

# Personal Health Monitor

Progetto Laboratorio Applicazioni Mobili 2019/2020 - Sara Vorabbi

## Obbiettivo - Specifiche del Progetto

Allo studente viene richiesta l'implementazione di una applicazione interattiva per il tracciamento di informazioni personali riguardanti la salute dell'utente da salvare in report giornalieri. In particolare, l'applicazione deve essere in grado di gestire i report all'interno di un calendario, mandare notifiche e filtrare i dati secondo filtri specifici.

## Overview

L'applicazione ha un design intuitivo ed essenziale, adatto a ogni tipo di utente. La schermata principale è la home, dalla quale è possibile avere subito una panoramica di tutte le funzionalità offerte. Nella parte superiore è possibile osservare un resoconto del numero di report aggiunti nel corso della giornata e la media dei parametri aggiunti nei report giornalieri, se inseriti. Al di sotto si presentano i bottoni che permettono all'utente di navigare all'interno dell'applicazione nelle activity *Calendario*, *Grafici*, *Filtri* e *Impostazioni*.

## Home

La home fornisce informazioni sulla giornata, ovvero il numero di report inseriti e la media dei parametri aggiunti nella giornata corrente. Se non è presente nessun valore viene mostrato il valore di default "0.0". Toccando la view "*Clicca per aggiungere un nuovo report!*" si aprirà l'activity che permette di inserire nuovi report nell'applicazione. Una volta inseriti i dati si riapre la home che vedrà il contatore dei report e la media dei parametri aggiornati.

Nella "MainActivity" vengono creati i ViewModel, che verranno utilizzati nell'intera applicazione per interagire col database, i listener dei bottoni, che permettono di accedere alle varie



Schermata Home

activity tramite intent, e le funzioni, che richiamano le classi asincrone (AsyncTask), sia all'apertura dell'app che nella chiamata della `onRestart()`. Quest'ultime mantengono aggiornate le informazioni sui report giornalieri ogni volta che si riattiva la Main Activity.

Se l'applicazione viene aperta per la prima volta, viene eseguita la funzione `firstRun()` che permette di inserire le tuple nel database sempre tramite AsyncTask e l'inizializzazione delle shared preferences utilizzate per la gestione delle notifiche.

## Calendario

La CalendarActivity permette di visualizzare e gestire i report all'interno di un calendario. Una volta aperta viene visualizzato il mese corrente e un segnalino per ogni giorno in cui è presente almeno un report. Cliccando su una data qualsiasi è possibile visualizzare la media dei valori dei parametri inseriti nei report della data selezionata. È presente un bottone, "Report", che permette di mostrare la lista di tutti i report giornalieri inseriti e di gestirli. In questa lista sono presenti i parametri specifici di ogni report e la nota aggiunta dall'utente.

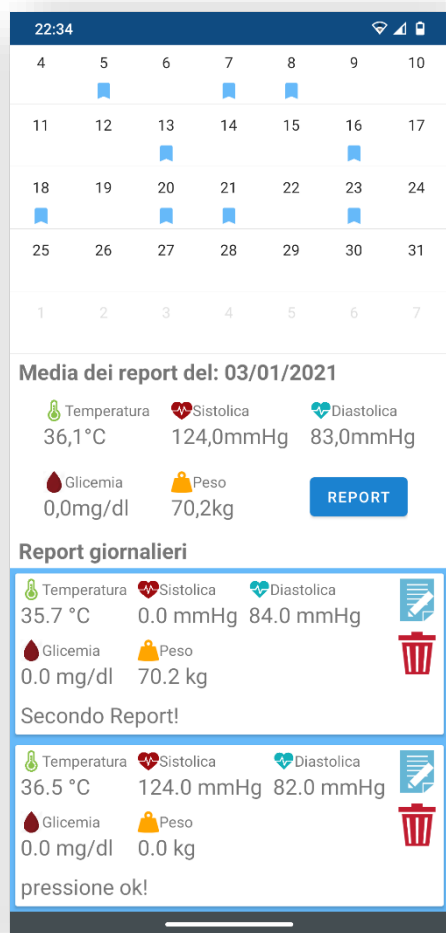
È possibile modificare il report, premendo l'ImageView della modifica, o cancellarlo, premendo l'ImageView del bidone. Cliccando la prima view si apre l'activity di modifica del report selezionato (discussa nella sezione di aggiunta e modifica dei report). Se si decide di cancellare il report, o lo si cancella per errore, è possibile tornare sui propri passi annullando l'operazione tramite uno snackbar che compare in sovraimpressione. Nel momento in cui i report vengono eliminati o modificati, la lista dei report si aggiorna automaticamente così come la media dei parametri. Se in una data non sono presenti report viene mostrato il valore di default "0.0".

Cliccando una data del calendario evidenziata dal segnalino azzurro, è possibile vedere la media dei parametri dei report aggiunti in quella giornata.

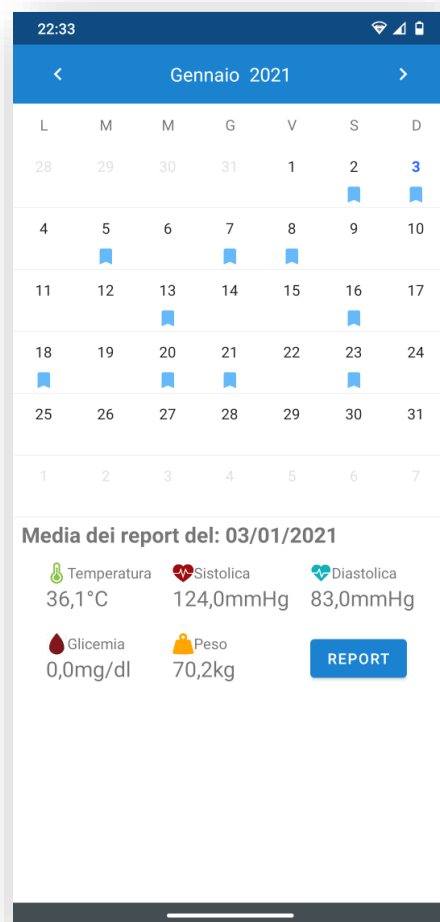
Cliccando il bottone "Report" viene fatta una query al database che ritorna una lista di report nel formato `LiveData<List<Info>>`. Questo formato permette di osservare i dati: se vengono apportati cambiamenti ai report ritornati (come modifiche dei valori o eliminazione del report stesso) è possibile mantenere aggiornata la media giornaliera, la lista dei singoli report mostrati e, nel caso vengano eliminati tutti i report in una giornata, viene eliminato il segnalino nella giornata corrispondente.

Per mostrare la lista dei report è stata utilizzata la classe `InfoListAdapter` che estende la `RecyclerView.Adapter`. Questa classe permette di inserire uno specifico insieme di dati dinamico, ovvero i report, dentro la `RecyclerView`.

All'interno della classe `InfoListAdapter` è definita una classe modificata, la `InfoViewHolder` che estende la `RecyclerView.ViewHolder`. Grazie a questa classe è possibile inserire i valori dei report recuperati dal database nelle View e inizializzare i listener per le ImageView che cancellano il report e che lo modificano.



**Schermata Calendario**



**Schermata Calendario  
con la lista dei report**

## Inserimento e modifica dei report

Come accennato precedentemente, toccando la view *"Clicca per aggiungere un nuovo report!"* si ha la possibilità di accedere a una nuova activity, *NewInfoActivity*, che permette di inserire un nuovo report nel database contenente informazioni sulla salute e annotazioni varie. L'utente potrà inserire i report solo nella data corrente e quest'ultima non potrà essere modificata in nessuna situazione.

Premendo il bottone *"Salva"*, vengono eseguiti i controlli per il corretto inserimento dei valori. Ogni report dovrà contenere almeno due parametri compresi fra:

- Temperatura
- Pressione Sistolica
- Pressione Diastolica
- Indice Glicemico
- Peso

e questi dovranno essere compresi tra un intervallo prestabilito (vedi sezione Database).

Allo stesso modo, nella `EditInfoActivity` si potranno modificare le informazioni aggiunte in un report specifico.

Se i parametri inseriti sono corretti, il report viene inserito nel database in modo asincrono tramite `AsyncTask`. Se l'utente sbaglia l'inserimento dei parametri, verrà notificato il tipo di errore effettuato tramite `Toast`. Nel caso in cui i campi delle `EditText` vengano lasciati vuoti, viene inserito automaticamente il valore "0.0".



09:15

Temperatura

Pressione Sistolica

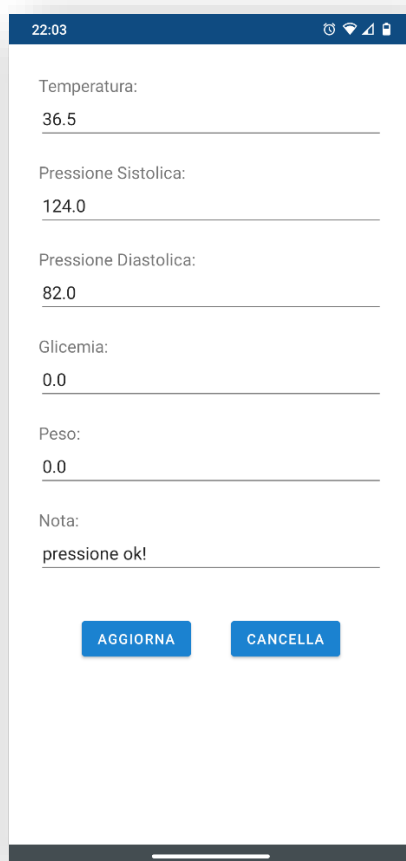
Pressione Diastolica

Indice Glicemico

Peso

Nota

SALVA



22:03

Temperatura:  
36.5

Pressione Sistolica:  
124.0

Pressione Diastolica:  
82.0

Glicemia:  
0.0

Peso:  
0.0

Nota:  
pressione ok!

AGGIORNA CANCELLA

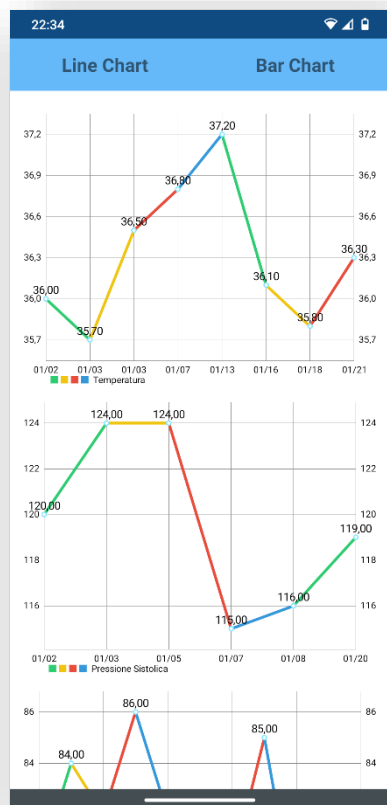
**A destra la schermata per l'aggiunta del report**

**A sinistra la schermata per la modifica del report**

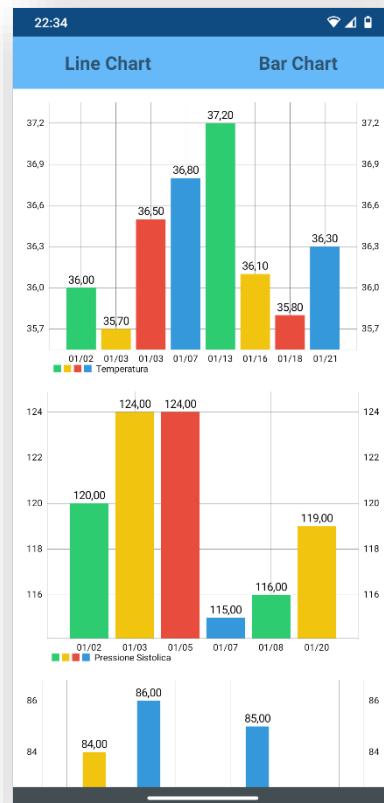
## Grafici

La `GraphActivity` permette di monitorare la tendenza dei valori aggiunti dall'utente. Avremo quindi cinque diversi grafici che mostrano i dati suddivisi per categoria (Temperatura, Pressione Sistolica, Pressione Diastolica, Indice Glicemico, Peso). Cliccando la view corrispondente sarà possibile visualizzare o un grafico lineare o un istogramma. La scelta è ricaduta su questi tipi di grafici poiché sono i migliori nel mostrare la variazione dei parametri nel corso del tempo.

I grafici mostrano sull'asse delle ascisse la data in cui è stato inserito il report nel formato "MM/dd". Per implementare questi grafici è stata usata una libreria esterna che ha permesso una maggiore personalizzazione.



Schermata Grafo Lineare



Schermata Istogramma

## Filtri

La DataFilterActivity si occupa di mostrare i report in base a due tipi di filtri: l'importanza dei report - che può variare da 1 a 5 - e la data di aggiunta dei report stessi. Una volta settati i filtri, verranno mostrati i report che hanno i valori con un'importanza maggiore o uguale a quella inserita. Dato che il valore di questa priorità viene scelto nelle impostazioni dell'applicazione e l'utente potrebbe non ricordare quali siano le priorità assegnate ai parametri, vengono mostrati i parametri che hanno un valore maggiore o uguale a quello scelto nella AlertDialog.

Una volta cliccato il bottone "Mostra i report" viene visualizzata la RecyclerView descritta nella sezione Calendar Activity.

Per recuperare le informazioni dal database è stata utilizzata una RawQuery poiché i dati che devono essere restituiti variano in base alla priorità dei parametri e quest'ultimi hanno priorità diverse in base alle esigenze dell'utente.

Nel codice sottostante viene mostrata la creazione dinamica della query:

```
public LiveData<List<Info>> getReportFromFilter(List<String> settingsList,
String begin, String end){
```

```

// Creo query
String new_query = "SELECT * FROM report WHERE (";

// Inserisco i parametri
if(settingsList.size() > 0) {
    for (int i = 0; i < settingsList.size(); i++) {
        new_query = new_query + settingsList.get(i) + "!=0";

        //aggiunta dell'OR nella query
        if (i != (settingsList.size() - 1)) {
            new_query = new_query + " OR ";
        }
    }
} else return infoDao.getAllInfoInDate("0");

new_query = new_query + ") AND (infoData>='" + begin + "' AND infoData<='" + end + "')";

// Passo la new_query che ho appena creato
return infoDao.getReportFiltered(new SimpleSQLiteQuery(new_query));
}

```

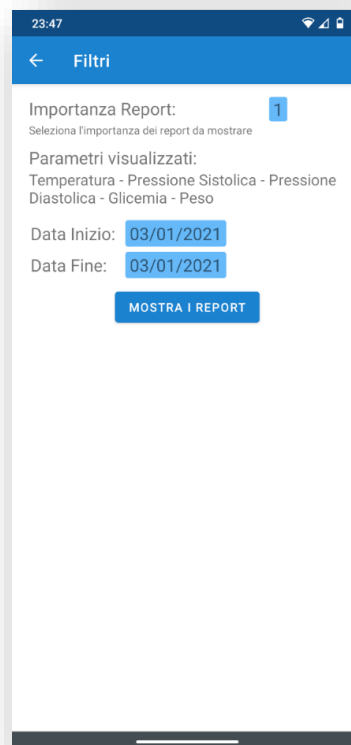
Dichiarazione della RawQuery nel Dao:

```

@RawQuery(observedEntities = Info.class)
LiveData<List<Info>> getReportFiltered(SupportSQLiteQuery query);

```

Viene passato in input la lista dei parametri con importanza desiderata insieme alle date inserite dall'utente che comprendono il periodo che l'utente vuole coprire. Questi parametri sono inseriti in una stringa che verrà trasformata in una query attraverso `new SimpleSQLiteQuery()`. La query appena creata viene passata in input alla `RawQuery getReportFiltered()`. È quindi possibile avere una query dinamica, che varia in base alla priorità dei parametri.



**Schermata Filtri  
all'apertura dell'Activity**



**Schermata Filtri  
all'apertura dell'Activity**

## Impostazioni e notifiche

La `SettingsActivity` permette di impostare le notifiche dell'applicazione. L'applicazione dispone di due tipologie di notifiche: la notifica giornaliera e la notifica per il monitoraggio dei parametri.

Nella parte superiore si può decidere se attivare o disattivare la notifica giornaliera interagendo con il Toggle. Se questo è settato su ON viene visualizzato un layout che permette di scegliere l'orario di notifica.

Al di sotto delle notifiche giornaliere si possono settare le preferenze di monitoraggio. A ogni parametro deve essere assegnato un valore da 1 a 5 attraverso una SeekBar. Se la priorità del valore scelto nella SeekBar è maggiore o uguale a 3, il parametro viene monitorato.

Quindi quando l'utente assegna un valore maggiore o uguale a 3 ad un certo parametro, l'utente dovrà anche inserire un intervallo di tempo durante il quale la media dei parametri non deve superare i limiti scelti.

Nel momento in cui l'utente clicca il bottone *"Update"*, vengono fatti dei controlli per verificare se le notifiche devono essere inviate. In caso affermativo, vengono creati degli Intent con una stringa extra che specifica il tipo di notifica che deve essere inviata all'utente.

Per esempio, per la notifica giornaliera viene inserito:

```
broadcastIntent.putExtra(NOTIFICATION_TYPE, TIMED_NOTIFICATION);
```

Vengono inoltre creati dei PendingIntent che sono passati all'AlarmManager. I PendingIntent sono utili per attivare il BroadcastReceiver che invierà la notifica specifica. Nella sezione seguente verranno spiegate le notifiche nel dettaglio.

## Notifica giornaliera

Se attiva, la notifica giornaliera risveglia il BroadcastReceiver ogni 24 ore nell'orario indicato dall'utente e gli ricorda di aggiungere un report. Nel caso in cui l'utente aggiunga un report prima dell'orario d'invio di notifica, viene settato il valore nelle SharedPreferences `NEW_REPORT=false` (questo avviene nella `NewInfoActivity`, l'activity per l'inserimento del nuovo report). Il BroadcastReceiver legge questo valore e decide se inviare o meno la notifica giornaliera.

Quando l'utente riceve la notifica può decidere di aggiungere un report. Cliccando *"Add"* presente nella notifica, l'utente viene indirizzato alla `NewInfoActivity`. È anche possibile



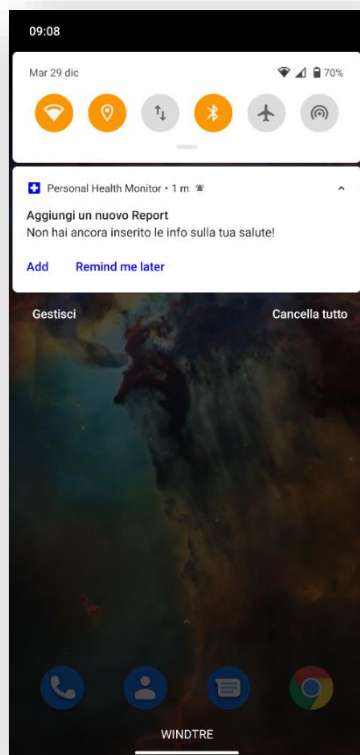
rimandare la notifica; premendo *"Remind Me Later"* si viene reindirizzati all'activity *RemindLater* che permette di scegliere il nuovo orario di notifica.

## Notifica di monitoraggio

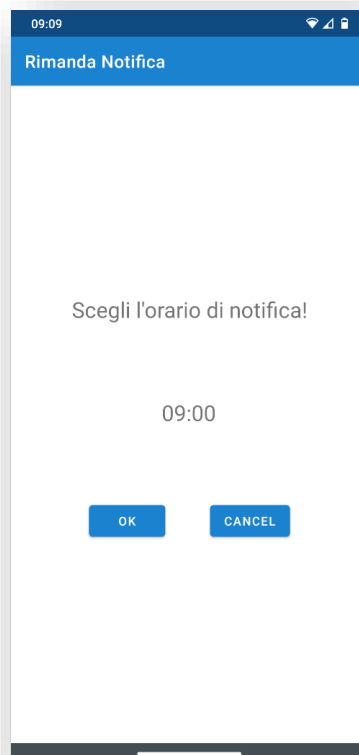
Nel momento in cui viene settata la notifica di monitoraggio, si crea un *PendingIntent* che risveglia il *BroadcastReceiver* ogni 24 ore. Il *BroadcastReceiver* riconosce l'intent ricevuto grazie alla stringa aggiunta che specifica il tipo di notifica (*MONITOR\_NOTIFICATION*) e avvia un *Service* in foreground. Quest'ultimo accede al database in un thread in background e controlla che i parametri, se hanno una priorità maggiore o uguale a 3, rispettino i rispettivi threshold. Quando i parametri non rientrano nell'intervallo specificato dall'utente, viene settata una variabile condivisa a *true* - per esempio *EXCEEDED\_TEMP=true* se la media della temperatura sfora le soglie.

Una volta finito il controllo nel *Service* viene settato un nuovo *PendingIntent* che richiama il *BroadcastReceiver* (con il rispettivo valore che specifica il tipo di notifica: *PARAMETERS\_OUT\_BOUND\_NOTIFICATION*) alle ore 9 della mattina. Dentro il *BroadcastReceiver* viene deciso se la notifica deve essere inviata. Se almeno uno dei valori delle *Shared Preferences* relativi al superamento del threshold è positiva, allora viene inviata la notifica di allerta all'utente.

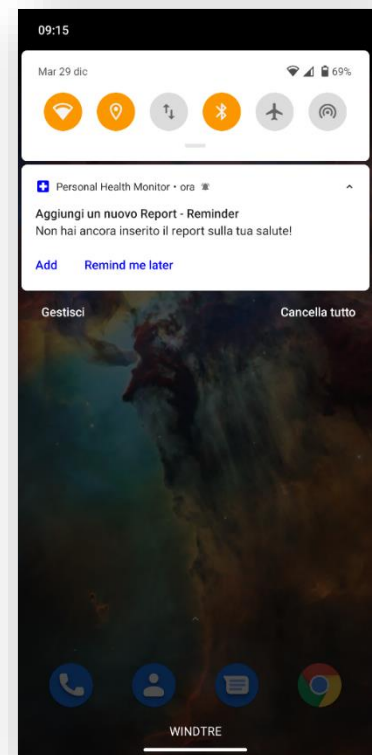
La scelta è ricaduta sull'utilizzo dei *Service* in foreground in quando sono gli ultimi processi che vengono terminati in caso di necessità di memoria da parte del dispositivo. C'è quindi una maggiore probabilità che il *Service* venga eseguito correttamente senza interruzioni improvvise.



Notifica Giornaliera

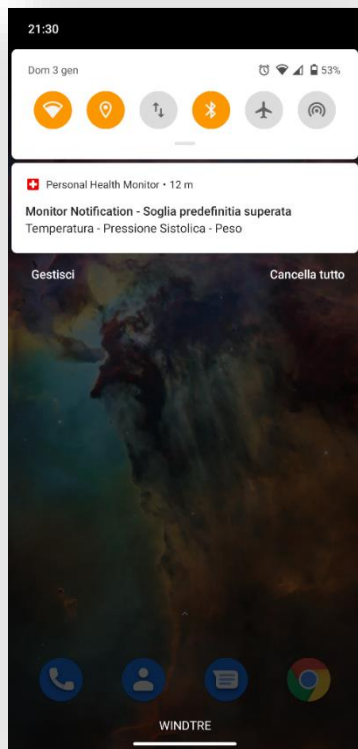


Schermata Reminder

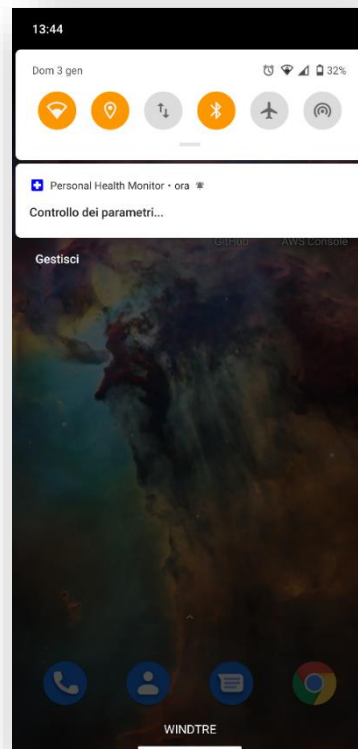


Notifica Giornaliera  
Reminder





**Notifica di Monitoraggio**



**Notifica Foreground**

## Database

Valutando la quantità e il tipo di dati che devono essere gestiti in un'applicazione che si occupa del monitoraggio dei valori della salute, si è deciso di utilizzare un database per il salvataggio dei dati. Room Database è una libreria che fornisce un livello di astrazione al di sopra di SQLite e permette di accedere facilmente al database sfruttando al pieno le capacità di SQLite.

Le tre componenti più importanti di Room Database sono:

- La classe database, `InfoRoomDatabase`, nella quale viene istanziato e controllato la corretta inizializzazione del DB, il quale deve essere un singleton.
- Le entità dati, corrispondenti alle tabelle del DB, `Info` e `Settings`.
- Il DAO (data access object), `InfoDao`, interfaccia che fornisce le query per svolgere operazioni all'interno del DB.

Per implementare i metodi definiti nell'interfaccia DAO sono stati utilizzati due ViewModel differenti, `InfoViewModel` e `SettingsViewModel`, che contengono le funzioni usate per interagire col database e gestire i dati al suo interno. Inoltre, i ViewModel permettono di osservare e acquisire le informazioni necessarie alle activities per aggiornare le UIs, se modificate all'interno del database, tramite l'utilizzo dei LiveData. Questa funzionalità è stata utile per implementare tutte quelle features

dell'applicazione che si basano sul mostrare dati dopo una modifica dei report nel database (come nella Recyclerview che aggiorna la lista di report se avvengono modifiche o eliminazioni).

## Diagramma delle entità

report		settings	
id	String Primary Key	id	Int Primary Key
infoTemperatura	double	parameter	String
infoPressioneSistolica	double	value	int
infoPressioneDiastolica	double	lower_bound	double
infoGlicemia	double	upper_bound	double
infoPeso	double	begin_date	String
infoNota	String	end_date	String
infoData	String		

## Report

L'entità `report` rappresenta appunto il report che l'utente ha la possibilità di aggiungere nel corso della giornata per monitorare i parametri riguardanti la salute. Questi devono essere compresi in un intervallo specifico per poter essere aggiunti al database.

`String id`: è una stringa generata in maniera casuale nel momento dell'inserimento del report nel database e rappresenta la Primary Key della tabella.

`Double infoTemperatura`: compresa fra 34°C - 42°C.

`Double infoPressioneSistolica`: compresa fra 80 mmHg - 190 mmHG

`Double infoPressioneDiastolica`: compresa fra 50 mmHg - 120 mmHG

`Double infoGlicemia`: compresa fra 50 mg/dl - 250 mg/dl

`Double infoPeso`: maggiore di 0.0 Kg

`String infoNota`: una stringa che contiene eventuali annotazioni dell'utente.

`String infoData`: una stringa nella quale viene salvata la data in cui viene aggiunto il report. Non può essere cambiata dall'utente ed è nel formato "MM/dd/yyyy".

## Settings

La tabella dei `settings` viene creata nel momento in cui si accede all'applicazione per la prima volta. Vengono inserite cinque righe, una per ogni parametro monitorato dall'applicazione. Nella `SettingsActivity`, l'utente può assegnare a ogni parametro un valore da 1 a 5 che definisce la sua importanza, con 5 come valore di priorità maggiore. Se il `value` è maggiore o uguale a 3 allora il parametro viene monitorato dall'applicazione.

`Int id`: intero che rappresenta la Primary Key della tabella

`String Parameters`: assume il valore del nome della colonna di uno dei parametri della tabella `reports` (`infoPeso`, `infoGlicemia` etc.).

`Int value`: importanza assegnata dall'utente nella `SettingsActivity`, se il valore supera 3 allora il parametro corrispondente viene monitorato dall'applicazione.

`Double lower_bound, upper_bound`: valori per il monitoraggio del parametro, se questo bound viene sfiorato, l'applicazione manda una notifica all'utente.

`String begin_date, end_date`: stringhe per il monitoraggio del parametro, rappresentano il periodo di inizio e fine monitoraggio. Il formato è "MM/dd/yyyy".

## InfoRoomDatabase

`InfoRoomDatabase` è la classe che definisce il database. Quando viene creato un `ViewModel`, questo utilizza la funzione `getMyAppDatabase()` della classe `InfoRoomDatabase` per restituire l'istanza del database. È proprio in questa funzione che si controlla che il database sia un singoletto, ovvero che non vengano create molteplici istanze. Se il database non è mai stato creato, allora si provvede a istanziarne uno.

## Utils e UndoAction

`Utils`, `UtilsValue` e `UtilsMain` sono delle classi create per raggruppare funzioni di uso comune in tutta l'applicazione. In `Utils` sono raggruppate le funzioni che si occupano di gestire il tempo ed è presente la funzione che controlla l'inserimento e la modifica dei valori rispettivamente nella `NewInfoActivity` e nella `EditInfoActivity`. In `UtilsValue` sono definite tutte le stringhe comuni e in `UtilsMain` sono definite le funzioni chiamate alla first run dell'applicazione e le classi asincrone che permettono di mantenere la `MainActivity` aggiornata.

La `UndoAction` permette di annullare l'eliminazione del report, il quale può essere cancellato nella `RecyclerView` mostrata sia nella `CalendarActivity` che nella `DataFilterActivity`. Nel caso in cui l'utente cambi idea o cancelli un report per errore, può decidere di tornare indietro sui suoi passi cliccando "Annulla" nello `Snackbar` che compare in fondo alla schermata.

## Scelte implementative

- La stringa che nel database si occupa di salvare la data è nel formato MM/dd/yyyy, tipicamente non italiano. È stato scelto questo formato perché ha permesso di sfruttare al meglio le query. In questo modo si è potuto richiedere al database dei dati già ordinati in base alla data ove necessario. Questo non sarebbe stato possibile col più tradizionale formato dd/MM/yyyy.
- In un report aggiunto, che presenta dei campi non riempiti, viene automaticamente inserito il dato di default "0.0".
- Per svolgere query asincrone e non bloccare il main thread, la scelta è ricaduta sugli AsyncTask in quanto sono le operazioni più adeguate se è necessario svolgere operazioni in background che mirano a cambiare la UI al momento della terminazione.
- Come già descritto precedentemente, per il monitoraggio dei valori è stato scelto il Service in foreground, poiché in questo modo l'utente è consapevole del lavoro svolto dall'applicazione.
- I colori per l'applicazione sono stati diverse gradazioni di azzurro, poiché creano un buon contrasto con il bianco dello sfondo.

## Monte ore

Per completare lo sviluppo dell'applicazione sono state impiegate un totale di 220 ore.

## Librerie esterne

### Calendario - Material Calendar View

Material-Calendar-View è un widget semplice e personalizzabile per applicazioni Android basato su Material Design. Per questo progetto è stata utilizzata la funzionalità del calendario nella CalendarActivity per mostrare all'utente i report inseriti.

Repository GitHub: <https://github.com/Applandeo/Material-Calendar-View>

### Grafici - MPAndroidChart

MPAndroidChart è una libreria semplice da utilizzare che implementa una vasta quantità di grafici. Grazie a questa libreria è stato possibile implementare efficacemente i grafici della GraphActivity.

Repository GitHub: <https://github.com/PhilJay/MPAndroidChart>

## Software per lo sviluppo del codice:

**Android Studio** - ambiente di sviluppo integrato per lo sviluppo per la piattaforma Android.

**GitHub** - servizio di hosting utilizzato per il salvataggio del codice.

## Fonti:

GitHub ( <https://github.com/> ) - servizio di hosting per progetti grazie al quale si è potuto trovare widgets da includere nell'applicazione.

Stack Overflow ( <https://stackoverflow.com/> ) - sito nel quale sono state trovare risposte a problemi comuni incontrati durante lo sviluppo.

Developer Android ( <https://developer.android.com/> ) - documentazione ufficiale per sviluppatori di applicazioni Android.