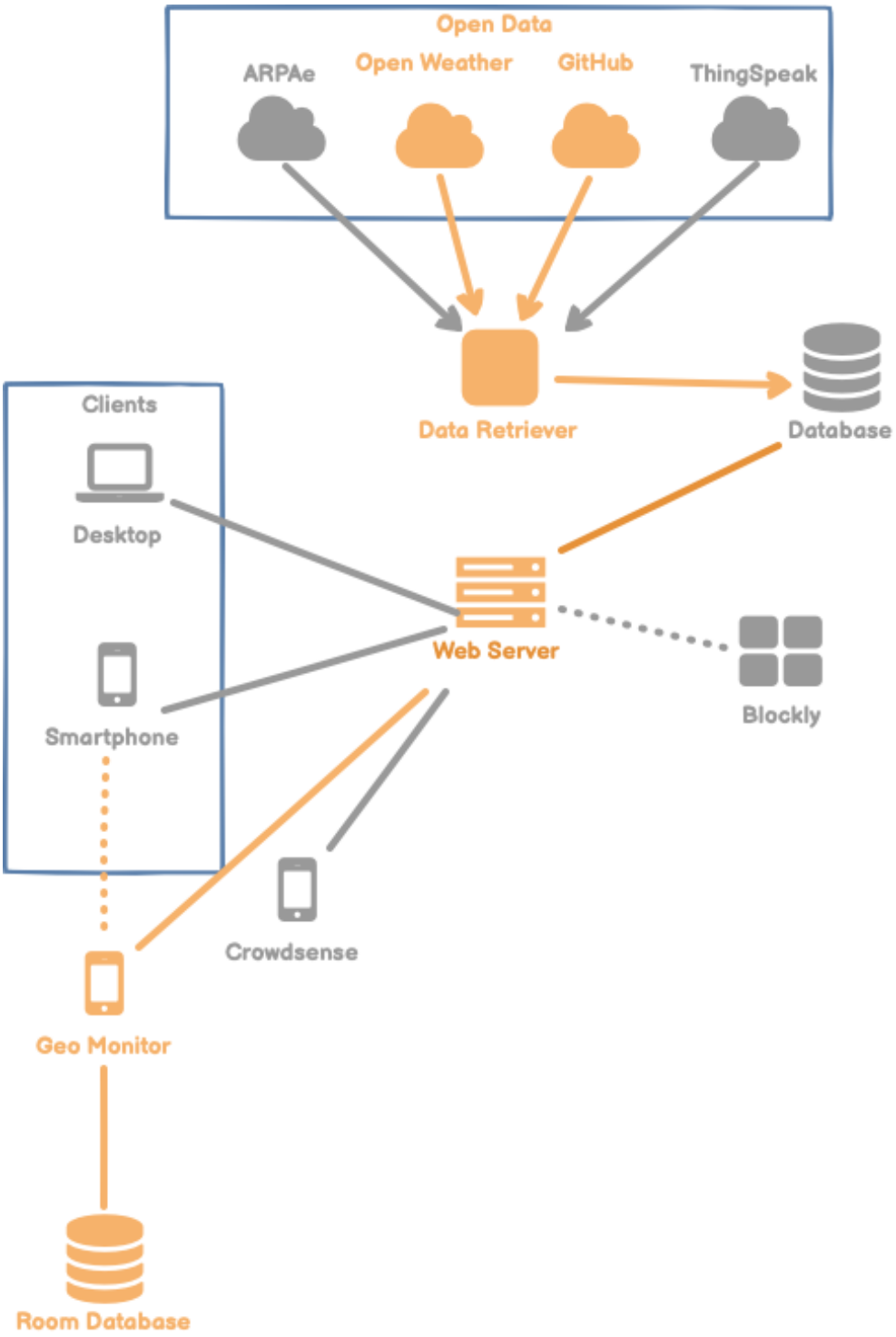



Figura 2.5: Contributi all'architettura SenSquare .

## 2.2 Linguaggi e framework utilizzati

### 2.2.1 Android Studio e Java

Android Studio è un ambiente integrato di sviluppo che permette la costruzione di applicazioni mobili per il sistema operativo Android. Il linguaggio utilizzato per l'applicazione mobile è stato Java. La versione di Android Studio utilizzata è stata la **3.6.1**.

### 2.2.2 Python

Python è un linguaggio di programmazione ad alto livello ed è stato utilizzato per scrivere gli scripts che si occupano di effettuare chiamate API ed inserire i valori ottenuti all'interno del database di SenSquare. Python è considerato un liguaggio performante, al passo coi tempi e al tempo stesso facile da usare. La versione di Python utilizzata è stata la **3.7.3**.

### 2.2.3 Django Rest Framework

Django Rest Framework [33] è un toolkit potente che è stato utilizzato nel progetto originario di SenSquare per la scrittura delle API. Esso permette di restituire i risultati delle query sotto forma di formato JSON. Dal momento che è stato necessario introdurre ulteriori API, in aggiunta a quelle costruite precedentemente su SenSquare, si è preferito mantenere come strategia l'uso di Django Rest Framework.

## 2.3 Funzionalità principali e struttura di GeoMonitor

Nella sezione **Contributi** sono state descritte le principali differenze tra la piattaforma SenSquare e GeoMonitor. Di seguito verranno elencate le funzionalità di maggior rilievo del mio applicativo, le quali saranno dettagliate nel corso dei successivi capitoli. Grazie a questa applicazione è quindi possibile:

- Visualizzare i template e le istanze presenti su SenSquare;

- Visualizzare i template e le istanze costruite localmente;
- Costruire un template con la possibilità di utilizzare sia dati remoti sia dati locali:

  È possibile costruire template locali utilizzando parametri remoti (data class provenienti da SenSquare) oppure locali. In particolare, al click sul pulsante "aggiungi regola SenSquare" verranno aggiunti tre componenti grafici: uno spinner mostrante i data class remoti, uno spinner contenente degli operatori (+, *, /, -, <, >, =) ed un campo in cui inserire un valore numerico. Qualora l'utente cliccasse su "aggiungi regola locale" verranno mostrati gli stessi elementi ma l'unica differenza riguarderebbe solo il primo spinner il quale mostrerà dei data class locali. L'insieme di queste tre componenti è definito "*regola*". Un template può essere formato da $n$ regole le quali vengono aggiunte volta per volta mediante il pulsante "costruisci".

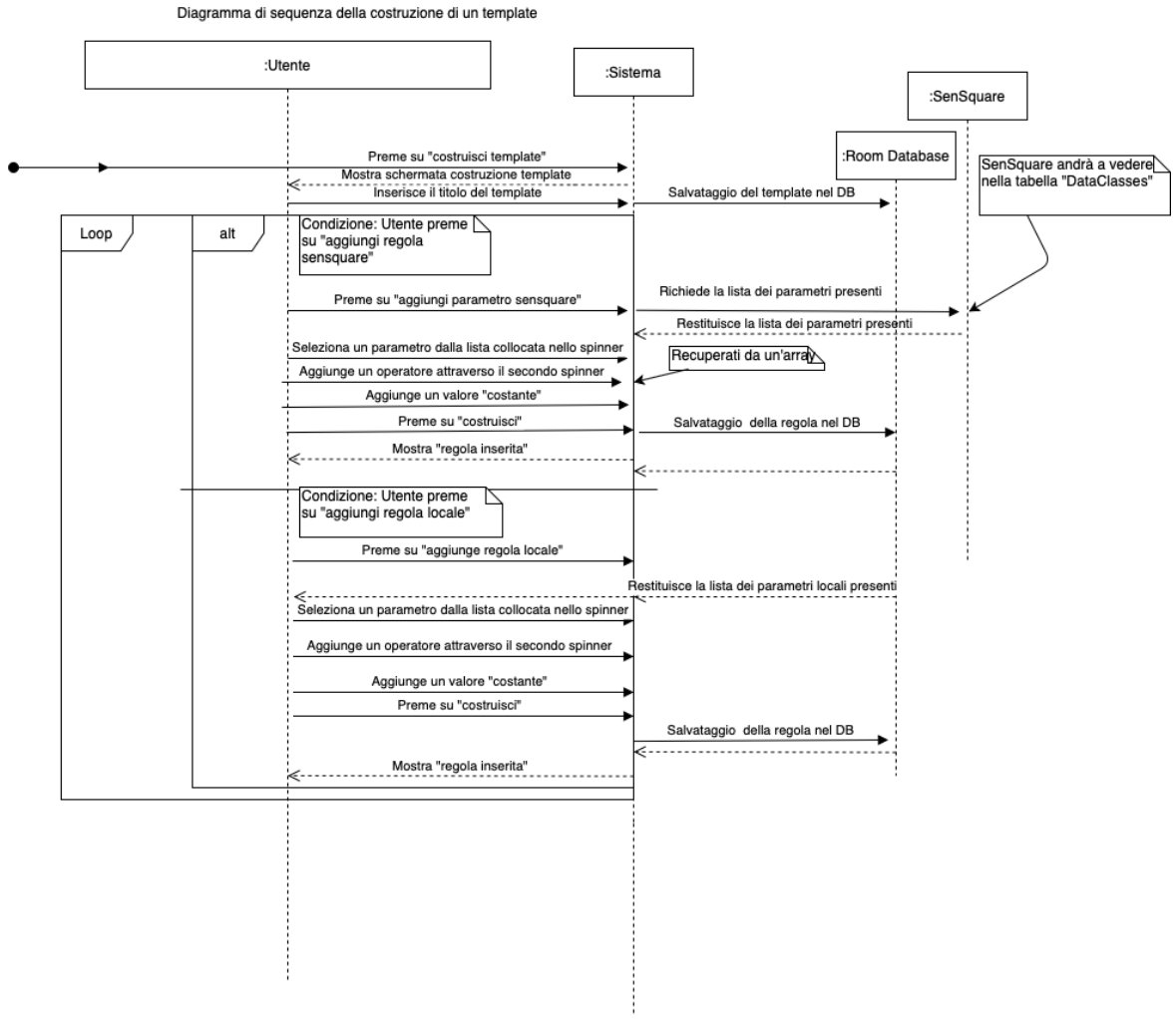



Figura 2.6: Diagramma di sequenza del processo di costruzione di un template locale.

- Istanziare un template remoto o locale in una determinata zona attraverso l'uso di una mappa:

  Quando parliamo di istanziare un template stiamo facendo riferimento a quel processo nel quale, mediante l'uso di una mappa, posizioniamo un'area virtuale circolare in modo tale da capire in quale area geografica l'istanza dovrà produrre dei risultati;

- Attivare un'istanza remota o locale utilizzando la tecnologia del Geo-fencing:

attivare un'istanza significa eseguire il template sul quale si basa l'istanza, solo nel caso in cui l'utente si trovi nell'area alla quale l'istaza medesima fa riferimento. Il processo di attivazione genererà il risultato dei template e restituirà anche i valori utilizzati provenienti dai vari sensori (come sensore di temperatura o di umidità);

- Visualizzare i valori ottenuti dal processo di attivazione delle istanze attraverso una recycler view (cioè una lista scrollabile) ed un grafico collocato nella home dell'app:

  I valori di ciascun parametro (o data class) presenti all'interno del template ed utilizzati nel processo di istanziazione, verranno mostrati all'utente mediante una schermata scrollabile composta da delle celle (definita come recycler view);

- Ricevere notifiche sulla veridicità o meno dei template calcolati;
- Inserire report giornalieri con lo scopo di tener traccia del proprio stato di salute;
- Cercare i report filtrandoli per delle priorità;
- Visualizzare i report attraverso grafici a barre e a torta;
- Ricevere notifiche che ricordino l'inserimento del report giornalmente;
- Modificare l'orario della ricezione delle notifiche attraverso la schermata delle impostazioni.

## 2.4 Privacy e GDPR

Al giorno d'oggi la questione della privacy e del trattamento dei dati personali sta diventando un argomento sempre più importante ed è per questo motivo che GeoMonitor non condividerà con nessuno i dati sensibili raccolti con i report giornalieri inerenti alla salute. Si definiscono dati personali "*le informazioni che identificano o rendono identificabile, direttamente o indirettamente, una persona fisica e che possono fornire informazioni sulle sue caratteristiche, le sue abitudini, il suo stile di vita, le sue relazioni personali, il suo stato di salute, la sua situazione economica, ecc..*" [34]. Essi possono essere suddivisi nelle seguenti categorie:

- Dati identificativi: permettono l'identificazione diretta del soggetto cui i dati si riferiscono;
- Dati anonimi: dati che in origine, o a seguito del trattamento, non possono essere associati a un interessato identificato o identificabile;
- Dati relativi a condanne penali o reati;
- Dati sensibili: dati che riguardano la personalità etico-sociale e le caratteristiche psicologiche come ad esempio dati che rilevano l'origine etnica, lo stato di salute o le opinioni politiche.

La normativa europea in materia della protezione dei è il GDPR (General Data Protection Regulation), il regolamento generale per la protezione dei dati personali entrato in vigore il 24 maggio 2016 ma attuato a partire dal 25 maggio 2018. Il GDPR ha sostituito il "codice della privacy" definito anche codice della protezione dei dati personali, il quale è stata una norma entrata in vigore il 1° Gennaio 2004.

# Capitolo 3

# Le chiamate API

## 3.1 Le chiamate API a SenSquare

Come anticipato precedentemente, si è reso necessario l'aggiunta di nuove API lato server-side di SenSquare, utilizzando il linguaggio Python. Di conseguenza è stata estesa l'interfaccia in grado di rispondere alle richieste API aggiungendo i seguenti URI (dove **/api/v1/** è l'entry point delle API):

- **/api/v1/showDataClasses** permette di recuperare i nomi di tutti i data class presenti su SenSquare;
- **/api/v1/showinst** recupera tutte le istanze presenti su SenSquare;
- **/api/v1/showtemp** recupera tutti i template presenti su SenSquare;
- **/api/v1/streams/nearby/latitude/longitude/radius/classes/** recupera i data stream presenti in una determinata zona e ogni ultima misurazione ad essi associata.

La risposta alle richieste API viene restituita mediante dei files json. Per l'aggiunta di queste nuove API si è dovuto andare a modificare i file: view.py, models, py, serializer.py e urls.py. Per esempio, il vettore CLASSES_WITH _MEASUREMENTS (presente in models.py) contiene tutti i data class i quali sono associati almeno ad una misurazione. Perciò si è resa necessaria l'aggiunta dei nuovi data class (utilizzati da GeoMonitor):

TEMP (temperatura), HUMY (umidità), WISP (velocità del vento), COVD (nuovi positivi covid-19). Un altro esempio lo possiamo riscontrare nel file View.py, all'interno del quale sono state aggiunte le funzionalità utili alla ricerca all'interno del database SenSquare.

```
@api_view(['GET'])
def showinst(request):
    if request.method == 'GET':
        paginator = PageNumberPagination()
        queryset = Customservices.objects.all()
        context = paginator.paginate_queryset(queryset, request)
        serializer = SerializerInstanze(context, many=True)
        return paginator.get_paginated_response(serializer.data)
```

Listing 3.1: Funzione che permette di mostrare tutte le istanze presenti su SenSquare suddivise per pagine.

Per quel che riguarda la view (presente nel file view.py) necessaria a restituire i data streams (e le ultime misurazioni ad essi associate) in una determinata area, era già stata implementata dalla versione originaria di SenSquare. Tuttavia per renderla funzionante ho dovuto modificare il file urls.py aggiungendo la seguente riga di codice.

```
router.register('NearbyDataStreamList', views.NearbyDataStreamList.
    as_view({ 'get': 'list' }), base_name='NearbyDataStreamList')
```

Listing 3.2: Modifiche nel file urls per l'API NearbyDataStreamList.

## 3.2 Ulteriori chiamate API

Come anticipato precedentemente, i valori inerenti alla temperatura, umidità e velocità del vento vengono reperiti da Open Weather, un servizio online che mette a disposizione dati meteorologici. Per utilizzare le sue API, è stato necessario registrarsi al portale web e richiedere la chiave API utile ad effettuare le varie chiamate. Il piano attivato è stato quello gratuito che offre 60 chiamate al minuto e previsioni meteorologiche fino a 5 giorni. Per recuperare i dati da Open Weather e i dati inerenti ai nuovi positivi da coronavirus (da GitHub) ho dovuto scrivere due scripts con il linguaggio Python. Successivamente sono stati mandati in esecuzione periodicamente attraverso l'uso di crontab

[35]. In particolare, i dati di Open Weather vengono recuperati ad ogni ora. Invece, i dati riguardanti i nuovi positivi vengono ottenuti ogni giorno alle ore 5:00 del mattino e segnalano i contagiati del giorno precedente.

### 3.2.1 Gli scripts in Python

Per quanto riguarda i dati di ambientali, essi segnalano le condizioni climatiche della città di Bologna. Si è scelto di utilizzare una sola città onde evitare di riempire eccessivamente il Database di SenSquare. Lo script che si occupa di fare ciò è denominato **scriptAmbiente.py** ed oltre al compito di recuperare i dati ambientali, ha quello di inserirli all'interno del database. Quindi si è reso necessario uno studio in merito a come fosse strutturato il database di SenSquare per capire quali fossero le tabelle interessate. Di seguito verranno elencate le tabelle utili a tale scopo:

- **DataClasses** contiene le informazioni utili alla classificazione di un dato come DataType (esempio WISP) e UnitOfMeasure (per esempio $m/s$);
- **DataStreams** contiene la tipologia di dati e può essere associato ad una o piu Measurements;
- **Measurements** contiene tutti i dati relativi alle misurazioni come ad esempio il valore o le coordinate (latitudine e longitudine).

```
1 url = 'http://api.openweathermap.org/data/2.5/weather?lat={}&lon={}&
    appid={}&units=metric'.format(latitudeLocal,longitudeLocal,apikey)
2 res = requests.get(url)
3 data = res.json()
```

Listing 3.3: Recupero dei dati ambientali attraverso le API di Open Weather.

Prima di inserire il valore delle misurazioni recuperate attraverso le richieste API, lo script creerà il nuovo datastream (nel caso non fosse ancora presente), al quale dovrà poi essere associata la misurazione attraverso l'attributo data_stream_ID collocato all'interno della tabella Measurements .

```
1 if streamsInseritiTemp == 0:
```

```
2     cursor.execute("INSERT INTO DataStreams(device_ID, name, data_class
    , last_entry_ID, reliability, accuracy, update_rate, description) \
3                    VALUES ('%s', '%s', '%s', '%s', '%s', '%s', '%s',
    '%s')" % (citta, stream_name, "TEMP", "-1", "100", "100", "1.0", "
    temperatura"))
```

Listing 3.4: Inserimento di un nuovo datastream qualora non sia ancora presente data una determinata città ed uno specifico dataclass.

Qualora il datastream sia già presente, si dovrà recuperare il suo l'id corrispondente che dovrà poi essere associato alla misurazione corretta.

```
1 if len(ans) == 0:
2     cursor.execute("SELECT ID from DataStreams WHERE (device_ID = '" +
    citta + "' AND data_class = 'TEMP')")
3     stream_ID = cursor.fetchall()[0][0]
4 else:
5     stream_ID = ans[0][0]
6     cursor.execute("INSERT INTO Measurements(data_stream_ID,
    GPS_latitude, GPS_longitude, MGRS_coordinates, value, timestamp,
    Sensor_type) \
7            VALUES ('%s', '%s', '%s', '%s', '%s', '%s', '%s')" % (
    stream_ID, latitudeLocal, longitudeLocal, "-1", temp, today, "OWM"))
8     db.commit()
```

Listing 3.5: Inserimento dei valori della temperatura ottenuti da Open Weather nella tabella Measurements del database di SenSquare.

Lo script incaricato di recuperare quotidianamente le informazioni in merito ai nuovi positivi, per ogni regione italiana, da coronavirus è denominato **scriptCovid.py**. Come per per lo script descritto precedentemente, prima di inserire il valori ottenuti dalla richiesta, dovrà essere verificata la presenza di un datastream, all'interno della tabella DataStreams, riportante il nome della regione associata a quel determinato valore ed il data class indicato con la sigla "COVD", altrimentri lo si dovrà creare.

```
1 if streamsInseritiCovid == 0:
2     cursorCov.execute("INSERT INTO DataStreams(device_ID, name,
    data_class, last_entry_ID, reliability, accuracy, update_rate,
    description) \
```

```
3                 VALUES ('%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s')
  " % (citta_regione, stream_name_cov, "COVD", "-1", "100", "100", "
  1.0", "Totale positivi covid-19"))
4     db.commit();
```

Listing 3.6: Inserimento del datastreams qualora non sia mai stato inserito.

Tuttavia, prima di procedere con l'inserimento del valore all'interno della tabella Measurements, si dovrà verificare che nella giornata di ieri non siano ancora state inserite misurazioni inerenti al covid-19. Infatti ogni giorno dovrà essere inserito al massimo un solo valore per ogni regione d'Italia, dal momento che i dati dei nuovi positivi da coronavirus della giornata precedente non sono variabili.

```
1 for r in data_dati_regioni:
2     if str(yesterday) in str(r):
3         inserisci_nuovi_positivi(lat_pos_reg[i], long_pos_reg[i],
  nuovi_pos[i], denomRegione[i])
4     i = i + 1
```

Listing 3.7: Inserimento dei valori dei nuovi positivi da coronavirus.

# Capitolo 4

# Implementazione Applicazione Mobile

## 4.1 Il database

Il database è stato implementato attraverso l'uso della libreria **Room**, un'astrazione di **SQLite**. Di seguito verranno elencate le tabelle che costituiscono il database:

- *CustomServiceLocal.java* rappresenta i templates che sono stati istanziati in una determinata zona;
- *CustomServiceTemplateLocal.java* tiene traccia dei templates costruiti;
- *Regola.java* contiene la struttura del template;
- *DatoUtente.java* rappresenta i report aggiunti giornalmente dall'utente;
- *DataClassesLocal.java* utile a tenere traccia dei data class locali;
- *IstanzeAttiveSensquare.java* tiene traccia delle istanze remote, provenienti da SenSquare, che sono state attivate;
- *MisurazioniSalute.java* contiene i valori monitorati dall'utente associati al loro rispettivo data class;
- *ValoriIstanze.java* contiene i valori calcolati dalle istanze attive localmente.

Le classi che presentano la parola chiave **@Dao** all'interno del codice (nella directory *Database*), permettono di interrogare la base di dati attraverso query e quindi, per esempio, di inserire o eliminare dati.

```
@Dao
public interface DaoCustomServiceTemplateLocal {
    @Query("select * from custom_service_template_local")
    public List<CustomServiceTemplateLocal> getTemplates();
    @Insert
    public void aggiungiTemplate(CustomServiceTemplateLocal
   templateCostruito);
```

Listing 4.1: *Data Access Object* per la tabella contenente i template costruiti

Le query verso il database non vengono effettuate all'interno del main thread, ma bensì utilizzando gli **AsyncTask**. In questo modo sarà possibile leggere e scrivere sul database in modo asincrono, senza aggiungere troppo lavoro al *main thread*.

```
public class AsyncTaskAggiungiTemplate extends AsyncTask<
   CustomServiceTemplateLocal, Void, Void> {
    @Override
    protected Void doInBackground(CustomServiceTemplateLocal...
   templates) {
        appDatabase.daoCustomServiceTemplateLocal().
   aggiungiTemplate(templates[0]);
        return null;
    }
    @Override
    protected void onPostExecute(Void aVoid) {
        super.onPostExecute(aVoid);
    }
}
```

Listing 4.2: Esempio di un async task che recupera i template contenuti nel database

L'ultimo elemento che collabora al funzionamento del database è *AppDatabase.java*, una classe astratta elencante i vari Database Access Object e accedibile dal codice mediante un oggetto di tipo AppDatabase.

### 4.1.1 Le shared preferences

Alcuni valori sono stati memorizzati nelle shared preferences (visibili solo a livelo di applicazione perchè settati con la modalità privata). Questo è stato utile, ad esempio, per memorizzare il parametro che l'utente ha scelto di monitorare o per capire se egli abbia settato o meno la notifica per ricordargli del report giornaliero.

## 4.2 Costruire un template

Una delle funzionalità principali dell'applicazione consiste nel costruire dei templates. Il template non è altro che lo scheletro (o struttura) dell'istanza che dovrà poi essere posizionata in una zona specifica e attivata. Il template possiede un titolo e un identificativo ed è formato da delle regole (o condizioni). La struttura della regola permette di esprimere le condizioni richieste dall'utente. Essa infatti possiede, come attributo, l'id del template al quale fa riferimento. Vi sono due tipi di regole che possono essere aggiunte: la prima recupera, attraverso una chiamata API, i data class esistenti su SenSquare. La seconda tipologia di regola, invece, recupera i data class inerenti ai parametri inseriti localmente (ad esempio il battito cardiaco o la temperatura corporea). Ciò è stato implementato per permettere all'utente di costruire dei template che potessero mettere insieme valori ottenuti da remoto e quelli locali. Per esempio, data una determinata temperatura corporea ed un livello di umidità in una zona specifica, l'utente potrebbe desiderare di essere notificato. Quindi, all'interno di questa classe vi sono due metodi, **addViewLocal()** e **addViewSensquare()**, incaricati di aggiungere rispettivamente alla scroll view le regole che utilizzano i data class locali e le regole che sfruttano i data class remoti. Ad esempio, il metodo addViewSensquare(), chiamato qualora l'utente abbia premuto sul bottone "aggiungi regola con parametri di sensquare", verifica subito se all'interno della scroll view siano presenti altre regole. Nel caso vi sia già una regola bisognerà far comparire lo spinner che permetterà di collegarla a quelle successive attraverso i seguenti operatori: AND, OR, $=$, $>$ e $<$ posti all'interno del vettore **teamList2** .

```
1 if(regoleInserite < 1){
2     spinnerTeam4.setVisibility(View.GONE);
3 }else{
```

```
4       spinnerTeam4.setVisibility(View.VISIBLE);
5       ArrayAdapter arrayAdapter4 = new ArrayAdapter(this, android.R.
        layout.simple_spinner_item, teamList2);
6       spinnerTeam4.setAdapter(arrayAdapter4);
7   }
```

Listing 4.3: Verifica del numero di regole presenti all'interno della scroll view.

Dalla seconda regola aggiunta è disponibile un ulteriore operatore che permette di collegare le due regole le quali possiedono anche un parametro che ne indica la tipologia, un operatore aritmetico/condizionale( +, -, <, = ...), ed un valore costante. Per procedere con la costruzione del template è necessario riempire tutti i campi di ogni regola inserita. Dopo aver inserito una regola, per confermare l'inserimento, sarà necessario premere sul pulsante "costruisci". Solo successivamente si potrà procedere con l'aggiunta della regola successiva. Tutte le regole costruite, vengono aggiunte dinamicamente ad una scroll view ed è possibile rimuoverle. Per quanto riguarda l'ottenimento dei data class remoti e locali, essi vengono gestiti all'interno del metodo **onCreate()** della classe **SchermataCostruzione.java**. I data class remoti vengono recuperati attraverso una chiamata API a SenSquare, mentre i data class locali dal database Room attraverso un AsyncTask.

```
1  public void avviaAPIDataClasses(String urlPaginaDaMostrare){
2      RequestQueue queue = Volley.newRequestQueue(this);
3      JsonObjectRequest objectRequest = new JsonObjectRequest(
4      Request.Method.GET,urlPaginaDaMostrare,null,
5      new Response.Listener<JSONObject>() {
6          @Override
7          public void onResponse(JSONObject response) {
8                  JSONArray jsonArray = response.getJSONArray("results");
9                  for(int i = 0; i<jsonArray.length(); i++) {
10                     JSONObject results = jsonArray.getJSONObject(i);
11                     String id = results.getString("id");
12                     String dataType = results.getString("data_type");
13                     tipiDiParametro.add(id);
14                 }
15                 String urlPaginaSuccessiva = response.getString("next")
       ;
16                 if(!urlPaginaSuccessiva.equals("")){
```

```
17                  avviaAPIDataClasses(urlPaginaSuccessiva);
18                 }
19         }
20     },
21     new Response.ErrorListener() {
22         ...
23     });
24     queue.add(objectRequest);
25 }
```

Listing 4.4: Chiamata API che recupera i data class presenti su SenSquare.

Come si può notare dal listato 3.4, all'interno della chiamata è stato aggiunto un controllo per verificare che siano presenti ulteriori pagine. Infatti, nel caso l'oggetto **next** restituito con la risposta in formato json non sia vuoto, verrà richiamato il metodo **avviaAPIDataClasses** permettendo una nuova richiesta all'url contenuto dall'oggetto.

## 4.3 Istanziare un template

### 4.3.1 Utilizzo di Google Maps

Dopo aver completato con successo la costruzione del template, sarà possibile istanziarlo in una determinata zona. La classe dedicata a tale processo è denominata **MappaTuoiTemplate.java**. Essa permetterà di utilizzare le mappe di **GoogleMaps**. Tuttavia per implementare ciò è stato necessario compiere le seguenti azioni:

- Installare i Google Play services [36];
- Registrarsi al sito **Google Cloud Plataform**;
- Creare un progetto ed abilitare **Maps SDK per Android**;
- Richiedere la **chiave API**, utile ad effettuare le richieste;
- Creare un fragment configurato per utilizzare la mappa (MappaTuoiTemplate.java);
- Inserire la chiave API all'interno del **Manifest.xml**;

- Implementare la richiesta dei permessi per utilizzare la posizione dell'utente.

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

Listing 4.5: Configurazione della chiave API per utilizzare Google Maps.

### 4.3.2 I permessi

I permessi di geolocalizzazione devono per prima cosa essere dichiarati nel Manifest.xml.

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

Listing 4.6: Dichiarazione dei permessi nel file Manifes.xml

Successivamente è stato implementato il codice per la richiesta di tali permessi. In particolare appena la mappa sarà caricata, verrà mostrato un dialog per consentire all'utente di sfruttare la localizzazione.

### 4.3.3 Il processo di istanziazione

La classe MappaTuoiTemplate.java riceve, attraverso dei metodi **putExtra** (una delle opzioni che permettono il passaggio di valori tra le activity), l'id del template in oggetto, e tre stringhe: la struttura del template, i data class locali e quelli remoti presenti all'interno del template. Questi valori oltre ad essere utilizzati all'interno di questa classe, verranno inseriti nella tabella delle istanze del database, se l'istanziazione andrà a buon fine. Per poter istanziare un template è necessario inserire il titolo e la descrizione negli appositi campi. Successivamente, tenendo premuto su un punto della mappa, sarà possibile posizionare un confine virtuale circolare. Dal punto di vista implementativo, viene

fatto l'override di un metodo denominato **onMapLongClick()** il quale prende in input un oggetto di tipo **LatLng** rappresentante le coordinate della latitudine e longitudine. Esso, tra i primi compiti da svolgere, ha quello di posizionare un'area circolare nella zona determinata dall'utente. Ciò verrà fatto attraverso il metodo **handleMapLongClick()** il quale prenderà in input un oggetto rappresentante le due coordinate e, successivamente, posizionerà il cerchio virtuale sulla mappa con la funzione **addCircle()**.

```
1  private void handleMapLongClick(LatLng latLng){
2      mMap.clear();
3      addMarker(latLng);
4      addCircle(latLng, raggioInstanza);
5  }
6  private void addMarker(LatLng latLng){
7          MarkerOptions markerOptions = new MarkerOptions().position(
       latLng);
8          mMap.addMarker(markerOptions);
9  }
10 private void addCircle(LatLng latLng, double radius){
11     CircleOptions circleOptions = new CircleOptions();
12     circleOptions.center(latLng);        circleOptions.radius(radius);
13     circleOptions.strokeColor(Color.argb(255, 0, 128, 0));
14     circleOptions.fillColor(Color.argb(64, 0, 128, 0));
15     circleOptions.strokeWidth(4);
16     mMap.addCircle(circleOptions);
17 }
```

Listing 4.7: Aggiunta sulla mappa dell'confine virtuale e del markers indicante il centro.

Dopodichè l'onMapLongClick() collocherà, attraverso dei markers che verranno posti sulla mappa, la posizione dei data stream presenti in quella zona. Questo viene fatto perchè nel caso non vi siano datastream, l'utente non sarà in grado di istanziare i template. Infatti non sarà possibile istanziare il template qualora, all'interno dell'area posizionata, non vi sia almeno un data stream per ogni tipologia di parametro (data class) presente all'interno della struttura del template, fatta eccezione per i parametri locali. Quindi, all'interno di **onMapLongClick()**, la stringa recuperata attraverso il metodo **getExtra** (come anticipato precedentemente), viene splittata e trasformata in un vettore denominato **data_class**. In seguito verrà inizializzato l'url il quale sarà utilizzato per effettuare

la richiesta API a SenSquare. La risposta che verrà restituita conterrà la lista dei data stream presenti all'interno di quelle coordinate.

```
1    url = "http://sensquare.disi.unibo.it/api/v1/streams/nearby/"+
   latitudineCliccata+"/"+longitudineCliccata+"/"+raggioInstanza+"/"+
   data_class[0]+"/";
```

Listing 4.8: Assegnazione dell'url (formato dai vari attributi neccessari alla richiesta API) alla variabile denominata url.

Il listato 3.5 mostra la struttura dell'url. Come è possibile notare esso utilizza la latitudine e la longitudine sulle quali si è cliccato, il raggio dell'area inserito, ed il data class nella prima posizione. Il raggio del confine circolare si potrà incrementare o diminuire a piacimento mediante il bottone "aumenta il raggio" presente nella parte alta della schermata. Successivamente, sempre all'interno di **onMapLongClick()**, verrà richiamata l'API attraverso il metodo **mostraUlterioriPagine()** il quale posizionerà un marker utilizzando le coordinate delle misurazioni per ogni datastreams ottenuto. Infatti, la risposta in formato JSON oltre a prevedere la restituzione di tutti i data stream, restituirà anche le ultime misurazioni ad essi associate.

```
GET /api/v1/streams/nearby/44.4937544/11.3409058/5000/TEMP/
```

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "ok": true,
    "page_count": 1,
    "count": 1,
    "results": [
        {
            "id": 28287,
            "name": "Bolognacittà",
            "data_class": "TEMP",
            "description": "temperatura",
            "reliability": 100,
            "accuracy": 100,
            "measurement": {
                "id": 20776,
                "data_stream_id": 28287,
                "gps_latitude": 44.4937544,
                "gps_longitude": 11.3409058,
                "mgrs_coordinates": "-1",
                "value": 11.33,
                "timestamp": "2020-11-11T21:00:03Z"
            }
        }
    ],
    "next": null,
    "previous": null
}
```

Figura 4.1: Esempio di una possibile risposta alla chiamata api richiedente i datastreams in una determinata zona.

Prima che l'esecuzione di mostraUlterioriPagine() termini verrà verificato che all'interno del vettore **data_class** non vi siano più valori. Infatti, bisognerà rieseguire il metodo (ovvero la chiamata API) per ogni singolo dataclass contenuto nel vettore.

```
1 indiceDataClassSuccessivo ++;
2 if(indiceDataClassSuccessivo<data_class.length) {
```

```
    String urlNuovo = "http://sensquare.disi.unibo.it/api/v1/streams/
    nearby/" + latitudineCliccata + "/" + longitudineCliccata + "/" +
    raggioInstanza + "/" + data_class[indiceDataClassSuccessivo] + "/";
    mostraUlterioriPagine(urlNuovo);
}
```

Listing 4.9: Verifica che non vi siano più data class.

Per quanto riguarda invece l'istanziazione di template che contengono data class di tipo locali (per esempio quelli inerenti al battito cardiaco o alla temperatura corporea) bisognerà controllare che all'interno del database siano presenti almeno un valore per ognuno dei data class trattati. Non sarà quindi possibile istanziare un template che consideri anche i valori del battito cardiaco nell'eventualità che non vi siano stati inseriti report contenenti tale parametro. Come detto precedentemente per i data class remoti, viene recuperata, attraverso il metodo getExtra(), anche la stringa contenente i dataclass locali (se effettivamente presenti nella struttura del template) successivamente splittata e trasformata in un array denominato **parLocali**. Al click sul pulsante utile ad istanziare il template verrà verificato che tale vettore non sia vuoto. Per ognuno dei valori presenti all'interno del vettore, verrà richiamato l'AsyncTask **AsyncTaskGetMisurazioniSalute()** il quale si occuperà di verificare l'esistenza dei data class trattati dal template all'interno della tabella riguardante i valori della salute. Supponendo che tutti i controlli vengano superati con successo, il processo di istanziazione terminerà con la creazione di un oggetto di tipo **CustomServiceLocal**.

```
CustomServiceLocal instanzaCreata = new CustomServiceLocal();
instanzaCreata.setDeployment_radius(raggioInstanza);
instanzaCreata.setGps_latitude(latitudineCliccata);
instanzaCreata.setGps_longitude(longitudineCliccata);
instanzaCreata.setDescrizione(noteTuoTemplate.getText().toString());
instanzaCreata.setTitolo(titoloInstanza.getText().toString());
instanzaCreata.setArgs(templateId);
instanzaCreata.setStruttura_template(strutturaTemplate);
instanzaCreata.setStatoIstanza("Non Attiva");
instanzaCreata.setId_data_stream_utili(idStreams);
```

Listing 4.10: Creazione di un oggetto di tipo CustomServiceLocal.

Come è possibile notare, all'interno della tabella rappresentante le istanze (CustomServiceLocal), verranno inserite anche la latitudine e la longitudine del centro dell'area posizionata ed il suo raggio. Questi attributi verranno recuperati in seguito per la costruzione del geo-fence. Un controllo verifica la presenza o meno di parametri locali e li setta all'oggetto rappresentante l'istanza.

```
instanzaCreata.setNomiParametriLocali(data_class_trattati_localmente);
```

Listing 4.11: Inserimento dei data class locali se trattati nella struttura del template.

Dopo aver settato tutti i suoi attributi l'istanza verrà inserita nel database tramite il metodo **execute()** dell'AsyncTask **AsyncTaskInserisciInstanza()** il quale accetterà come parametro l'oggetto istanza creato.

```
public class AsyncTaskInserisciInstanza extends AsyncTask<
    CustomServiceLocal, Void, Void> {
        @Override
        protected Void doInBackground(CustomServiceLocal... instanzas){
            appDatabase.daoCustomServiceLocal().aggiungiInstanza(
    instanzas[0]);
            return null;
        }
        @Override
        protected void onPostExecute(Void aVoid) {
            super.onPostExecute(aVoid);
            mMap.clear();
            noteTuoTemplate.setText("");
            titoloInstanza.setText("");
```

Listing 4.12: Inserimento di un'istanza nel database.

## 4.4 Attivare un'istanza

Come sostenuto precedentemente, è possibile attivare sia istanze provenienti da SenSquare (remote) sia istanze prodotte in locale attraverso le rispettive classi di **MapsActivity.java** e **MappaTuaIstanza.java**. L'attivazione di un'istanza consiste nell'esecuzione del template sul quale essa si basa permettendo all'utente di essere notificato sui valori calcolati, e comporta l'utilizzo delle seguenti componenti:

- Una mappa per monitorare costantemente la posizione dell'utente;
- Un service per continuare a monitore la posizione con l'applicazione in background;
- L'area geo-fence coincidente con il confine circolare impostato in fase di istanziazione;
- Una recycler view nella quale collocare i risultati calcolati;
- Un **broadcast receiver**, componente in ascolto che svolgerà determinate funzioni quando chiamato.

### 4.4.1 Istanze remote e locali

La classe incaricata di gestire le attivazioni di istanze remote è MapsActivity.java. Appena arrivati su questa activity, nel metodo **onStart()**, verrà fatto partire il service il quale avrà il compito di monitorare la posizione in background. Successivamente subentrerà l'esecuzione dell'AsyncTask utile a recuperare lo stato dal database dell'istanza, se presente. Infatti è stata predisposta una apposita tabella la quale conterrà tutte le istanze provenienti da SenSquare che sono state attivate. Qualora venissero disattivate, verranno rimosse dal database. Il recupero dello stato servirà ad impostare il testo del bottone presente subito sotto la mappa. Se lo stato dell'istanza fosse attivo, il bottone mostrerà il testo "disattiva istanza". Viceversa, se lo stato non fosse attivo, il bottone conterrà "attiva istanza". Nel caso in cui l'istanza non fosse ancora stata attivata, al click sul pulsante "attiva istanza" verrà mandato in esecuzione l'AsyncTask **AsyncTaskInserisciInstanzaSensquare** incaricato di aggiungere un'istanza al database con lo stato "attivo".

```
1 public class AsyncTaskInserisciInstanzaSensquare extends AsyncTask<
    IstanzeAttiveSensquare, Void, Void> {
2     @Override
3     protected Void doInBackground(IstanzeAttiveSensquare...
    istanzeAttiveSensquares) {
4         appDatabase.daoIstanzeAttiveSensquare().
    aggiungiInstanzaSensquare(istanzeAttiveSensquares[0]);
5         return null;
```

```
6       }
7       @Override
8       protected void onPostExecute(Void aVoid) {
9           super.onPostExecute(aVoid);
10          Log.d(TAG, "onPostExecute: Istanza Aggiunta");
11          addGeofence(locationInstanza, raggioInstanza);
12      }
13  }
```

Listing 4.13: Inserimento dell'istanza proveniente da SenSquare con stato attivo nel database.

Come è possibile notare dal listato qui sopra, una volta aggiunta l'istanza al database, verrà chiamato il metodo **addGeofence()** (il suo funzionamento è stato descritto nella sezione 4.5.1). La classe MappaTuaIstanza.java ha un funzionamento simile a quello di MapsActivity.java. Tuttavia differisce per alcuni aspetti importanti, quale la gestione delle notifiche, il broadcast receiver utilizzato e la gestione dello stato. Infatti in questa activity non si utilizza una tabella contenente gli stati attivi o non attivi delle istanze. Dal momento che esisteva già una tabella nel database rappresentate le istanze create localmente, si è pensato di aggiungere un attributo che indicasse lo stato di tali oggetti. Infatti, come si sarà potuto notare precedentemente, in fase di creazione dell'istanza viene settato immediatamente lo stato come "non attivo". Come prima, al click sul pulsante presente nell'activity MappaTuaIstanza, verrà controllato il valore dello stato nella tabella CustomServiceLocal. Invece, per quanto riguarda le notifiche ed i broadcast, verranno dettagliati nelle sezioni successive.

## 4.5 Implementazione del Geo-fencing

In questo progetto il Geo-fencing è stato utilizzato per calcolare il risulato dei template e recuperare i valori dei parametri trattati, solamente nel caso in cui l'utente sia entrato all'interno dell'area stabilita dall'istanziazione del template.

### 4.5.1 Aggiungere un geofence

Per utilizzare i geo-fencing, all'interno dell'onCreate(), sono stati istanziati due oggetti: uno di tipo GeofencingClient e l'altro di tipo GeofenceHelper, senza i quali sarebbe stato impossibile utilizzare tale funzionalità. Quando l'utente deciderà di attivare l'istanza, verrà eseguito il metodo **addGeofence()** il quale prenderà in input le coordinate del centro ed il raggio dell'area circolare. Per prima cosa si chiamerà il metodo **getGeofence()**, posto all'interno della classe GeofenceHelper, il quale avrà il compito di costruire il geofence utilizzando i seguenti metodi:

- setCircularRegion(): prende in input latitudine, longitudine e raggio;
- setRequestId(): prende in input l'id del geo-fence da costruire;
- setTransitionTypes(): imposta il tipo di transizioni che in questo caso sono *enter*, *dwell* e *exit*;
- setLoiteringDelay(): indica il tempo dopo il quale verrà considerata la transizione dwell (ovvero potrebbe essere mandata una notifica all'utente nel caso fosse dentro la zona da almeno 5 minuti);
- setExpirationDuration(): per indicare per quanto tempo il geo-fence rimarrà attivo.

```
1 public Geofence getGeofence(String ID, LatLng latLng, float radius, int
      transitionTypes){
2     return new Geofence.Builder()
3         .setCircularRegion(latLng.latitude, latLng.longitude, radius)
4         .setRequestId(ID)
5         .setTransitionTypes(transitionTypes)
6         .setLoiteringDelay(10000)
7         .setExpirationDuration(Geofence.NEVER_EXPIRE)
8         .build();
9 }
```

Listing 4.14: Costruzione del geofence.

Successivamente verrà preparato un pendingIntent utilizzando il metodo **getPendingIntent()** della classe GeofenceHelper. All'interno di esso verrà inizializzato un oggetto

di tipo Intent indicando il Broadcast che dovrà essere richiamato quando l'utente sarà entrato nell'area.

```
public PendingIntent getPendingIntent(int idInstanza){
    if(pendingIntent != null){
        return pendingIntent;
    }
    Intent intent = new Intent(this, GeofenceBroadcastReceiver.class);
    intent.putExtra("idInstanza", idInstanza);
    pendingIntent = PendingIntent.getBroadcast(this, 2607, intent,
   PendingIntent.FLAG_UPDATE_CURRENT);
    return pendingIntent;
}
```

Listing 4.15: Viene preparato un pending intent il quale dovrà risvegliare il broadcast indicato.

Tornando alla classe MapsActivity, sarà il turno del metodo **addGeofences()** il quale avrà il compito di risvegliare il broadcast, non appena l'utente entrerà nell'area, attraverso il passaggio del pending intent come attributo.

```
geofencingClient.addGeofences(geofencingRequest, pendingIntent)
    .addOnSuccessListener(new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            Log.d(TAG, "onSuccess: Geofence aggiunto...");
        }
    })
    .addOnFailureListener(new OnFailureListener() {
        ...
    });
```

Listing 4.16: Aggiunta del geofence all'interno del metodo addGeofence della classe MapsActivity.

### 4.5.2 Rimuovere un geo-fence

Entrambe le classi MapsActivity.java e MappaTuaIstanza.java dispongono di un metodo necessario alla rimozione dei geo-fences. Infatti, supponendo che l'istanza sia stata

attivata, sarà possibile disattivarla premendo sul pulsante "disattiva istanza". La disattivazione di un'istanza non comporterà la distruzione del service utile a monitorare la posizione in background. Essa rimuoverà soltanto il geo-fence dalla mappa e fermerà l'esecuzione del timer presente all'interno del broadcast receiver (il quale verrà dettagliato nelle sezioni successive). Dal punto di vista implementativo, all'interno delle due activity è presente un AsyncTask dedicato a: rimuovere l'istanza dal database nel caso in cui l'istanza in oggetto sia remota oppure aggiornare lo stato dell'istanza a "non attiva" qualora essa sia locale. Per esempio, in MappaTuaIstanza, per prima cosa si procederà a rimuovere il geo-fence attraverso il metodo **removeGeofence()** e successivamente si richiamerà l'AsyncTask **AsyncTaskAggiornaStatoIstanzaANonAttiva()** il quale aggiornerà lo stato nel database.

```
1  private void removeGeofence(){
2      geofencingClient.removeGeofences(geofenceHelperTuaIstanza.
       getPendingIntent(idInstanza))
3          .addOnSuccessListener(this, new OnSuccessListener<Void>() {
4              @Override
5              public void onSuccess(Void aVoid) {
6                  Log.d(TAG, "onSuccess: geofence rimosso!");
7                  rimuoviGeofence.setText("Attiva Istanza");
8              }
9          })
10         .addOnFailureListener(this, new OnFailureListener() {
11             ...
12         });
13 }
```

Listing 4.17: Metodo removeGeofence().

Quindi l'istanza locale verrà aggiornata a "non attiva" come mostrato dal listato sottostante.

```
1  public class AsyncTaskAggiornaStatoIstanzaANonAttiva extends AsyncTask<
       Integer, Void, Void> {
2      @Override
3      protected Void doInBackground(Integer... integers) {
4          appDatabase.daoCustomServiceLocal().aggiornaStatoIstanza("Non
       Attiva", integers[0]);
```

```
5           return null;
6       }
7       @Override
8       protected void onPostExecute(Void aVoid) {
9           super.onPostExecute(aVoid);
10          Log.d(TAG, "Stato istanza aggiornato a Non Attiva");
```

Listing 4.18: Aggiornamento dello stato dell'istanza locale a "non attiva" in MappaTuaIstanza.java .

Invece, l'istanza remota verrà rimossa dal database delle istanze (remote) attive.

## 4.6 I broadcast receiver

Il broadcast receiver è una componente che resta in ascolto e attende di essere chiamato. In questo caso sono stati utilizzati due broadcast: GeofenceBroadcastReceiver.java e BroadcastGeofenceTuaIstanza.java. Essi svolgono funzioni differenti, per esempio il BroadcastGeofenceTuaIstanza, utilizzato dalle istanze locali, tra i vari compiti ha anche quello di calcolare il risultato del template locale utilizzando i valori recuperati da SenSquare e/o quelli inerenti allo stato di salute. Al contrario GeofenceBroadcastReceiver, utilizzato in fase di attivazione di istanze remote, non dovrà calcolare il template perchè questa funzionalità è stata delegata al server online (dal momento che il processo di creazione è differente per via della libreria Blockly utilizzata).

### 4.6.1 Il broadcast receiver delle istanze remote

Il broadcast GeofenceBroadcastReceiver viene risvegliato appena l'utente entra nell'area geo-fence. Tra le sue funzionalità principali ritroviamo:

- Individuare il tipo di transizione: l'utente entra nell'area o esce dall'area;
- La possibilità di essere ri-eseguito finchè l'utente si trova all'interno dell'area;
- Fare una richiesta API che permetta di eseguire sul server l'istanza in oggetto e di calcolarne il risultato;

- Mostrare il risultato dell'esecuzione del template attraverso una notifica;
- Mostrare in una recycler view i valori utilizzati dai data class presenti all'interno del template;
- Notificare l'utente nel caso in cui esca dall'area per poi essere successivamente fermato.

Il metodo **onReceive()** è il primo ad essere eseguito non appena il brodcast verrà chiamato. Al suo interno viene controllato il tipo di transizione che è avvenuta.

```
final int transitionType = geofencingEvent.getGeofenceTransition();
```

Listing 4.19: Riconoscimento del tipo di transizione entranti o uscenti dall'area geofence.

Successivamente attraverso uno *switch-case* si avranno diversi comportamenti sulla base delle entrate/uscite dall'area geo-fence.

```
switch (transitionType){
    case Geofence.GEOFENCE_TRANSITION_ENTER:
        ...
    case Geofence.GEOFENCE_TRANSITION_EXIT:
        ...
}
```

Listing 4.20: Transizione entrante nell'area

Nel caso l'utente entri nell'area geo-fence, attraverso il metodo getExtra(), verrà recuperato l'id dell'istanza in questione e verrà fatto partire un timer. Il timer servirà per ripetere la chiamata API ogni $n$ minuti, (il numero di minuti viene impostato dall'utente nella schermata precedente l'activity MapsActivity.java). All'interno del timer viene controllato che il service per il monitoraggio della posizione in background sia ancora attivo, qualora non lo fosse il timer verrà fermato insieme all'esecuzione del broadcast. Al contrario, nel caso in cui il service fosse ancora attivo, verrà eseguito l'AsyncTask **AsyncTaskGetStatoIstanzaSens2()** che recupererà lo stato dell'istanza. Infatti, solo se l'istanza sarà attiva verrà fatta la chiamata API a SenSquare altrimenti anche in questo caso verrà richiamato il metodo cancel() il quale terminerà l'esecuzione del timer. Il metodo che si occupa di effettuare la chiamata API a SenSquare è denominato

**recuperaValori()**. Per prima cosa, al suo interno viene inizializzato l'url nel modo seguente:

```
String url ="http://sensquare.disi.unibo.it/api/v1/instances/"+
    idInstanza+"/execute/";
```

Listing 4.21: Inizializzazione dell'url necessario alla chiamata API.

La risposta in formato json che verrà restituita, qualora la chiamata vada a buon fine, conterrà un oggetto value il quale indicherà il risultato dell'esecuzione del template. Di conseguenza è stato necessario recuperare il contenuto di tale oggetto e salvarlo in una variabile denominata *valoreCalcolato*.

```
valoreCalcolato = response.getDouble("value");
```

Listing 4.22: Recupero del risultato calcolato da SenSquare in merito all'esecuzione dell'istanza.

Questa variabile verrà poi passata come parametro al metodo incaricato di inviare la notifica.

```
notificationHelper.sendHighPriorityNotification("Valore calcolato", ""+
    valoreCalcolato, MapsActivity.class);
```

Listing 4.23: Costruzione della notifica da inviare.

La chiamata API permette anche di recuperare i valori dei data class utilizzati nell'esecuzione dell'istanza. Quindi, sempre all'interno di recuperaValori(), questi valori sono stati recuperati, salvati in un ArrayList e inviati alla classe MapsActivity la quale costruirà una recycler view mostrandoli all'utente. Inoltre, all'interno di MapsActivity è presente un metodo denominato **setLivello()** il quale controllerà i valori ricevuti dal broadcast e li classificherà sulla base della loro pericolosità. Per esempio, se il PM10 fosse maggiore di 50 allora il livello verrà indicato come *critico*. La classificazione dei livelli è stata effettuata mediante i valori riportati sul sito dell'ARPAe [28]. Quando l'utente uscirà dall'area geo-fence si ricadrà nella transizione di uscita e, oltre a fermare l'esecuzione del timer, verrà inviata una notifica comunicante tale uscita.

```
notificationHelper.sendHighPriorityNotification("SEI USCITO DALL'AREA
    DELL'INSTANZA", "Sei uscito dall'area monitorata", MapsActivity.
    class);
```

Listing 4.24: Notifica inerente all'uscita dall'area geo-fence.

### 4.6.2 Il broadcast receiver delle istanze locali

Il broadcast utilizzato dalle istanze locali è denominato BroadcastGeofenceTuaIstanza. Tra le sue funzionalità ritroviamo:

- Individuare il tipo di transizione: l'utente entra nell'area o esce dall'area;
- La possibilità di essere ri-eseguito finchè l'utente si trova all'interno dell'area;
- Il recupero dei valori dei data class sia remoti e sia locali che possono essere trattati dal template;
- L'esecuzione dell'istanza del template costruito in locale;
- Inviare una notifica solamente nel caso la condizione espressa dal template sia veritiera;
- Mostrare in una recycler view i valori utilizzati dai data class presenti all'interno del template;
- Notificare l'utente nel caso in cui esca dall'area per poi essere successivamente fermato.

Il funzionamento del metodo onReceive() del broadcast è analago a quello descritto nel caso delle istanze remote. Di conseguenza anche in questo caso si utilizzerà uno *switch-case*, per compiere determinate azioni sulla base del tipo di transizione avvenuta, ed il timer. All'interno del timer verrà eseguito l'AsyncTask **AsyncTaskGetStatoIstanza2()** il quale verificherà che l'istanza sia attiva e in tal caso chiamerà l'AsyncTask **AsyncTaskGetIstanzaFiltrata()** incaricato di recuperare le informazioni relative all'istanza in oggetto. Recuperate tali informazioni, verrà richiamato il metodo **ottieniUltimaMisurazione()** il quale prenderà in input l'url necessario alla chiamata per ottenere le

ultime misurazioni. In questo caso l'API utilizzata è quella che permette la visualizzazione dei datastream sulla mappa in fase di istanziazione del template, dal momento che essa possiede le informazioni relative alle ultime misurazioni recuperate. Una volta terminato il recupero delle misurazioni, ci sarà un controllo il quale verifcherà che la struttura del template sia formata da data class remoti o se all'interno vi siano anche data class locali. Nel caso non ci siano data class locali, verranno chiamati i metodi **preparaLista()** e **caricaRegole()**. Il primo avrà il compito di inizializzare degli ArrayList utili per calcolare l'istanza, mentre il secondo si occuperà di reperire le regole, trattate dal template in oggetto, dal database. L'AsyncTask incaricato di fare ciò è denominato **AsyncTaskGetRegoleFiltrate()** e le regole recuperate verranno collocate all'interno di un ArrayList con nome: *listaRegole*. Nel caso in cui il template contenga anche data class locali, dovranno essere recuperati i valori ad essi associati. Per fare ciò verrà utilizzato l'AsyncTask **AsyncTaskGetValoreSalute()** il quale collocherà in un'ArrayList denominato *misurazioniSalute* l'ultima misurazione per ogni parametro. Dopodichè verranno chiamati i metodi utilizzati nel caso in cui l'istanza avesse trattato solo data class remoti, vale a dire **preparaLista()** e **caricaRegole()**. Successivamente si potrà chiamare il metodo *calcolaValoreTemplate()* il quale, utilizzando le regole recuperate e i valori delle misurazioni, calcolerà il risultato dell'istanza del template. All'interno di questo processo verrà utilizzato un vettore di appoggio nel quale verranno inseriti degli 0 o degli 1 a seconda che le condizioni espresse dalle regole siano veritiere o meno. La notifica comunicante la veridicità del template verrà inviata solamente nel caso in cui questo vettore contenga solo degli 1.

```
for(int i = 0; i<listaDiRisultati.size(); i++){
    if(listaDiRisultati.get(i) == 0){
        lanciaNotifica = false;
        break;
    }else{
        lanciaNotifica = true;
    }
}
```

Listing 4.25: Verifica del vettore di appoggio.

Successivamente verrà preparato un vettore contenente le indicazioni inerenti la criticità o meno dei valori calcolati per poi passare all'activity, incaricata di mostrare i valori, gli ArrayList utili alla costruzione della recyler view. Per fare ciò, dal broadcast, verrà chiamato il metodo **updateUI()**, appartenente a MappaTuaIstanza, con lo scopo di aggiornare la recycler view e mostrare i valori. Anche in questo caso, il timer si stopperà ed il brodcast terminerà la sua esecuzione non appena l'utente sarà uscito dall'area geofence. Infine, solo i valori ottenuti dai data class locali verranno salvati all'interno del database nella tabella ValoriIstanze (tramite l'AsyncTask **AsyncTaskAggiungiValoreIstanza()**) insieme ai data class loro associati e alla data d'inserimento. Questi valori verranno visualizzati sul grafico presente nella home dell'applicazione.

### 4.6.3 Il service

I services di Android sono delle componenti che possono essere eseguite anche quando l'applicazione è in backgroud. Ci sono due tipo di services:

- I **bound services**: la loro esecuzione termina con la chiusura dell'activity alla quale sono associati;
- I **foreground services**: continuano ad essere eseguiti anche nel caso in cui l'applicazione venga chiusa.

In questo progetto si è scelto di usare il foreground service, è stato realizzato mediante la classe **MyService.java** e svolge la funzione di monitorare la posizione in modo tale che i geo-fence possano funzionare anche quando l'applicazione è in background. Per recuperare la posizione del dispositivo è stato usato un oggetto della classe FusedLocationProviderClient, la quale permette una gestione ottimizzata e una localizzazione precisa [37]. Si è reso necessario creare un oggetto di tipo LocationRequest il quale mi ha permesso di impostare l'accuratezza della posizione e l'intervallo di tempo (indicante ogni quanto recuperare la posizione). Dopodichè nell'onCreate() verrà richiamato il metodo showNotificationAndStartForegroundService() il quale farà comparire una notifica e avvierà il service. All'interno di esso verrà preparato un pending intent denominato *stop_service* ed il suo compito sarà quello di fermare il service al click sul pulsante presente nella notifica andando a risvegliare il **BroadcastFermaService**.

```
Intent intentFermaService = new Intent(this, BroadcastFermaService.
    class);
PendingIntent stop_service = PendingIntent.getBroadcast(this, (int)
    System.currentTimeMillis(), intentFermaService, PendingIntent.
    FLAG_CANCEL_CURRENT);
```

Listing 4.26: Pending intent per fermare il service.

```
NotificationCompat.Builder builder;
NotificationManager notificationManager =
                (NotificationManager) getSystemService(Context.
    NOTIFICATION_SERVICE);
builder = new NotificationCompat.Builder(this, CHANNEL_ID);
builder.setSmallIcon(R.drawable.service_notification)
.setContentTitle(getString(R.string.app_name))
.setContentText("Avvio del monitoraggio della posizione")
.addAction(R.drawable.ic_launcher_background, "Ferma il monitoraggio.",
     stop_service);
```

Listing 4.27: Costruzione della notifica.

```
@Override
public void onReceive(Context context, Intent intent) {
    Intent service = new Intent(context, MyService.class);
    context.stopService(service);
}
```

Listing 4.28: Struttura del broadcast utile a fermare il service.

Inoltre, all'interno della classe MyService è presente una variabile di tipo booleana denominata *isRunning*, la quale conterrà il valore *true* nel caso in cui il service fosse attivo. Ciò mi è stato utile perchè all'interno dei timer dei due broadcast descritti precedentemente (BroadcastGeofenceTuaIstanza e GeofenceBroadcastReceiver) è presente un controllo che verificherà se *isRunning* abbia o meno valore *true*. Qualora fosse *false*, l'esecuzione del timer verrà fermato e insieme a lui l'esecuzione del broadcast. Richiamando il metodo onDestroy() della classe MyService sarà possibile fermare il service, e la posizione non sarà più recuperata. Per utilizzare tali funzionalità sarà necessario recarsi nelle impostazioni di localizzazione del dispositivo, cliccare sull'applicazione GeoMonitor e consentire sempre la posizione.

### 4.6.4 Le notifiche

Le notifiche sono già state descritte nel corso delle sezioni precedenti. Vi sono due tipi di notifiche che possono essere ricevute:

- Quelle che comunicano il valore risultante dall'esecuzione di un'istanza remota;
- Quelle che comunicano se il calcolo dell'istanza del template locale è risultato veritiero o meno.

Le notifiche vengono richiamate dai due broadcast utilizzando un oggetto della classe **NotificationHelper**. Utilizzando il metodo *sendHighPriorityNotification()* si potrà impostare il titolo della notifica ed il suo **contenuto**.

```
notificationHelper.sendHighPriorityNotification("Istanza calcolata!", "
    La condizione posta dal template risulta essere vera. Controlla l'
    output nella schermata sottostante la mappa.", MappaTuaIstanza.class
    );
```

Listing 4.29: Esempio di una notifica.

Dopo aver costruito la notifica, *sendHighPriorityNotification()* si occuperà di inviarla attraverso il metodo **notify()**.

```
public void sendHighPriorityNotification(String title, String body,
    Class activityName){
     Notification notification = new NotificationCompat.Builder(this,
    CHANNEL_ID)
          .setSmallIcon(R.drawable.ic_place_green)
          .setPriority(NotificationCompat.PRIORITY_HIGH)
          .setStyle(new NotificationCompat.BigTextStyle()
              setSummaryText("Notifiche Geofencing")
              .setBigContentTitle(title)
              .bigText(body))
              .setAutoCancel(true)
              .build();
     NotificationManagerCompat.from(this).notify(new Random().nextInt(),
     notification);
}
```

Listing 4.30: Costruzione di una notifica.

## 4.7 Visualizzazione istanze remote e locali

Il procedimento con il quale sono state realizzate le activity contenenti la lista dei template e delle istanze è lo stesso, perciò andrò a descrivere solamente una delle due: la visualizzazione delle istanze. Per realizzare ciò si è utilizzata solo un'activity **SchermataTueInstanze.java** all'interno della quale è stato collocato un *frame layout* in grado di ospitare i fragment. Il frame layout può essere considerato come un contenitore di fragment i quali possono essere sostituiti uno all'altro dinamicamente. In particolare, nella parte alta dell'activity, sono presenti due pulsanti *radiobox* utilizzati per passare dalle istanze locali a quelle remote o viceversa, insieme ad un bottone "aggiorna" utile a confermare l'operazione. Nel caso in questione i fragment utilizzati sono due:

- **FragmentIstanzeRemote**: incaricato di mostrare le istanze *remote*;
- **FragmentIstanzeLocali**: incaricato di mostrare le istanze *locali*.

```
1 btnAggiorna.setOnClickListener(new View.OnClickListener() {
2     @Override
3     public void onClick(View v) {
4         selectedRadioButton  = (RadioButton)findViewById(radioGroup.
    getCheckedRadioButtonId());
5         String scelta = selectedRadioButton.getText().toString();
6         if(scelta.equals("Istanze Locali")){
7          getSupportFragmentManager().beginTransaction().replace(R.id.
    frag_graf_istanze, new FragmentIstanzeLocali()).commit();
8         }else{
9          getSupportFragmentManager().beginTransaction().replace(R.id.
    frag_graf_istanze, new FragmentIstanzeRemote()).commit();
```

Listing 4.31: Implementazione del passaggio da un fragment all'altro.

Entrambi i fragment sono dotati di una recycler view (una schermata scrollabile verticalmente) la quale verrà riempita da tante celle quante sono le istanze. La recycler view fa uso di un layout manager (che posiziona gli elementi nel layout), un'adapter e di un view holder. Il view holder ha il compito di recuperare gli elementi definiti nel file *xml* che costruisce la struttura della cella. In questo caso sono stati utilizzati due file *xml*: *riga_tue_istanze* per le istanze locali e *vedi_istanze_sensquare* per le istanze remote. Il

fine era quello di dare due stili diversi per le due categorie di istanze. Successivamente i fragment passeranno all'adapter i valori (provenienti dal database nel caso delle istanze locali oppure dalla richiesta API nel caso delle istanze remote) che saranno inseriti nelle varie celle attraverso il view holder. Dal momento che l'adapter utilizzato è solamente uno per entrambi i fragment, esso utilizzerà una variabile booleana (passata dai due fragment) la quale indicherà se dovranno essere caricati dati locali o remoti. In questo modo l'adapter sarà in grado di recuperare il file *xml* corretto.

```
if(mostraRemoti){
    View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.
    vedi_instanze_sensquare, parent, true);
    AdapterTueInstanze.TueInstanzeHolder myHolder = new
    AdapterTueInstanze.TueInstanzeHolder(v, mOnNoteListener, true);
}else{
    View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.
    riga_tue_instanze, parent, false);
    AdapterTueInstanze.TueInstanzeHolder myHolder = new
    AdapterTueInstanze.TueInstanzeHolder(v, mOnNoteListener, false);
}
```

Listing 4.32: Caricamento del file xml corretto.

```
for(CustomServiceLocal istanzaRecuperata : instanzaLocale ){
    arrayTitoli.add(istanzaRecuperata.getTitolo());
    arrayID.add(istanzaRecuperata.getId());
}
mAdapter = new AdapterTueInstanze(false, arrayTitoli, arrayIDIstanze,
    getActivity(), interfaceClick);
```

Listing 4.33: Passaggio dei valori necessari alla costruzione della recycler view nel caso di istanze locali all'interno del **FragmentIstanzeLocali**.

```
AdapterTueInstanze adapter = new AdapterTueInstanze(true,
    arrayTitoliSenSquare, arrayIDSenSquare, getActivity(),
    interfaceClick);
    RecyclerView.LayoutManager layoutManager = new LinearLayoutManager(
    getActivity().getApplicationContext());
    mRecyclerView.setLayoutManager(layoutManager);
```

```
4     mRecyclerView.setItemAnimator(new DefaultItemAnimator());
    mRecyclerView.setAdapter(adapter);
```

Listing 4.34: Passaggio dei valori necessari alla costruzione della recycler view nel caso di istanze remote all'interno del **FragmentIstanzeRemote**.

# Capitolo 5

# Interfaccia Grafica

## 5.1 Modifiche a SenSquare

Grazie alle modifiche fatte alle API di SenSquare, è ora possibile visualizzare i nuovi parametri nello spinner presente nella costruzione dei template. Tuttavia per fare ciò è stato necessario attuare un ulteriore modifica oltre a quelle elencate precedentemente nel capitolo 2. Infatti, SenSquare utilizza una libreria dedicata alla traduzione delle varie pagine denominata *ngx-translate* [38] la quale permette di creare diversi file JSON contenenti le varie lingue con le quali si vuole internazionalizzare il sito. Al suo interno sono presenti anche i data class con i quali potrebbero essere costruiti i template. Per fare funzionare correttamente SenSquare, dal momento che sono stati aggiunti nuovi data class, è stato necessario aggiornare il contenuto del file JSON. I file modificati si trovano all'interno della directory i18n del progetto originario SenSquare.

```
1       "DATA_CLASSES": {
2     "BRTH": "Brightness",
3     "COPR": "Carbon Monoxide",
4     "GASL": "Gas Level",
5     "HUMY": "Humidity",
6     "NO2A": "Nitrogen Dioxide",
7     "NOXO": "Nitrogen Monoxide",
8     "O3OZ": "Ozone",
9     "PM10": "PM10",
10    "PM25": "PM2.5",
```

```
    "POWR": "Power",
    "PRES": "Pressure",
    "RAIN": "Rain Index",
    "TEMP": "Temperature",
    "VOLT": "Voltage",
    "WIDR": "Wind Direction",
    "WISP": "Wind Speed",
    "COVD": "New Cases Covid19"
}
```

Listing 5.1: Aggiornamento del file en.json contenente la traduzione in inglese.

La figura *5.1* mostra uno dei primi passi da compiere in fase della costruzione di un template (su SenSquare) mentre nell'immagine *5.2* è possibile notare l'istanza di uno dei template con oggetto il nuovo data class inerente ai nuovi positivi da coronavirus, istanziata nell'area di Bologna.

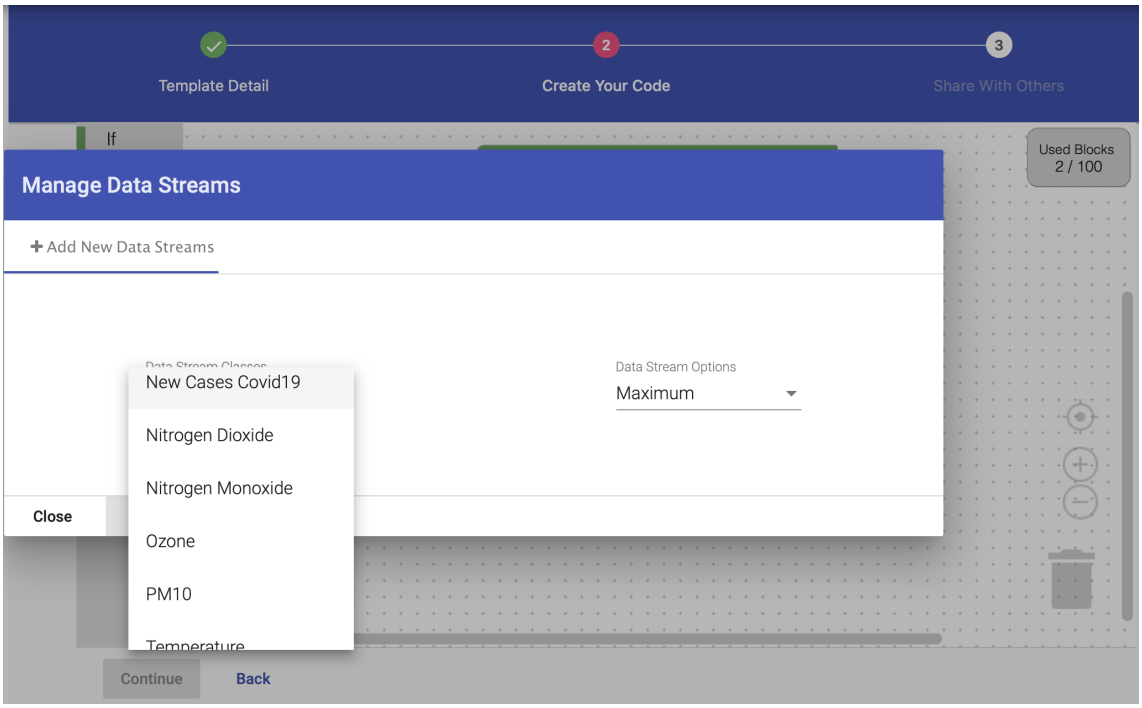



Figura 5.1: Lo spinner

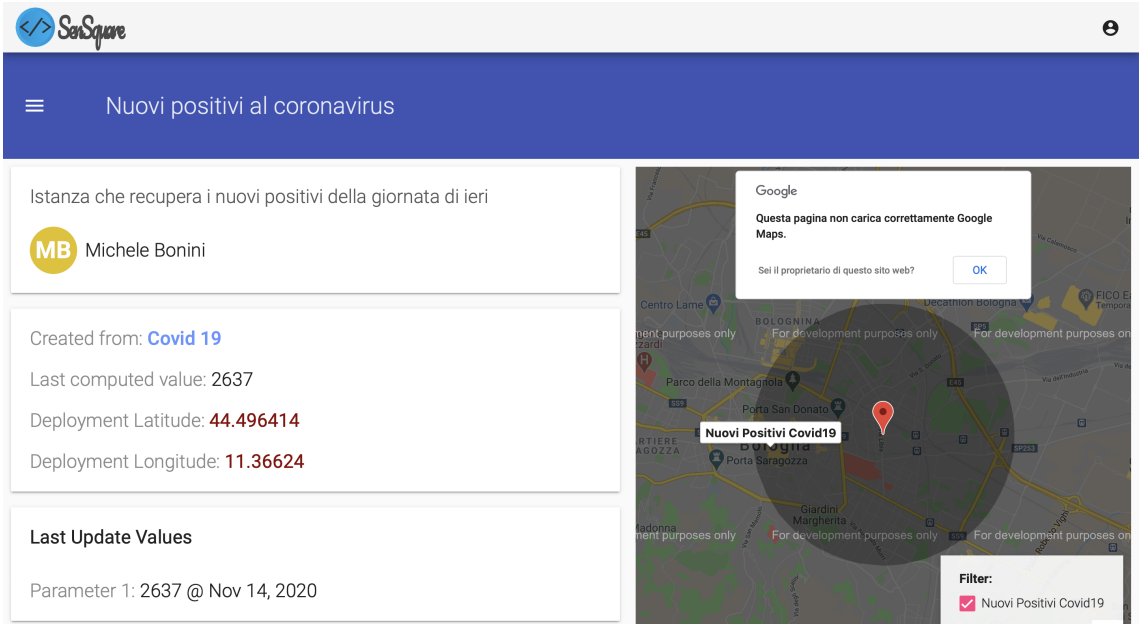


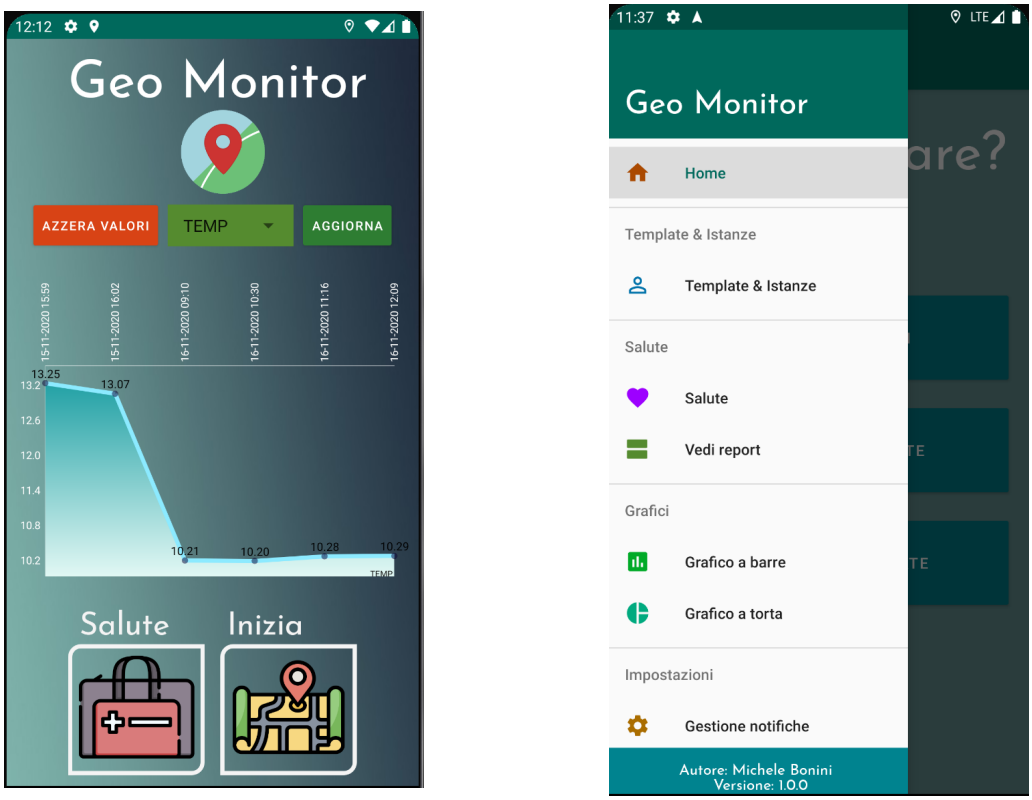

Figura 5.2: Istanziazione di un Template

### 5.1.1 Activities e Fragments

L'applicazione GeoMonitor [39] è stata costruita mediante l'uso di activities e fragments alcune delle quali navigabili mediante un navigation drawer. L'activity è quella porzione visibile dello schermo grazie al quale l'utente può interagire con l'app, mentre il fragment può essere definito come una componente dell'activity dal momento che senza di essa non potrebbe esistere. In particolare i fragments sono vantaggiosi perchè permettono la riusabilità del codice all'interno dell'applicativo. Il navigaton drawer è il menu laterale, ed è stato aggiunto per migliorare l'esperienza dell'utente permettendogli di raggiungere le schermate principali con un semplice click.

## 5.2 La home

La home è la schermata di avvio dell'applicazione. Essa è stata implementata attraverso l'activity **PaginaPrincipale.java**. Al suo interno è presente un frame layout il quale avrà il compito di ospitare un fragment contenente grafico.




(a) La home

(b) Il menu laterale

Figura 5.3: La home ed il menu laterlae

Il grafico è stato realizzato mediante la classe **SchermataGraficoFragment.java** e mostra sull'asse delle $x$ i valori di un data class scelto dall'utente e sull'asse delle $y$ le date di recupero di tali valori. È possibile scegliere il data class al quale saranno associati i valori presenti nel grafico attraverso lo spinner presente sotto al logo di GeoMonitor. Premendo sul pulsante rosso "azzera valori" verranno rimossi i valori dal grafico (e dal database) in modo tale da poter ri-iniziare un monitoraggio magari in un'altra area geografica. Il menu laterale permette una navigazione migliore tra le varie schermate all'interno dell'applicazione.

Esso può essere aperto mediante uno *swipe* verso destra oppure cliccando sul pulsante (*hamburger menu*) in alto a sinistra, e presenta le seguenti opzioni:

- **Home**: è la schermata principale (o di avvio) dell'applicazione;

- **Template e Istanze**: permette di raggiungere il menu con le tre opzioni;
- **Salute**: permette di raggiungere l'activity dedicata al monitoraggio della salute;
- **Vedi report**: è la schermata mostrante tutti i report aggiunti;
- **Grafico a barre**: mostra il grafico a barre di uno dei parametri della salute;
- **Grafico a torta**: mostra il grafico a torta di uno dei parametri della salute;
- **Gestione notifiche**: schermata dedicata all'impostazione delle notifiche inerenti alla salute.

## 5.3 Gestione templates ed istanze

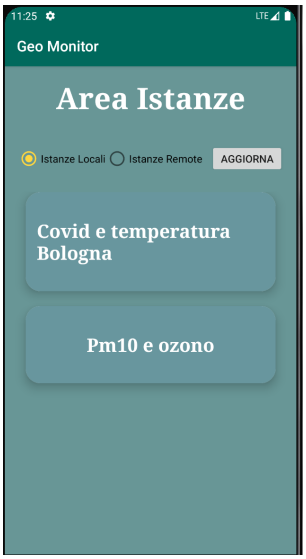
La visualizzazione delle istanze e dei template mediante fragment e recycler view, descritta nel capitolo 4, è mostrata in questa sezione. Come è possibile notare vi sono 2 radiobox nella parte alta dell'activity. Una volta selezionata una scelta, bisognerà premere sul pulsante "aggiorna" per completare l'operazione.


Figura 5.4: Schermata delle tre opzioni

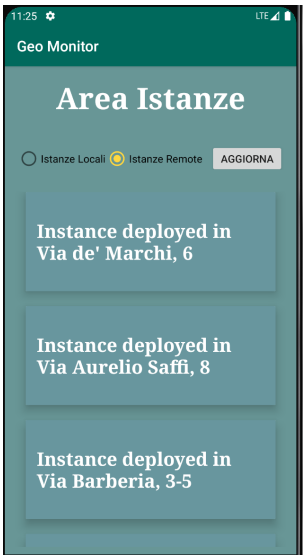
Le istanze (o template) locali provengono dal database mentre quelle (o quelli) remote/i dal sito SenSquare. Le celle degli oggetti locali presentano una forma più tondeggiante rispetto a quelli remoti. Ciò è stato fatto per poterle distinguere meglio.


(a) Lista template locali


(b) Lista template remoti




(c) Lista istanze locali




(d) Lista istanze remote

Figura 5.5: Visualizzazione dei template e delle istanze

Mentre per i template locali vengono visualizzate anche le loro strutture, ovvero

l'output del processo di costruzione, per i template remoti non è stato possibile dal momento che il processo di costruzione avviene mediante libreria blockly e non si è potuto reperire la struttura del template. Quindi la visualizzazione del template è solo a scopo informativo e permette di reperire le seguenti informazioni: l'id, il titolo, la descrizione, i parametri trattati e quali operatori sono stati usati. Infatti su SenSquare, dati un numero $n$ di data stream presenti nella zona e uno specifico data class, è possibile recuperare i valori provenienti da:

- La **media** delle ultime misurazioni di ogni data stream;
- Il **massimo** o il **minimo** delle ultime misurazioni a cui sono associati i data stream;
- Uno **specifico** data stream di quella zona, scelto dall'utente.

Nell'applicazione GeoMonitor questa funzionalità non è stata offerta. Nel nostro caso, dati un numero $n$ di data stream sulla mappa, il valore delle misurazioni ottenuto è dato dalla media di tutti i valori associati ai data stream. I template remoti non possono essere istanziati a causa della mancanza delle informazioni utili a tale processo.


(a) Dettagli dei template locali (b) Dettagli dei template remoti

Figura 5.6: Dettagli template

Per quel che riguarda la visualizzazione dei dettagli delle istanze remote, differisce solamente per la presenza della struttura del template nel caso delle istanze locali. Le informazioni in comune delle istanze (locali e remote) mostrate dalle due activity sono: l'id, il titolo, la descrizione, lo stato, le coordinate del centro dell'area posizionata sulla mappa ed il raggio. Inoltre è presente un *picker* mediante il quale l'utente può scegliere ogni quanto attivare l'istanza.


(a) Riepilogo informazioni istanza


(b) Selezione dei minuti

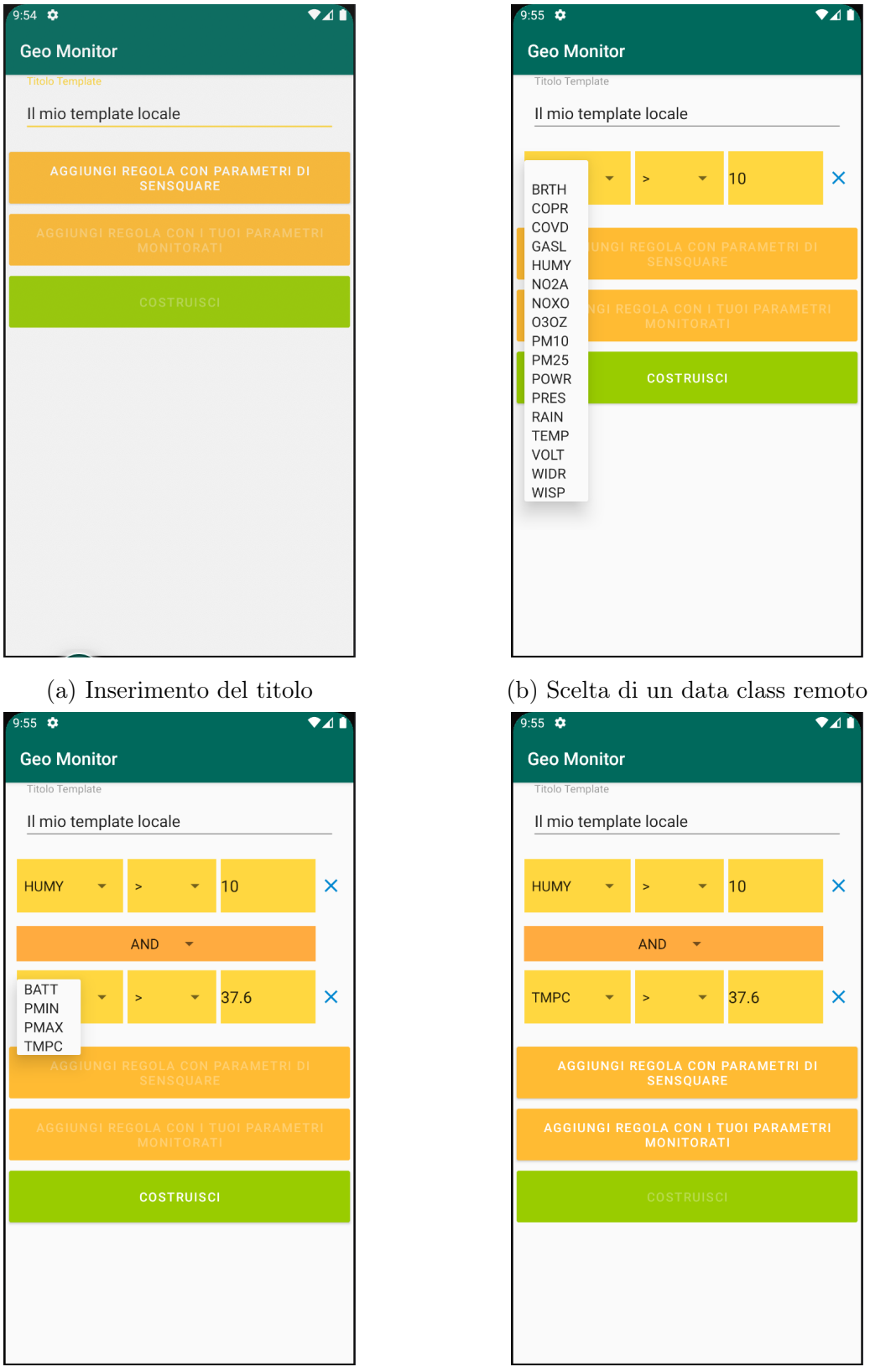

(c) Pulsante visiona istanza

Figura 5.7: Attivare un'istanza: fase 1

Solamente nel caso in cui l'istanza fosse attiva, il pulsante "attiva istanza" sarà sostituito dal pulsante "visiona".

### 5.3.1 Costruire un template

L'activity dedicata costruzione dei template è denominata **SchermataCostruzione.java**. Inizialmente essa mostra tre pulsanti disponibili, di cui due disabilitati e un campo dedicato all'inserimento del titolo del template che dovrà essere costruito. Di conseguenza, il template dovrà contenere almeno una regola con parametri provenienti da SenSquare. Dopo aver aggiunto ciò, sarà possibile aggiungere un numero $n$ di regole dedicate al monitoraggio ambientale, positivi da coronavirus (e quindi regole di SenSquare) oppure regole riguardanti lo stato di salute. Questi costrutti verranno aggiunti ad una schermata scrollabile. Le regole di SenSquare sono rappresentate da tre componenti grafiche: uno spinner contenente i data class di SenSquare, un altro contenente degli operatori aritmetici e poi un campo dedicato all'introduzione di valori numerici. La stessa cosa vale per le regole inerenti la propria salute, nelle quali il primo spinner sarà popolato con data class locali. Solo nel caso le regole inserite fossero più di una, verrà aggiunto un'ulteriore spinner che permetta alle due regole di essere in un qualche modo correlate attraverso uno i seguenti operatori: and, or, $<$, $>$, $=$ . Ancora, per poter procedere con la costruzione del template, al termine dell'inserimento di una regola, sarà necessario confermare l'operazione premendo sul pulsante "costruisci". Solo successivamente i pulsanti arancioni (mostrati nella figura 5.8) diverrano cliccabili mentre quello verde diverrà disabilitato (e si riabiliterà una volta cliccato uno dei due arancioni). È possibile eliminare le regole inserite. Per inserire i template tutti i campi delle regole devono essere completati altrimenti verrà restituito un messaggio d'errore.




(a) Inserimento del titolo

(b) Scelta di un data class remoto

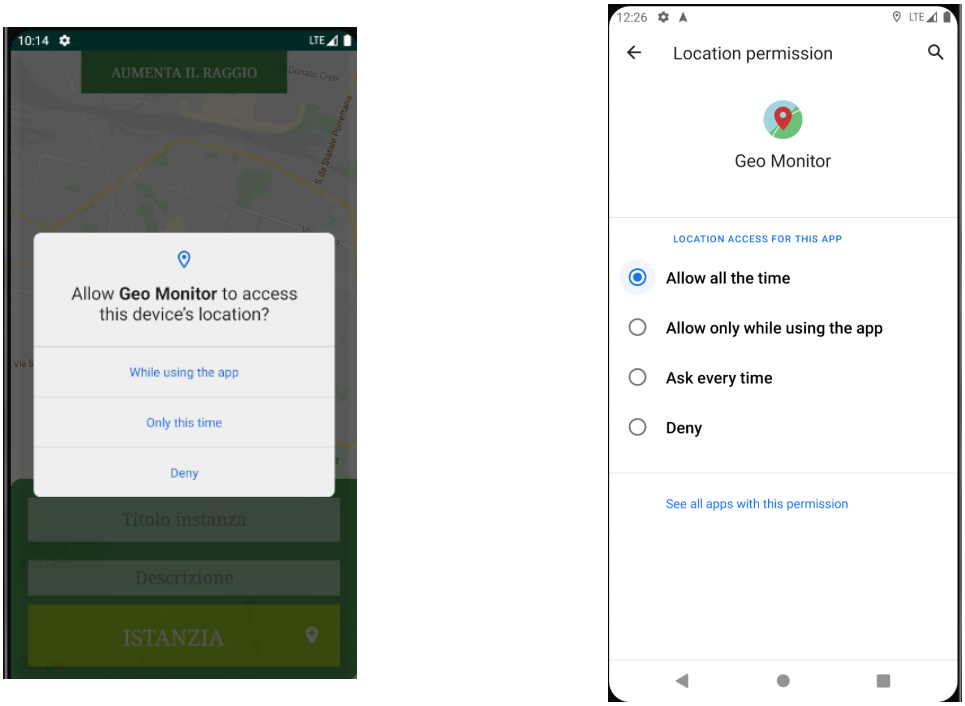
(c) Scelta di un data class locale

(d) Lista istanze remote

Figura 5.8: Esempio della costruzione di un template

### 5.3.2 Istanziare un template

Per utilizzare la funzionalità della mappa presente su GeoMonitor sarà necessario concedere i permessi di localizzazione. Perciò bisognerà acconsentire ai permessi come mostrano le immagini *a* e *b* nella figura 5.9 . Per raggiungere la schermata *b* bisognerà recarsi in *impostazioni* > *localizzazione* > *GeoMonitor*.



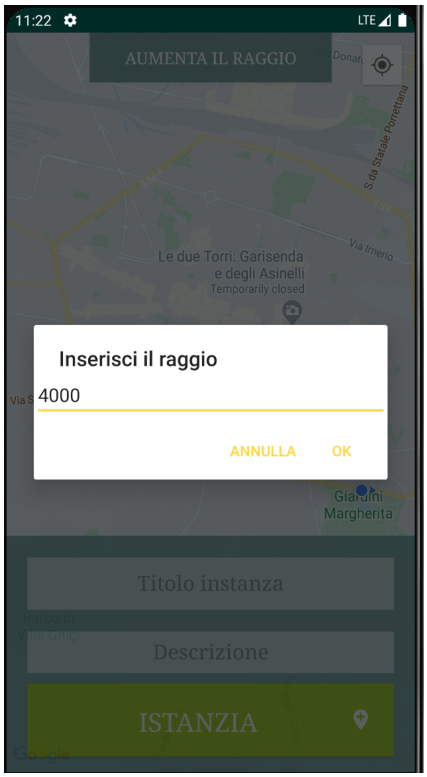

(a) Permessi dall'applicazione

(b) Permessi dalle impostazioni

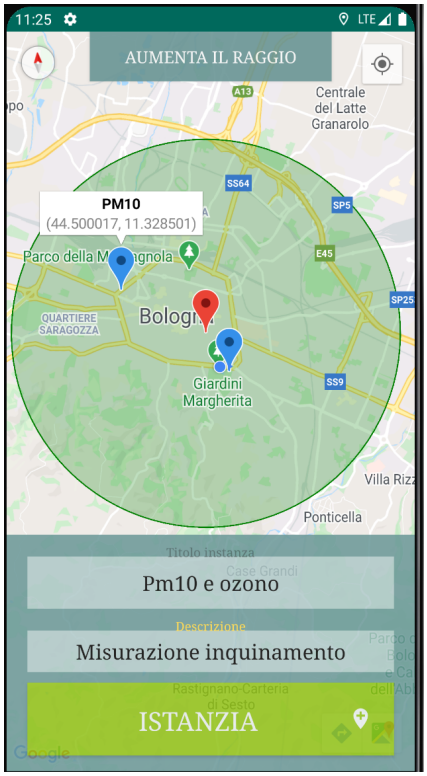
Figura 5.9: Utilizzo dei permessi

Istanziare un template permette all'utente di scegliere la zona dalla quale verranno prelevati i valori per poter essere computati seguendo le linee guida del template. Tuttavia, per poter istanziare un template, dovranno essere disponibili i data stream all'interno dell'area. In particolare, come si potrà notare nell'immagine *b* della figura 5.10 sono presenti due marker rappresentanti i data stream in quella zona. Qualora l'utente provasse ad istanziare un template senza inserire il titolo dell'istanza o la sua descrizione oppure nel caso in cui all'interno del confine virtuale non vi siano i data stream necessari, il sistema notificherà un'errore. Durante tale processo sarà possibile aumentare il raggio del confine virtuale (un'unità è pari ad 1 km).


(a) Riepilogo informazioni template




(b) Aumentare il raggio del confine virtuale

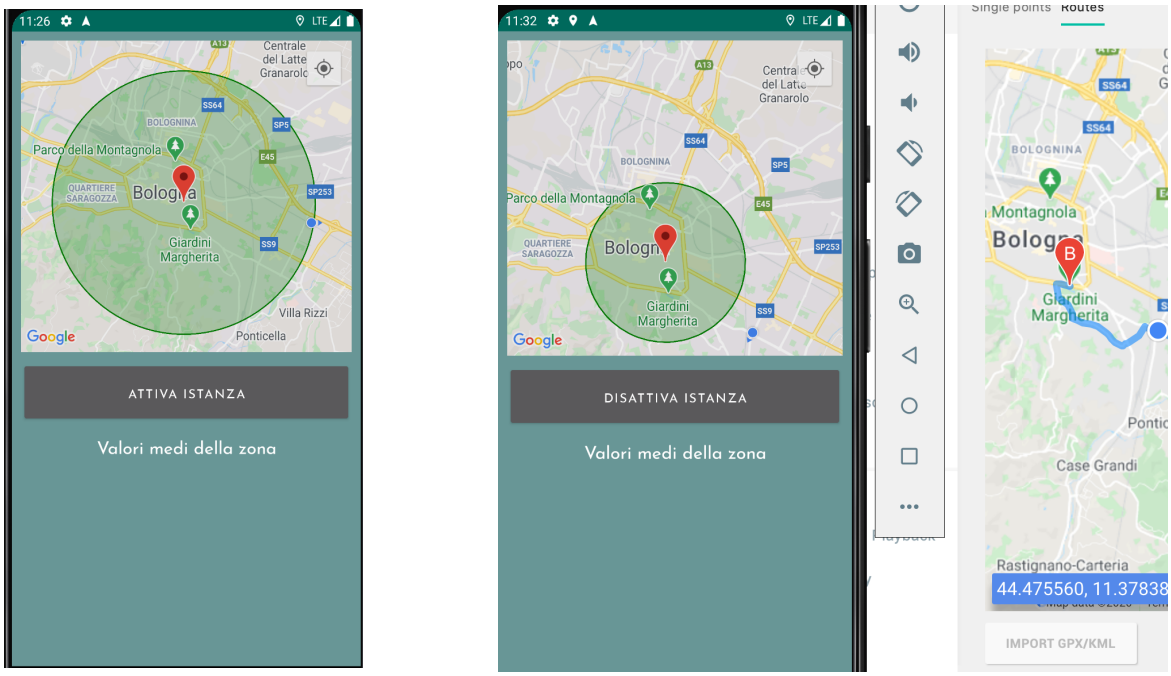



(c) Visualizzazione dei data stream presenti nella zona selezionata

Figura 5.10: Istanziare un template

### 5.3.3 Attivare un'istanza

Passando per l'activity mostrata nella figura 5.7, è possibile attivare un'istanza. Tale funzionalità permetterà all'utente di ricevere notifiche le quali segnaleranno: il valore risultante dall'esecuzione dell'istanza nel caso esse siano remote oppure la veridicità dell'istanza nel caso siano locali. Sempre nell'activity della figura 5.7 è possibile visionare lo stato dell'istanza. Esso potrà essere "attiva" o "non attiva". Come si potrà notare nell'immagine, qualora l'istanza sia attiva, il pulsante in figura *b* non mostrerà più "attiva istanza" ma bensì "visiona istanza".

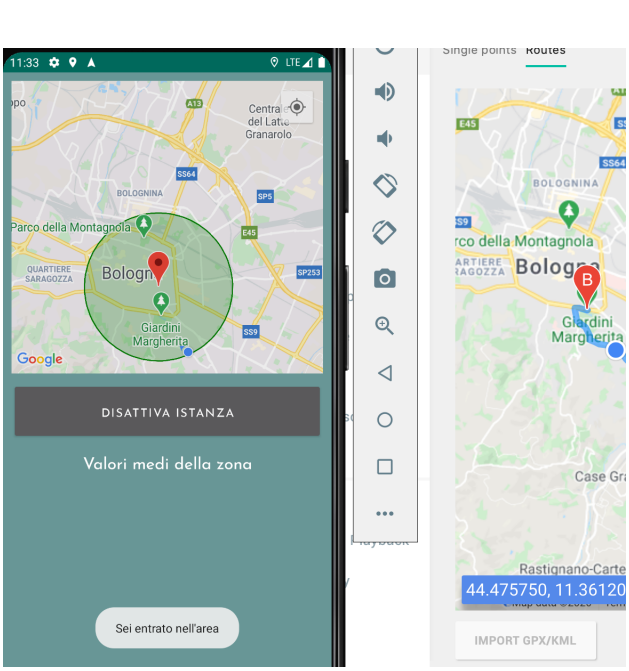



(a) L'istanza è disattivata

(b) L'istanza è attiva ma l'utente è fuori dall'area geo-fence

Figura 5.11: Attivare un'istanza: fase 2

La figura 5.11 a mostra la posizione dell'utente all'interno dell'area geo-fence. Tuttavia i valori di quella zona non sono stati calcolati perchè l'istanza non è stata attivata.




(a) L'utente sta entrando nell'area geo-fence


(b) L'utente è entrato nell'area geo-fence

Figura 5.12: Attivare un'istanza: fase 3

L'avvio di un'istanza permetterà di far partire il service il quale monitorerà la posizione in background oppure con l'applicazione chiusa. Il monitoraggio della posizione si potrà fermare premendo sul pulsante "*ferma il monitoraggio*" presente nella notifica di foreground.

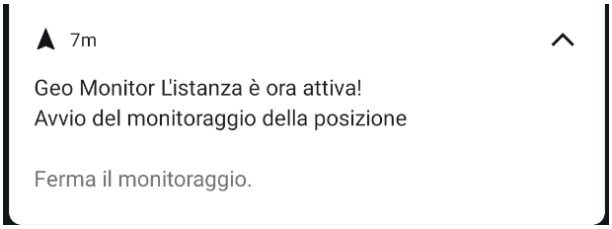



Figura 5.13: Notifica dell'avvio del foreground service

Ancora, attivare un'istanza significherà andare a creare una nuova area geo-fence sulla base della zona nella quale si sarà deciso di istanziare i template (sia remoti che locali).

Solo quando l'utente entrerà nell'area geo-fence verrà mandata in esecuzione l'istanza e verranno mostrati i valori dei parametri presenti all'interno del template. Tale processo è rappresentato all'interno della figura 5.11 e 5.12. Nel caso in cui l'utente uscisse dall'area geo-fence, l'istanza non sarebbe più eseguita. Inoltre, nell'immagine 5.14 è rappresentato l'output dell'attivazione di una possibile istanza. Come già accennato in precedenza, alcuni parametri possiedono un campo denominato livello il quale indicherà la pericolosità del valore monitorato di un certo parametro. Tuttavia, non tutti i parametri possiedono un livello.

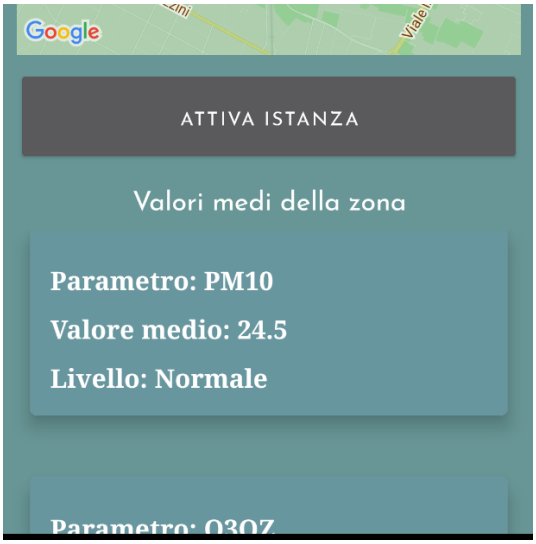
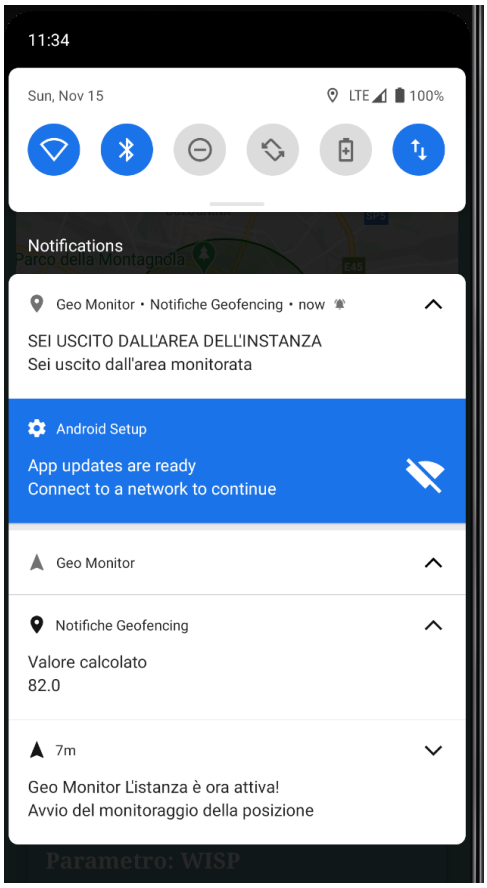
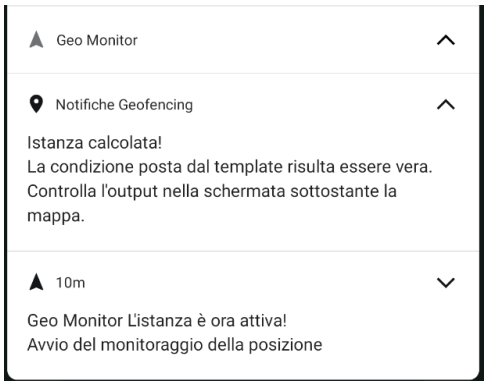



Figura 5.14: Recycler view con i valori ottenuti

Per concludere, tra le varie notifiche che si possono ricevere, sono presenti anche quelle che comunicano l'uscita dall'area geo-fence.




(a) Notifiche delle istanze SenSquare e dell'uscita dall'area geo-fence




(b) Notifiche delle istanze locali

Figura 5.15: Le notifiche

# Capitolo 6

# Conclusioni e Sviluppi futuri

Nella prima parte di questa tesi sono stati introdotti i paradigmi dell'Internet of Things, del context awareness e del IoT collaborativo. Dopodichè ho discusso e confrontato le principali funzionalità di GeoMonitor con SenSquare, una piattaforma IoT collaborativa la quale è in grado di utilizzare dati eterogenei nella creazione di servizi individuali o per la comunità. Successivamente ho presentato e dettagliato il funzionamento dell'applicazione Android, denominata GeoMonitor, da me realizzata. Sono state discusse le modifiche attuate a SenSquare e le modalità con le quali sono stati recuperati ed utilizzati dei nuovi parametri inerenti il monitoraggio ambientale ed i nuovi positivi da coronavirus giornalieri. Il potenziale più grande di questo progetto consiste nell'estendibilità perchè i nuovi parametri aggiunti alla piattaforma SenSquare potranno essere utilizzati anche dall'applicazione mobile. Concludendo, GeoMonitor è in grado di consentire agli utenti di costruire servizi ad hoc da poter istanziare in zone specifiche e di essere aggiornati sulle condizioni ambientali di tali aree, magari tenendo conto anche del proprio stato di salute. Inoltre tra questi parametri è considerato anche il numero di nuovi positivi da coronavirus in modo tale da avvisare l'utente nel caso in cui l'area in cui egli è entrato sia più o meno pericolosa.

## 6.1 Sviluppi futuri

Le estensioni future possibili sono numerose. Per esempio, per quel che riguarda la sezione del monitoraggio della salute, si potrebbe implementare la possibilità di rilevare i valori attraverso degli wearable come per esempio gli smartwatch che monitorano la frequenza cardiaca o i livelli di ossigeno nel sangue. Inoltre, l'utilità di questa applicazione diventerà sempre più grande all'aumentare del numero di parametri in grado di essere raccolti da SenSquare. Infine, anche un aumento dei sensori nelle nostre città permetterà agli utenti di costruire ed utilizzare servizi sempre più precisi ed attendibili.

## 6.2 Difficoltà riscontrate

Sono state riscontrate delle difficoltà nello studio di alcuni componenti lato server-side di SenSquare come ad esempio le modalità con le quali si effettuavano chiamate API. Inoltre ho trascorso qualche settimana a provare a caricare online una versione più recente di SenSquare. Tuttavia la configurazione della mia macchina non me lo ha permesso e si è deciso di continuare ad utilizzare la versione più "obsoleta" dal momento che essa possedeva comunque tutti i requisiti per poter essere utilizzata senza alcun problema.

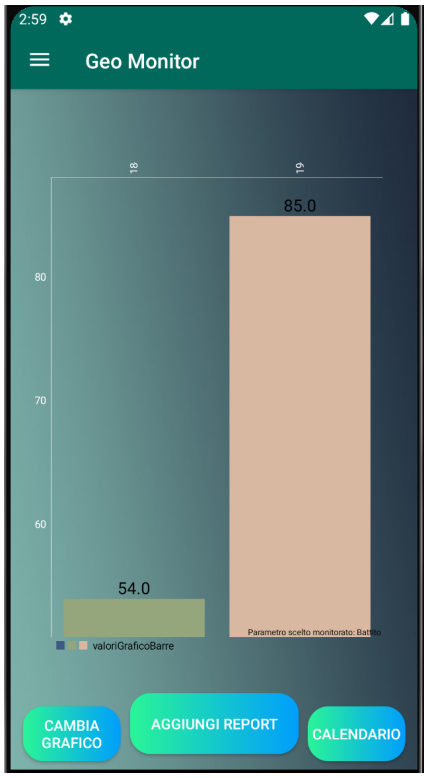
## 6.3 Testing

I simulatori virtuali utilizzati per testare l'applicazione sono stati i seguenti: Pixel 3, Pixel 2 e Nexus 6. Inoltre, tutte le funzionalità sono state provate su un dispositivo mobile reale *Samsung A40*. L'applicazione è disponibile a partire dalla versione **Marshmellow 6.0** (API 23) a quella più recente (**Android 10**).

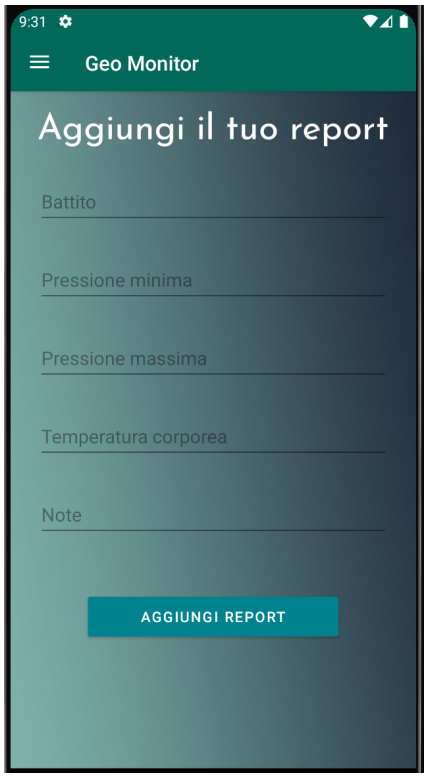
# Appendice A

# Il monitoraggio della salute

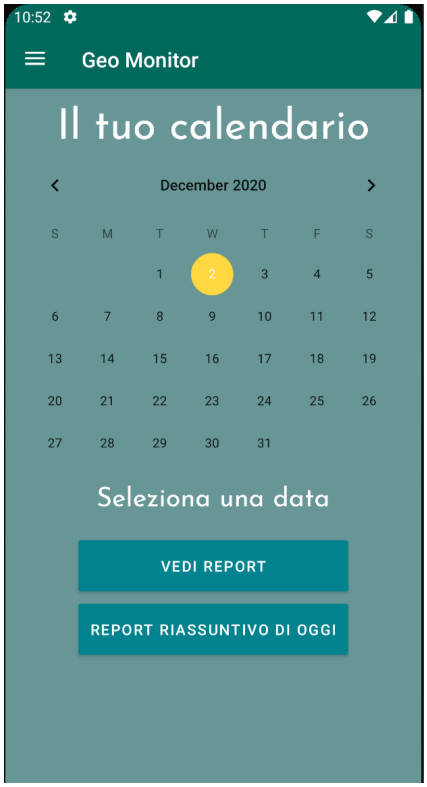
La sezione dell'applicazione GeoMonitor dedicata al monitoraggio dello stato di salute è stata realizzata per il progetto di applicazioni mobili. Durante questo lavoro di tesi è stata apportata qualche miglioria dal punto di vista implementativo e grafico, tuttavia sono stati cambiamenti minimi come ad esempio la visualizzazione dei report all'interno del grafico a barre suddivisi per i giorni della settimana. Le specifiche richiedevano la realizzazione di un'app che permettesse di monitorare giornalmente i parametri della salute dell'utente ricordandogli, attraverso notifiche, di inserire dei report giornalieri. L'orario di tali notifiche si potrà modificare sulla base della preferenza dell'utilizzatore. Ad ogni report è associata una priorità ed è possibile cercarli filtrandoli per questi livelli di importanza. Si potranno visionare tutti i report inseriti nell'arco di una giornata grazie ad un calendario presente all'interno dell'applicazione oppure semplicemente un report riassuntivo (i cui parametri sono stati calcolati facendo la media dei valori per ogni parametro). Un'altro tipo di notifiche presenti è quello riguardante la soglia dei parametri della salute. Infatti l'utente potrà impostare un intervallo di tempo ed una soglia per uno specifico parametro e nel caso in cui la media dei valori inseriti per quel parametro, al raggiungimento della fine dell'intervallo di tempo prestabilito, abbia superato la soglia inizialmente impostata, l'utente riceverà una notifica. È possibile visionare i report inseriti nell'arco di 7 giorni attraverso un grafico a barre oppure utilizzare un grafico a torta.




(a) Schermata iniziale




(b) Aggiunta di un nuovo report




(c) Il calendario


(d) Visualizzazione report filtrati per priorità

Figura A.1: Monitoraggio della salute

# Bibliografia


[1] Immagine dell'esperimento di Simon Weckert. *https://www.ansa.it/canale_motori/notizie/attualita/2020/02/04/traffico-simulato-come-forma-darte-inganna-google-maps_52ff2633-43be-4c2b-b923-b934cc1399a3.html*.

[2] Immagine Geo-fence. *https://developer.android.com/training/location/geofencing*.

[3] Gerd Kortuem, Fahim Kawsar, Daniel Fitton, and Vasughi Sundramoorthy. Smart objects as building blocks for the internet of things. *Internet Computing, IEEE*, 14:44 – 51, 03 2010.

[4] Kevin Ashton et al. That 'internet of things' thing. *RFID journal*, 22(7):97–114, 2009.

[5] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Context aware computing for the internet of things: A survey. *IEEE Communications Surveys Tutorials*, 16(1):414–454, 2014.

[6] Ovidiu Vermesan, Peter Friess, Patrick Guillemin, Sergio Gusmeroli, Harald Sundmaeker, Alessandro Bassi, Ignacio Soler Jubert, Margaretha Mazura, Mark Harrison, Markus Eisenhauer, et al. Internet of things strategic research roadmap. *Internet of things-global technological and societal trends*, 1(2011):9–52, 2011.

[7] ALICIA Asin and DAVID Gascon. 50 sensor applications for a smarter world. *Libelium Comunicaciones Distribuidas, Tech. Rep*, pages 589–594, 2012.

[8] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of things for smart cities. *IEEE Internet of Things journal*, 1(1):22–32, 2014.

[9] Definizione di sensore. *https://www.treccani.it/enciclopedia/sensore/*.

[10] F. Montori, L. Bedogni, and L. Bononi. A collaborative internet of things architecture for smart cities and environmental monitoring. *IEEE Internet of Things Journal*, 5(2):592–605, 2018.

[11] Graham Whitelaw, Hague Vaughan, Brian Craig, and David Atkinson. Establishing the canadian community monitoring network. *Environmental monitoring and assessment*, 88(1-3):409–418, 2003.

[12] R. K. Ganti, F. Ye, and H. Lei. Mobile crowdsensing: current state and future challenges. *IEEE Communications Magazine*, 49(11):32–39, 2011.

[13] L'esperimento Google Maps di Simon Weckert. *https://www.lastampa.it/tecnologia/news/2020/02/04/news/artista-inganna-google-maps-simula-traffico-con-99-smartphone-1.38424575*.

[14] B. N. Schilit and M. M. Theimer. Disseminating active map information to mobile hosts. *IEEE Network*, 8(5):22–32, 1994.

[15] P. J. Brown, J. D. Bovey, and Xian Chen. Context-aware applications: from the laboratory to the marketplace. *IEEE Personal Communications*, 4(5):58–64, 1997.

[16] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In Hans-W. Gellersen, editor, *Handheld and Ubiquitous Computing*, pages 304–307, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[17] How is Context and Context-awareness Defined and Applied? A Survey of Context-awareness. *http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.90.4517&rep=rep1&type=pdf*.

[18] Sachin W Rahate and MZ Shaikh. Geo-fencing infrastructure: location based service. *International Research Journal of Engineering and Technology*, 3(11):1095–1098, 2016.

[19] Pagina di Wikipedia in merito al Covid-19. *https://it.wikipedia.org/wiki/COVID-19*.

[20] Air Matters: un'applicazione per il monitoraggio ambientale. *https://air-matters.com*.

[21] BrezooMeter: un'applicazione per il monitoraggio ambientale. *https://breezometer.com*.

[22] L'applicazione Creek Watch ed il monitoraggio dei bacini d'acqua. *https://scistarter.org/creek-watch*.

[23] Yajun Lu and J Cecil. An internet of things (iot)-based collaborative framework for advanced manufacturing. *The International Journal of Advanced Manufacturing Technology*, 84(5-8):1141–1152, 2016.

[24] S. H. Alsamhi, O. Ma, M. S. Ansari, and F. A. Almalki. Survey on collaborative smart drones and internet of things for improving smartness of smart cities. *IEEE Access*, 7:128125–128152, 2019.

[25] Applicazione Salute di Apple. *https://www.apple.com/it/ios/health/*.

[26] Come funziona immuni. *https://www.agensir.it/italia/2020/10/15/coronavirus-immuni-che-cose-e-come-funziona-dieci-cose-da-sapere/*.

[27] Informazioni su Blockly. *https://developers.google.com/blockly/guides/overview/*.

[28] Arpae Emilia-Romagna. *https://www.arpae.it*.

[29] F. Montori, L. Bedogni, G. Iselli, and L. Bononi. Delivering iot smart services through collective awareness, mobile crowdsensing and open data. In *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 1–6, 2020.

[30] Sito web ThingSpeak. *https://thingspeak.com*.

[31] Open Weather. *https://openweathermap.org*.

[32] Repository GitHub dati Covid-19 Italia. *https://github.com/pcm-dpc/COVID-19/tree/master/dati-andamento-nazionale*.

[33] Django Rest Framework. *https://www.django-rest-framework.org*.

[34] Definizione di dato personale. *https://www.garanteprivacy.it/home/diritti/cosa-intendiamo-per-dati-personali*.

[35] Crontab. *https://crontab.guru*.

[36] Guida per installare i Google Play services. *https://developer.android.com/studio/intro/update.html#sdk-manager*.

[37] Informazioni sul fused location provider client . *https://developers.google.com/location-context/fused-location-provider*.

[38] NGX-Translate client . *https://github.com/ngx-translate*.

[39] Repository Gitlab del progetto . *https://gitlab.com/BonnyBay/progettotesi*.

# Ringraziamenti

Innanzitutto vorrei ringraziare il mio Relatore, il Dottor Federico Montori, per avermi permesso di lavorare a questo progetto innovativo, ma soprattutto per essere stato disponibile a supportarmi in questi mesi di lavoro.

Ringrazio i miei genitori, per essere stati un punto di riferimento e per avermi motivato a dare il massimo per tutto il percorso di questa laurea triennale, le mie due sorelle, Valentina e Alice, ed i miei nonni. Grazie a mio cognato, Alessandro, per i suoi consigli e per essersi confrontato con me nel momento del bisogno.

Ringrazio Saverio per i suoi preziosi suggerimenti e per le chiacchierate fatte, le quali sono state fonte di ispirazione ed incoraggiamento.

Grazie ai miei colleghi e amici Riccardo, Emanuele e Martino, compagni di studio e progetti, per i bei momenti passati insieme e per il supporto a me dato durante questi mesi impegnativi.

Infine, vorrei ringraziare i miei amici ed amiche per essermi stati vicini e per le giornate trascorse insieme.