POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Studio e sviluppo di un'interfaccia mobile per smartwatch con applicazione medicale

Relatori
Prof. Eros PASERO
Ing. Vincenzo RANDAZZO

Candidati
Como Giuseppe
Schiopu Andrei Stefan

Aprile 2022

Indice

Capitolo 1

1. Introduzione

1.1 Premessa

Con l'avvento della pandemia, i ricoveri per molte patologie sono passati in secondo piano per l'impossibilità di recarsi in ospedale.
E' qui che entra in campo la telemedicina, cioè la possibilità di fornire prestazioni mediche a distanza a pazienti "a rischio" o affetti da patologie gravi.
Per quanto riguarda le malattie cardiovascolari che sono molto frequenti, è emersa la necessità di disporre di strumenti compatibili con la telemedicina.
Con questa tesi ci si è concentrati sul telemonitoraggio, per monitorare a distanza pazienti con patologie cardiovascolari, anche a scopo preventivo, tramite la misurazione di parametri vitali (elettrocardiogramma, frequenza cardiaca, fotopletismografia, saturazione di ossigeno nel sangue, pressione sistolica e diastolica).
Tali parametri sono misurabili tramite un dispositivo in grado di acquisire i dati attraverso dei sensori e trasmetterli tramite bluetooth ad un'applicazione installata sul proprio dispositivo mobile. Tale applicazione permette di visualizzare ed elaborare i dati acquisiti consentendone l'invio al medico, il quale si occuperà di controllare lo stato di salute del paziente, potendo così intervenire in modo tempestivo in caso di necessità.

1.2 Fondamenti di Medicina

1.2.1 Elettrocardiogramma

Le contrazioni delle diverse parti del cuore dell'essere umano, il cui compito è quello di permettere la corretta circolazione del sangue all'interno del corpo, vengono coordinate da impulsi elettrici.
L'elettrocardiogramma (ECG) è un esame diagnostico che registra i sopracitati impulsi elettrici per poter mostrare la loro forza, la tempistica, la velocità con cui batte il cuore, il ritmo dei battiti, etc…
L'elettrocardiogramma, permette al cardiologo di rilevare:

- La presenza di aritmie cardiache;
- Un'ischemia o un infarto del miocardio;
- La presenza di alterazioni strutturali delle cavità cardiache, atri e/o ventricoli;
- Gli esiti di un precedente attacco di cuore;
- La presenza di condizioni cardiache, caratterizzate da un'alterazione della conduzione elettrica;
- Il funzionamento di pacemaker e dispositivi analoghi;
- Gli effetti sul cuore di quei farmaci che potrebbero alterare, in alcune circostanze, la frequenza o la conduzione elettrica del cuore.

(Griguolo)

Il funzionamento di tale esame si basa sulla misurazione delle differenze di potenziale derivanti dagli impulsi elettrici generati dal cuore tramite degli elettrodi.
Il tracciato ECG, derivante dalla rappresentazione del segnale acquisito, è caratterizzato da una sequenza di deflessioni positive e negative, denominate «onde», separate da alcuni tratti rettilinei, denominati «segmenti». La sequenza si ripete ad ogni ciclo cardiaco, come è possibile osservare nella Fig. 1.1. (Einthoven et al.)

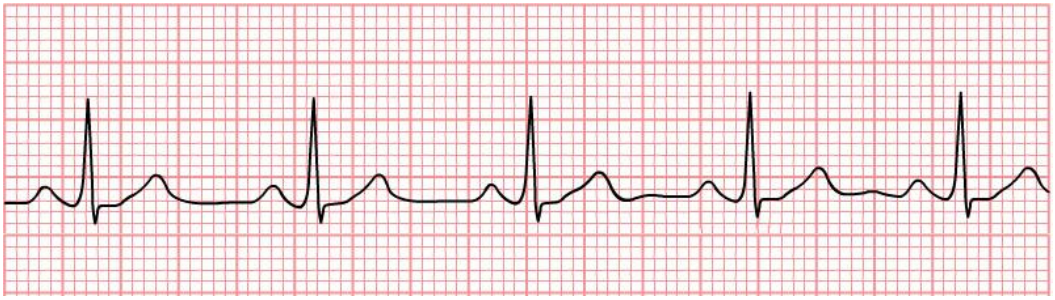


Fig. 1.1: Tracciato tipico di un ECG

1.2.1.1 Struttura della sequenza

Come si può notare dalla figura soprastante (Fig. 1.1) e da quella sottostante (Fig. 1.2), la sequenza periodica di deflessioni di una persona sana presenta 5 onde caratteristiche, identificate con le lettere maiuscole P, Q, R, S e T.
E' possibile monitorare ed individuare potenziali patologie mediante l'osservazione della forma di tali onde.

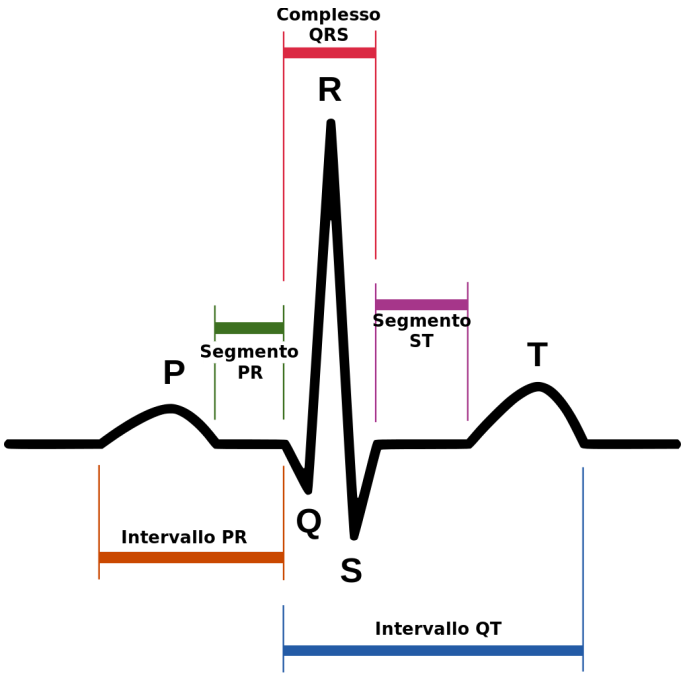


Fig. 1.2: Struttura tipica di un tracciato elettrocardiografico

1.2.1.2 Derivazioni

In un ECG completo a 12 derivazioni, quattro elettrodi (periferici) vengono posizionati su polsi e caviglie e sei (precordiali) sulla superficie del torace.
Quindi il potenziale elettrico complessivo del cuore viene misurato in dodici punti (derivazioni) e viene registrato per un determinato periodo di tempo.
Nello specifico, le 12 derivazioni si suddividono in:

- derivazioni bipolari degli arti: D_I, D_{II}, D_{III};
- derivazioni unipolari degli arti: aVR, aVL, aVF;
- derivazioni precordiali: V_1, V_2, V_3, V_4, V_5, V_6.

(Einthoven et al.)

Nelle derivazioni bipolari si usano due coppie di elettrodi, rispettivamente la prima coppia su polso destro e sinistro, la seconda coppia su caviglia destra e sinistra.
Congiungendo i punti di applicazione virtuale degli elettrodi si viene a formare un triangolo equilatero, noto come triangolo di Einthoven (Fig. 1.3), che ha il suo centro nel cuore.
Per queste derivazioni è necessario porre delle convenzioni:

- in D_I il tracciato va verso l'alto quando la spalla sinistra è positiva rispetto alla spalla destra;
- in D_{II} il tracciato va verso l'alto quando la gamba sinistra è positiva rispetto alla spalla destra;
- in D_{III} il tracciato va verso l'alto quando la gamba sinistra è positiva rispetto alla spalla sinistra.

Nei prossimi capitoli il tracciato ECG preso in esame riguarda la I derivazione (D_I); esso è in grado di fornire informazioni sul battito e ritmo cardiaco e permette di identificare una possibile fibrillazione atriale.

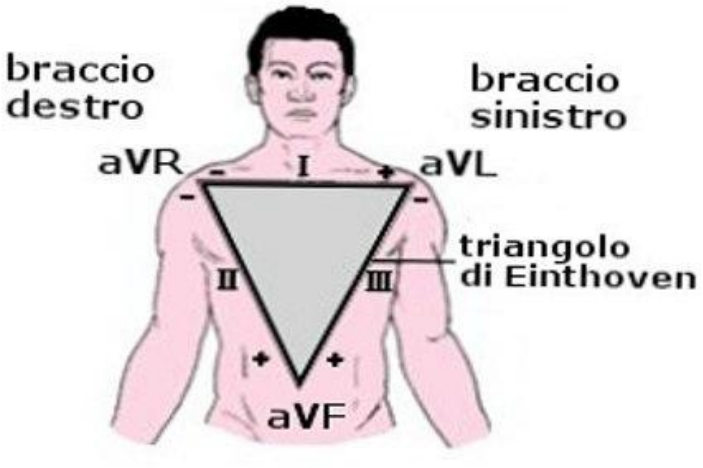


Fig. 1.3: Triangolo di Einthoven

1.2.1.3 Frequenza cardiaca

La frequenza cardiaca indica la velocità con cui il cuore si contrae e viene misurata in battiti al minuto (bpm).
In condizioni normali, la frequenza cardiaca è compresa nell'intervallo 60-100 bpm.
Nelle circostanze in cui si ha un valore superiore a 100 bpm si parla di tachicardia, mentre nel caso in cui si ha un valore inferiore a 60 bpm si parla di bradicardia.
Quando il cuore non batte con un andamento regolare si parla di aritmia. (Borgacci)

1.2.1.4 Fibrillazione atriale

La fibrillazione atriale è un'aritmia che rende il battito del cuore molto rapido e irregolare, come osservabile nella Fig. 1.4.
Viene provocata da una generazione anomala degli impulsi che contraggono gli atri del cuore.
L'elettrocardiogramma di una persona con fibrillazione atriale presenta le seguenti caratteristiche:

- Assenza di onde P. Questo denota il difetto di contrazione degli atri, tipico della fibrillazione atriale;

- Tratti rettilinei irregolari;
- Complessi QRS di forma irregolare.

(Griguolo)

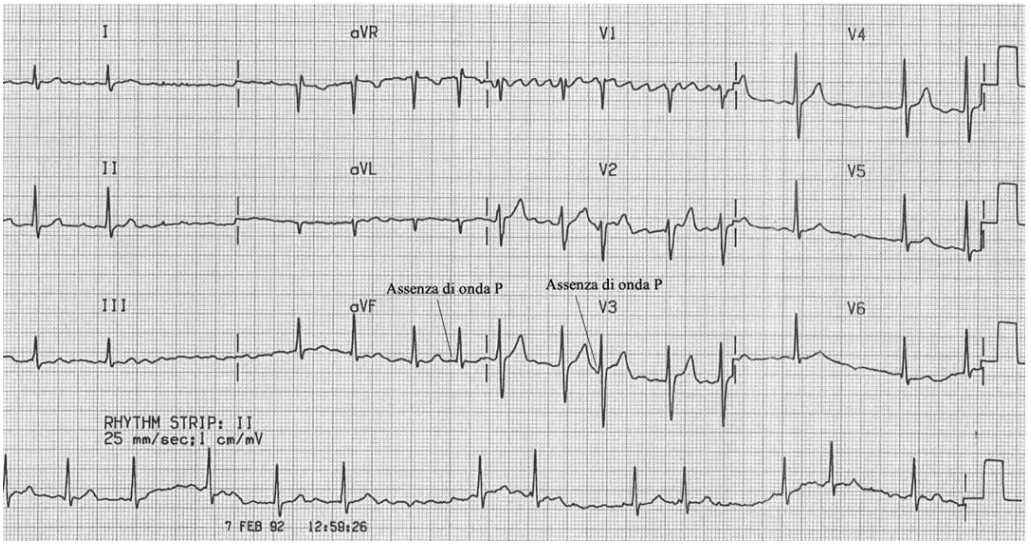


Fig. 1.4: Tracciato di una fibrillazione atriale

1.2.2 Fotopletismografia

La fotopletismografia (PPG) è una tecnica usata per visualizzare graficamente la variazione di volume del sangue nella circolazione periferica.
Viene eseguita per mezzo di un modulo ottico composto da una sorgente luminosa e un fotorilevatore (Fig. 1.5).
Quando viene appoggiato un dito della mano sul modulo ottico, la sorgente luminosa emette dei raggi che attraversano la superficie del dito e vengono assorbiti da muscoli, ossa e in maggior modo dal sangue che scorre nei vasi sanguigni.
La parte rimanente di luce viene riflessa e in seguito catturata dal fotorilevatore, il quale la converte in segnale elettrico.
In questo modo è possibile rilevare i cambiamenti nel flusso sanguigno in base all'intensità della luce. ("ECG/PPG Measurement Solution")

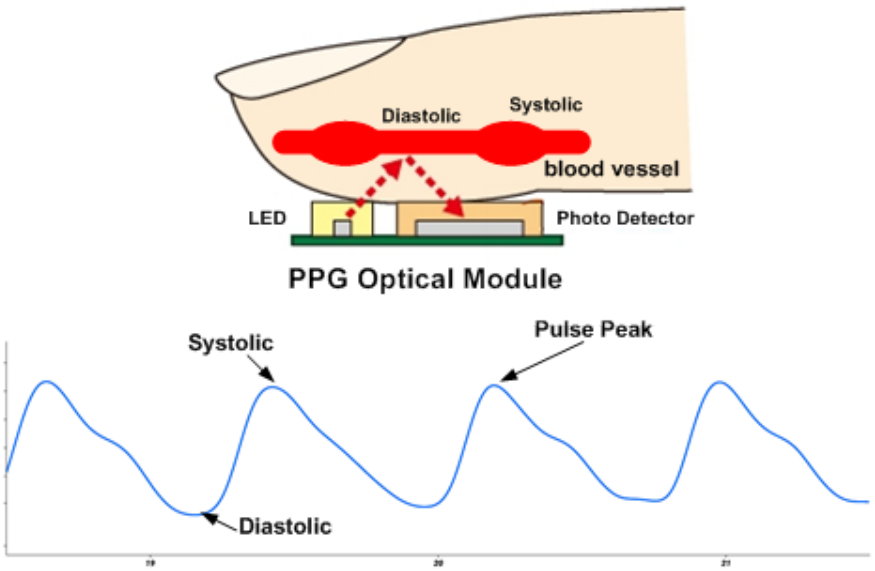


Fig. 1.5: Modulo ottico e tracciato tipico PPG

1.2.2.1 Saturazione di ossigeno nel sangue

La saturazione di ossigeno (SpO2) è un indice ematico che permette di stabilire la percentuale di emoglobina satura rispetto alla quantità totale di emoglobina presente nel sangue.
La saturazione di ossigeno si misura tramite un saturimetro (o pulsiossimetro), dalla forma simile a quella di una molletta (Fig. 1.6).
Questo dispositivo è composto da due sensori che emettono raggi luminosi di diversa lunghezza d'onda che comunicano con una fotocellula.
Tramite l'assorbimento della luce emessa dal dispositivo applicato su un dito è possibile stimare l'indice ematico.
I valori di saturazione di ossigeno considerati nella norma sono compresi tra 95% e 100%, per valori inferiori al 95% si parla di ipossia (condizione in cui il sangue contiene una quantità di ossigeno inferiore rispetto al normale), che può essere lieve (91%-94%), moderata (86%-90%) e grave (pari o inferiori all'85%). (Bertelli)

Fig. 1.6: Saturimetro da dito

1.2.3 Pressione arteriosa

La pressione arteriosa è quella forza con cui il sangue viene spinto attraverso i vasi sanguigni.
La scala di riferimento è il millimetro di mercurio (mmHg).
Ogni volta che il cuore si contrae, il sangue viene messo in circolo tramite l'aorta, il principale vaso arterioso.
Le pareti dell'aorta sono in grado di dilatarsi e contrarsi, in questo modo è possibile regolare in modo efficace la pressione del sangue.
La pressione massima (sistolica) dipende dalla quantità di sangue espulso a ogni contrazione e dall'elasticità delle pareti delle arterie.
In condizioni normali la pressione sistolica è 120 mmHg.
Una volta che il cuore si svuota inizia la fase di riempimento (diastole).
Durante questa fase la pressione nelle arterie diminuisce fino a raggiungere il suo valore minimo (pressione minima o diastolica).
La pressione diastolica dipende dalla resistenza che il sangue incontra nei tessuti periferici, in condizioni normali il suo valore è 80 mmHg.
("Pressione arteriosa, cos'è e come si misura")
Per poter tenere sotto controllo la pressione del sangue è possibile usare ad esempio lo sfigmomanometro digitale (Fig. 1.7), uno strumento composto da un manicotto costituito da una camera d'aria e da un apparecchio elettronico collegato ad esso tramite un tubo in gomma.

Una volta posizionato il manicotto attorno al braccio, è possibile avviare lo strumento tramite un apposito tasto; a questo punto il manicotto si gonfierà e successivamente tramite un sensore verranno raccolti i dati, trasferiti all'apparecchio elettronico ed una volta elaborati verranno mostrati i valori di pressione sistolica e diastolica sul display digitale.
(Ilaria)

Fig. 1.7: Sfigmomanometro digitale

Capitolo 2

2. Creazione dell'app EcgMed

2.1 Git

Git è un software per il controllo di versione distribuito che permette di far collaborare contemporaneamente più sviluppatori sullo stesso progetto. ("Git Guides")
Ogni volta che vengono effettuati dei cambiamenti sul progetto, possono essere salvati tramite un ID e una descrizione; tale operazione prende il nome di "commit".
Per gestire un progetto complesso è possibile creare diverse versioni, ognuna contenente i propri "commit".
Le diverse versioni prendono il nome di rami (branch) e rappresentano dei percorsi staccati da quello principale (master) del progetto.
L'utilizzo di diversi "branch" è utile per gestire flussi paralleli indipendenti, in modo da poter aggiungere nuove funzionalità e testare il loro funzionamento prima di unirle al ramo principale (master), tramite un'operazione che prende il nome di "merge".

2.1.1 GitHub

GitHub è una piattaforma web che offre un servizio di hosting per progetti software. ("GitHub")
Esso permette di sfruttare le funzionalità offerte da Git in modo che più sviluppatori possano collaborare allo sviluppo di un progetto e visionare in tempo reale le modifiche effettuate.
L'elemento fondamentale di GitHub è la "repository", un archivio contenente il codice e la documentazione relativa a un progetto.
Partendo da un "repository" è possibile, tramite un'operazione che prende il nome di "clone", creare una copia locale del codice relativo al progetto, in modo da poter effettuare modifiche.

Una volta effettuate e salvate le modifiche sul proprio repository locale, è possibile tramite un'operazione che prende il nome di "push", caricare il codice sul repository remoto in modo da renderlo disponibile ad altri sviluppatori.

2.1.2 Approccio feature branches

Per questo progetto è stato usato il modello di sviluppo "feature branches" in modo da creare diversi "branches" al cui interno è stato possibile sviluppare diverse funzionalità da aggiungere all'applicazione. ("Git Feature Branch Workflow")
Lo sviluppo basato sul modello "feature branches" consiste nell'utilizzare branches più piccoli per implementare funzionalità, quindi unirli in un ramo principale, solitamente il master.
Questa strategia passa dallo sviluppo su branches comuni di lunga durata a modifiche più piccole, fondendo rami singoli nella base di codice principale. Uno dei molti vantaggi di questa strategia è che riduce al minimo il cosiddetto "merge hell".
Lavorando, dunque, in parallelo su diversi task, una volta implementate e testate le funzionalità richieste, è stato possibile fare il "merge" di quest'ultime con il ramo principale (master).

2.2 Setup ambiente di sviluppo

L'ambiente di sviluppo utilizzato è stato l'IDE Android Studio. ("Android Studio")
Inizialmente, partendo dal repository remoto dell'applicazione PulsEcg, è stata creata una copia del codice in un nuovo repository remoto per realizzare l'applicazione EcgMed, in modo da poter distinguere le due applicazioni.
Successivamente è stato importato su Android Studio il codice dell'applicazione EcgMed, in modo da creare un repository locale, sul quale poter effettuare modifiche.

In particolare, per distinguere in modo univoco le due applicazioni, nel codice dell'applicazione EcgMed sono stati cambiati:

- il nome del "package", cioè il pacchetto principale che include tutti i file necessari per lo sviluppo dell'applicazione;
- l'ID, cioè l'identificativo che definisce in modo univoco l'applicazione.

("Configure the app module")

2.3 Dispositivo EcgMed

Il dispositivo EcgMed presente nella Fig. 2.1, è caratterizzato principalmente da due elettrodi, da un circuito per la misurazione dell'elettrocardiogramma e da una batteria, il tutto montato su un mockup stampato in 3D.
Il circuito è costituito da:

- un connettore degli elettrodi, un connettore USB e un connettore della batteria che formano l'interazione tra il PCB e il mondo esterno;
- un modulo Bluetooth 4.0 per la comunicazione dei dati ad altri dispositivi, in particolare per inviare i dati all'applicazione EcgMed sviluppata da questa tesi;
- blocchi di alimentazione:
    - Battery Charger e Battery Gauge entrambi usati per controllare lo stato della batteria e la sua modalità di carica
    - Voltage Regulator e Analog Power Domain, che generano una tensione di alimentazione stabile e una tensione di riferimento per tutti gli altri componenti
- filtri utilizzati per il condizionamento del segnale elettrocardiografico;
- un microcontrollore Texas Instruments CC2640R2F per controllare tutti gli altri blocchi e raccogliere i dati dell'elettrocardiogramma attraverso il suo convertitore analogico-digitale (ADC) interno e il valore di carica della batteria attraverso la comunicazione I2C.

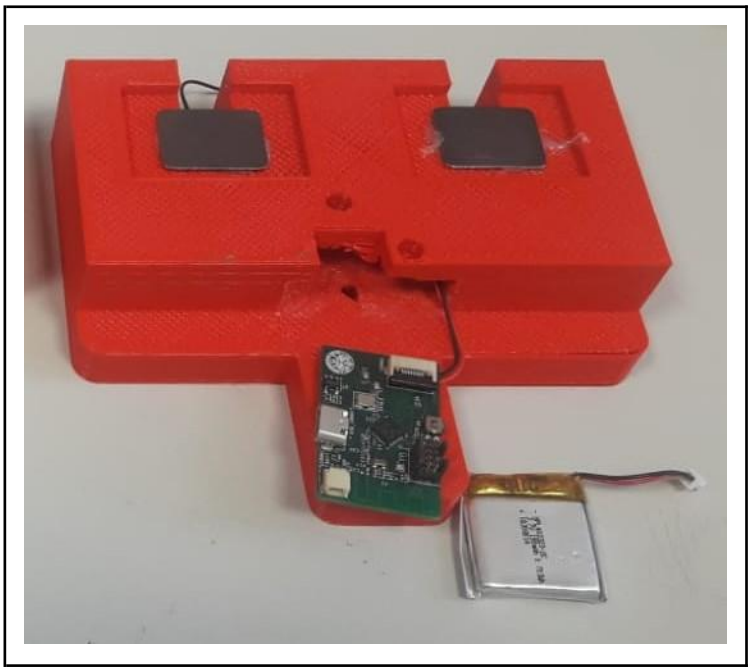


Fig. 2.1: Dispositivo EcgMed

2.4 Modifica app EcgMed

Una volta effettuato il setup iniziale, per creare l'applicazione EcgMed sono state rimosse alcune funzionalità; nello specifico è stata rimossa la parte riguardante l'acquisizione e l'elaborazione della fotopletismografia (PPG), la parte riguardante il calcolo della saturazione di ossigeno nel sangue e la parte riguardante il calcolo della pressione sistolica e diastolica.
E' possibile osservare le modifiche effettuate nelle Figg. 2.2 e 2.3, dove nell'applicazione EcgMed non sono più presenti i riquadri con le informazioni sulla saturazione di ossigeno nel sangue e sulla pressione sistolica e diastolica.

Fig. 2.2:
A sinistra la schermata di riepilogo dell'applicazione PulsEcg,
a destra la schermata di riepilogo dell'applicazione EcgMed

Nella Fig. 2.3, sul lato sinistro (PulsEcg), si evidenzia dall'alto verso il basso la schermata di visualizzazione dell'ecg e della frequenza cardiaca, schermata di visualizzazione del ppg e della saturazione di ossigeno nel sangue, schermata di visualizzazione della pressione sistolica e diastolica, mentre sul lato destro la versione EcgMed nella quale viene visualizzato esclusivamente l'ecg e la frequenza cardiaca.

Fig. 2.3:
A sinistra sono mostrate le 3 schermate di visualizzazione dettagli dell'applicazione PulsEcg. A destra è mostrata la schermata di visualizzazione dettagli dell'applicazione EcgMed

Le informazioni sono condivisibili tramite file JSON (formato di testo ideale per lo scambio dei dati) o tramite file PDF (formato di file per poter rappresentare documenti di testo e immagini). Le Figg. 2.4 e 2.5 mostrano un esempio di file PDF ottenuto rispettivamente dalle due applicazioni.

Fig. 2.4: PDF generato dall'applicazione PulsEcg

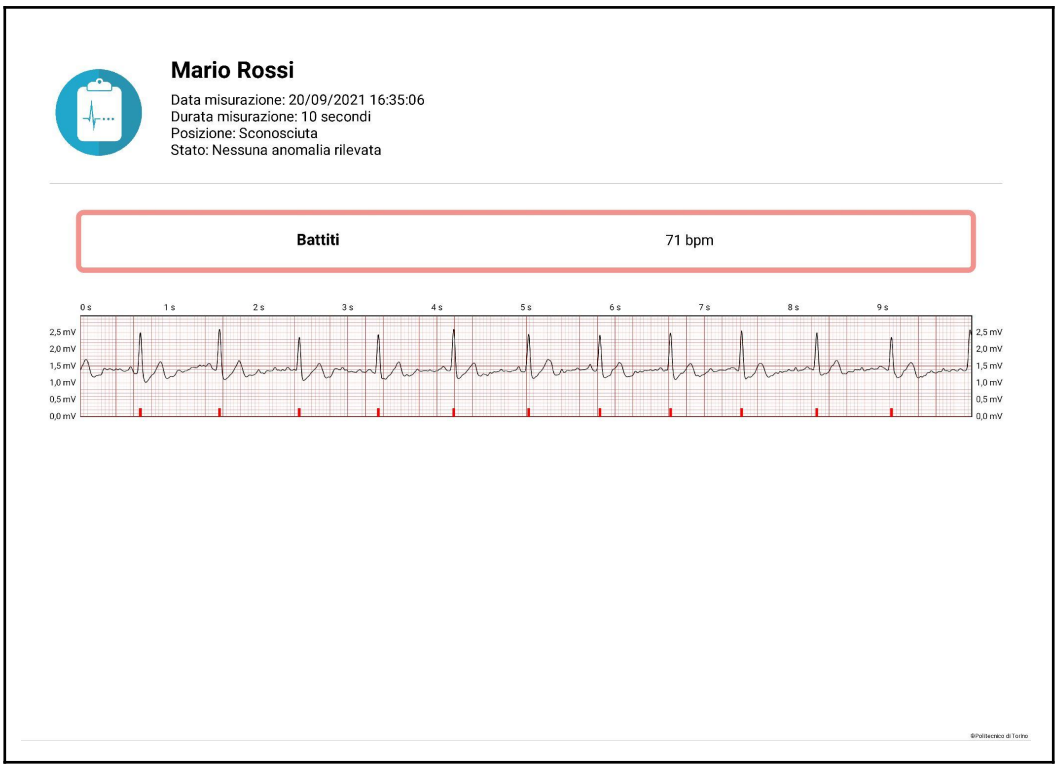


Fig. 2.5: PDF generato dall'applicazione EcgMed

Dopo aver effettuato un redesign dell'applicazione (come dettagliato nel successivo capitolo 3), all'apertura dell'applicazione è stata aggiunta una finestra di dialog (utile all'utente per effettuare una scelta) contenente alcune informazioni essenziali sull'utilizzo dei dati da parte dell'applicazione e un link all'informativa sulla privacy. ("Privacy policy per app Android")
La finestra è bloccante, quindi l'utente dopo aver letto l'informativa sulla privacy deve accettare i termini per proseguire all'utilizzo dell'applicazione.
E' possibile visionare la finestra di dialog nella Fig. 2.6.

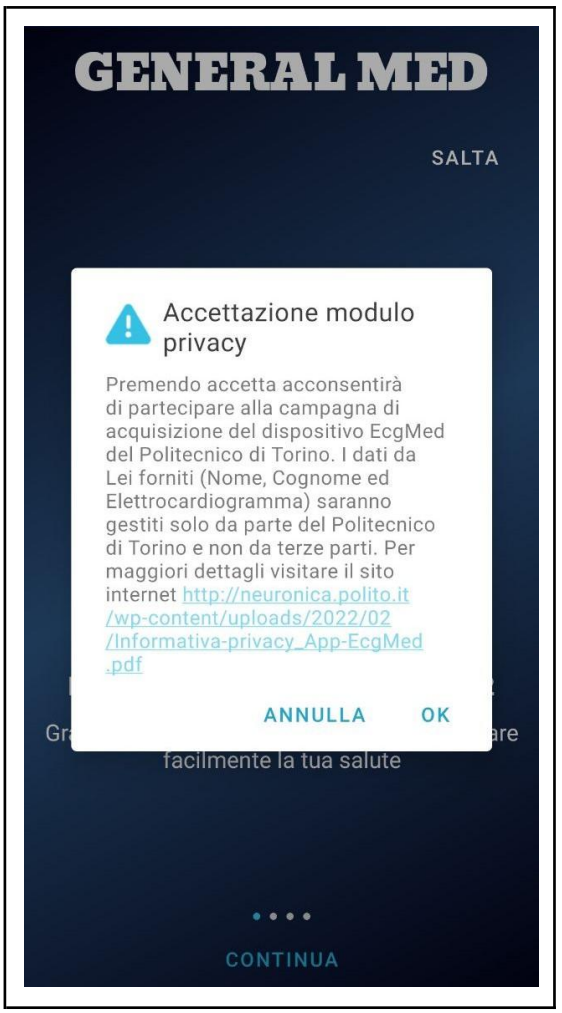


Fig. 2.6: Finestra di dialog riguardo l'informativa sulla privacy

Inoltre sono stati aggiunti messaggi per segnalare all'utente casi di bradicardia o tachicardia.
In particolare, nella schermata di riepilogo, viene mostrato un messaggio con scritto "Bradicardia", di colore arancione nel caso in cui la frequenza cardiaca sia compresa tra 40 e 59 bpm e ciò per indicare una bradicardia lieve-moderata, di colore rosso nel caso in cui la frequenza cardiaca sia inferiore ai 40 bpm per indicare una bradicardia grave. ("Bradicardia - Frequenza cardiaca")
In alternativa viene mostrato nella schermata di riepilogo un messaggio con scritto "Tachicardia", di colore arancione nel caso in cui la frequenza cardiaca sia compresa tra 101 e 140 bpm per indicare una tachicardia lieve-moderata, di colore rosso nel caso in cui la frequenza cardiaca sia

superiore ai 140 bpm per indicare una tachicardia grave. ("Tachicardia: quando può essere pericolosa?")
Di conseguenza gli stessi colori vengono usati per mostrare la frequenza cardiaca nella schermata di visualizzazione dettagli.
Le Figg. 2.7, 2.8, 2.9 e 2.10 illustrano le diverse situazioni.

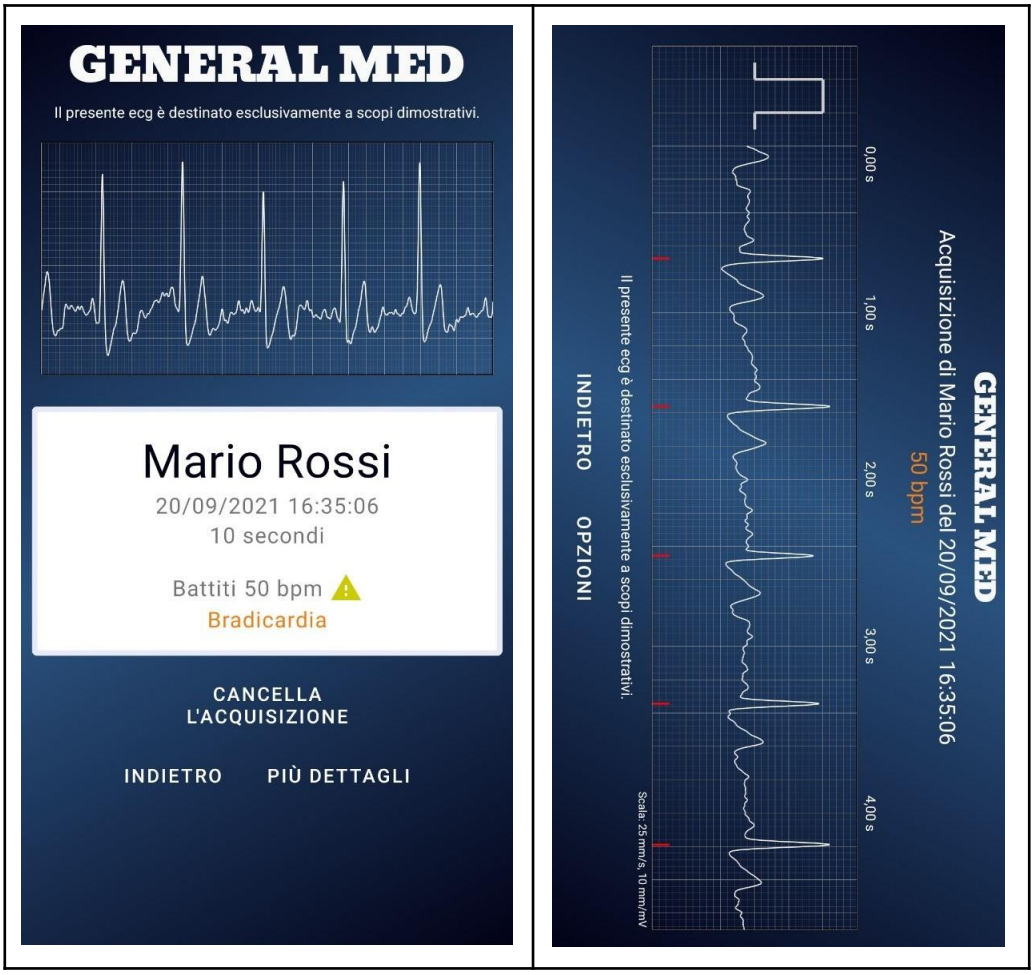


Fig. 2.7: Schermate con bradicardia lieve-moderata

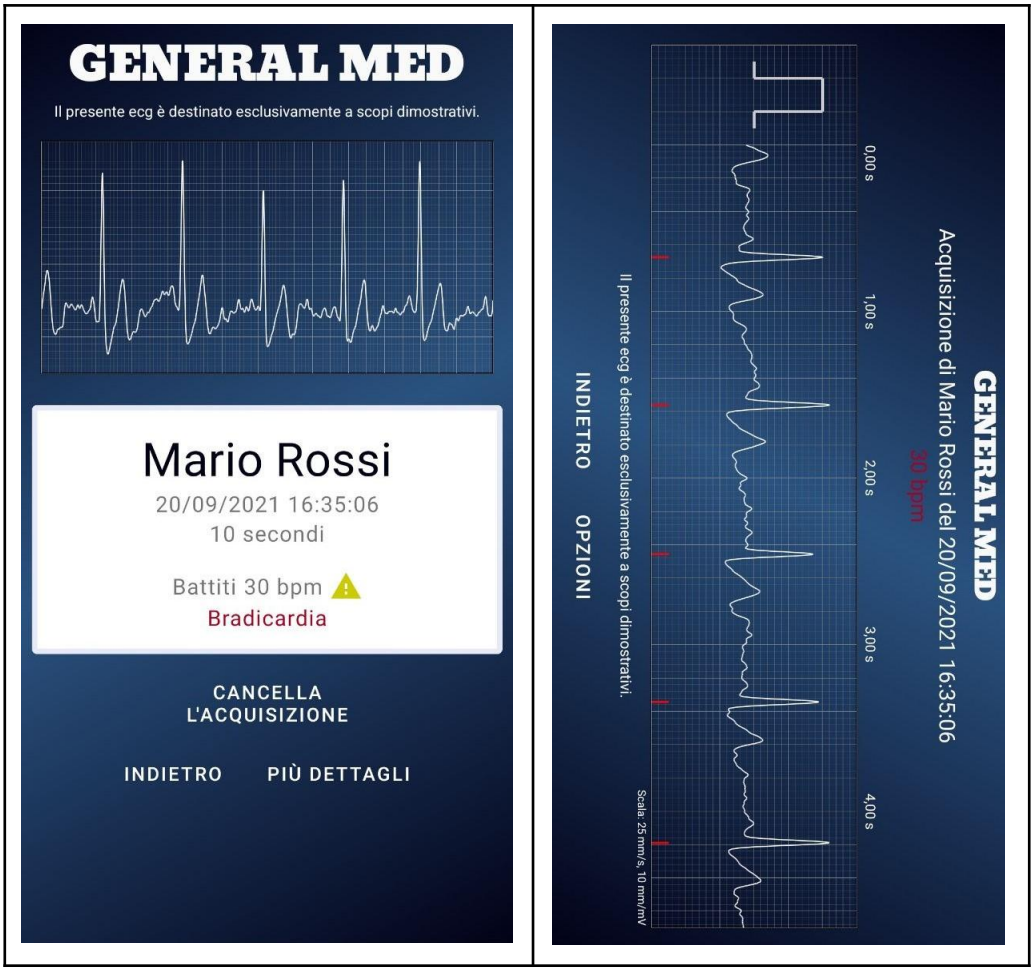


Fig. 2.8: Schermate con bradicardia grave

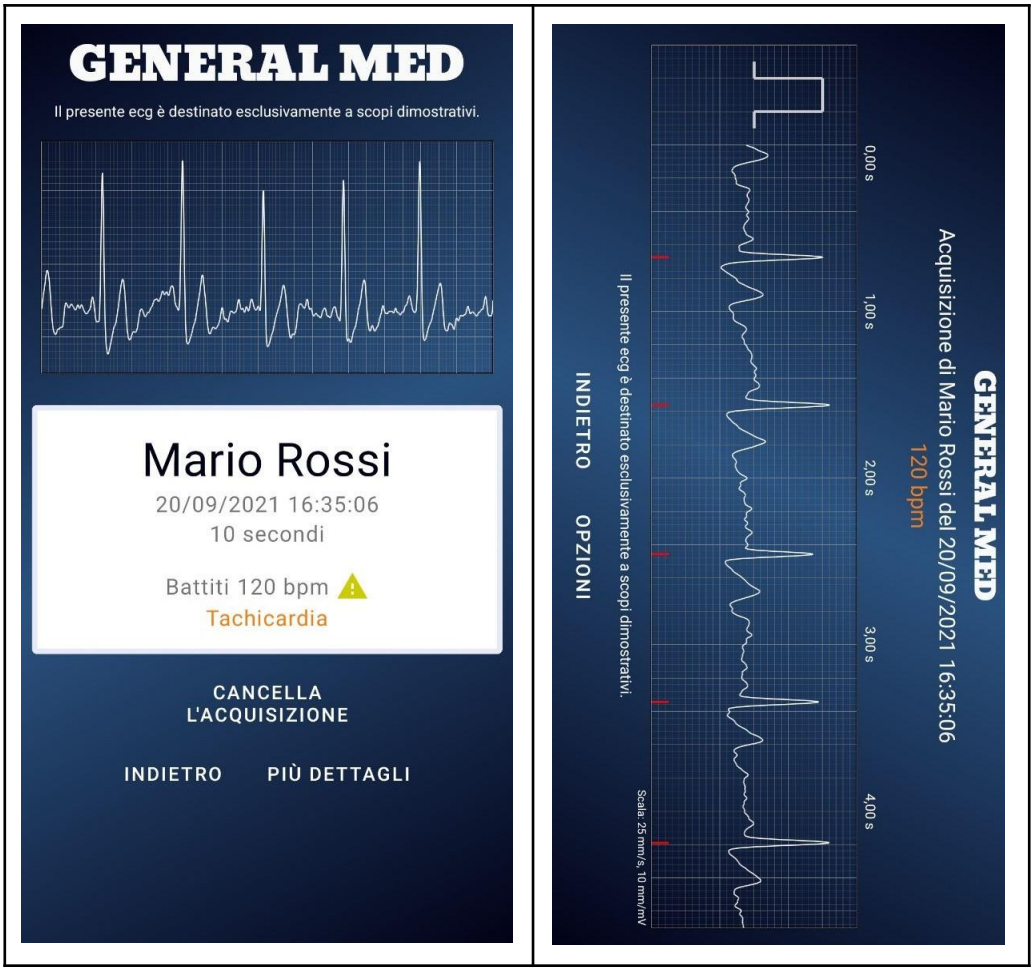


Fig. 2.9: Schermate con tachicardia lieve-moderata

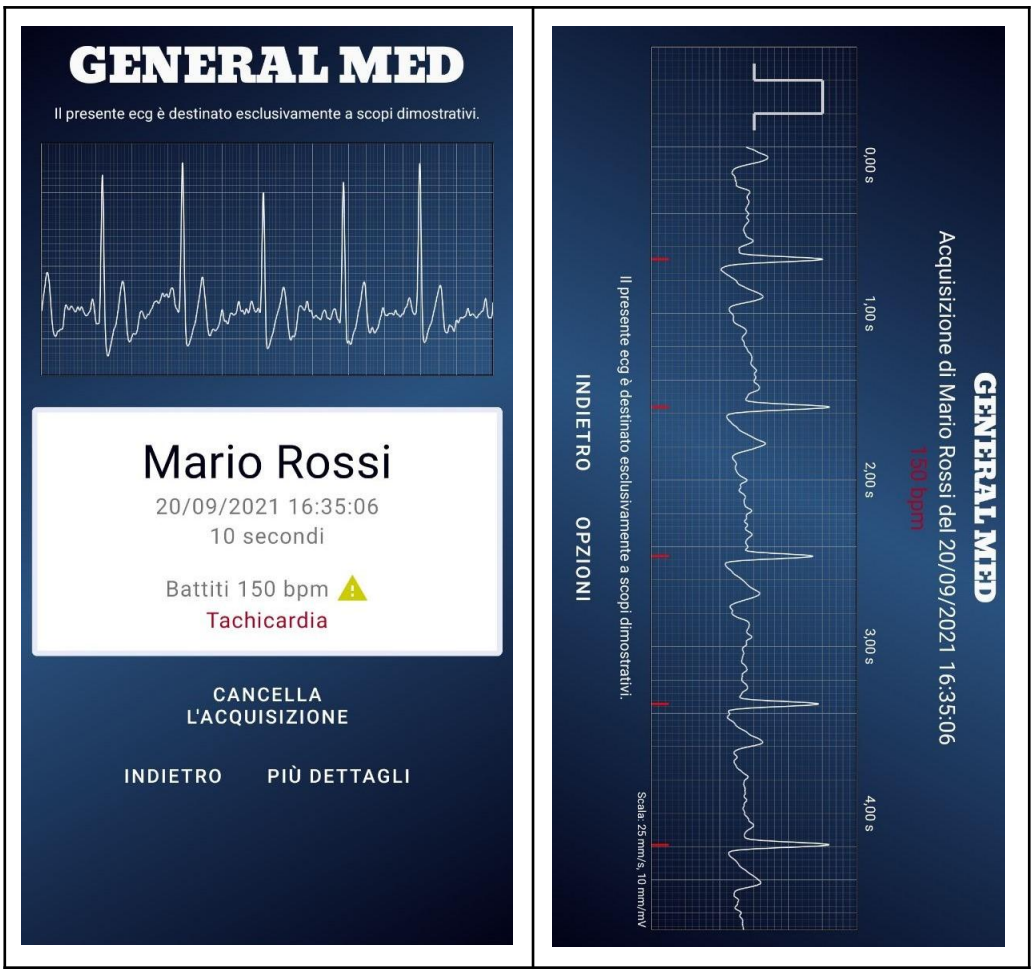


Fig. 2.10: Schermate con tachicardia grave

2.5 Refactoring codice app EcgMed

E’ stato effettuato un refactoring del codice dell’applicazione in modo da renderlo più leggibile, mantenibile e riutilizzabile.
Si tratta di una tecnica usata per modificare la struttura interna di parti di codice senza alterarne il comportamento esterno. (“Refactoring”)
Il refactoring è stato motivato dalla rilevazione di numerosi code smell, quali, per esempio, metodi eccessivamente lunghi e complessi, contenenti molto codice duplicato o eccessivamente interlacciato con altri oggetti il cui compito, in linea teorica, sarebbe dovuto differire.
L'azione di refactoring ha mirato ad eliminare alcuni dei sopracitati problemi attraverso l’introduzione di nuove utilities o della separazione logica degli “attori” all’interno del codice, ovvero il compito assegnato

alle diverse tipologie di classi. Le modifiche effettuate hanno voluto ricalcare un requisito di semplicità al fine di ridurre il rischio di introdurre errori.
Più nello specifico, alcuni esempi:

- E' stata separata la logica operativa implementata nelle `activities` dalla parte di visualizzazione implementata nei `fragments`, attraverso l'introduzione di `view handlers`, i quali permettono di raggruppare ed astrarre l'intervento su un fragment. Questo ha permesso di passare dal codice presente nella Fig. 2.11 a quello presente nella Fig. 2.12, attraverso l'handler della Fig. 2.13 il cui compito è quello di astrarre la logica di visualizzazione presente nel fragment della Fig. 2.14;

- E' stata introdotta una classe `FragmentedActivity`, dedicata alle activities composte da più fragments, che consente di aggiornare le informazioni visive su un determinato fragment, qualora questo sia visibile, mediante la famiglia di metodi `doOnFragmentIfAvailable`;

- E' stata introdotta un'interfaccia `FragmentNavigator`, contenente la logica di navigazione tra fragments in modo da poter essere implementata in maniera unificata da qualsiasi componente voglia effettuare la navigazione tra diversi fragments;

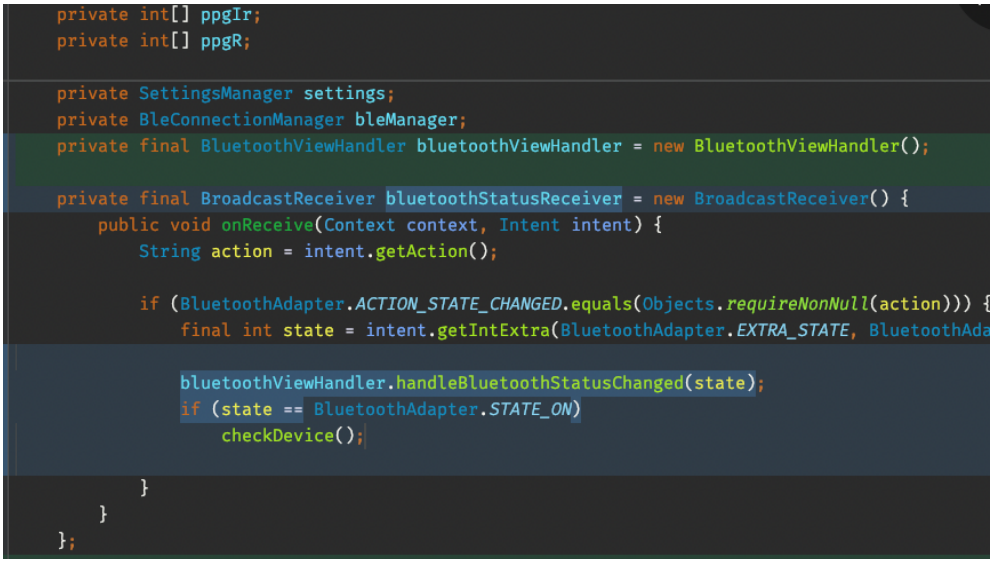
- Sono stati introdotti i concetti di `EnhancedFragment` / `EnhancedActivity` e relativi derivati, contenenti un insieme di logiche e utilities utilizzabili dai fragments e dalle activities.

```
private ImageView ivBluetooth;
private MaterialTextView tvBluetoothDevice;
private LinearLayout llBattery;

private SettingsManager settings;
private BleConnectionManager bleManager;
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();

        if (BluetoothAdapter.ACTION_STATE_CHANGED.equals(Objects.requireNonNull(action))) {
            final int state = intent.getIntExtra(BluetoothAdapter.EXTRA_STATE, BluetoothAda
            switch (state) {
                case BluetoothAdapter.STATE_OFF:
                    Snackbar.make(findViewById(android.R.id.content), getString(R.string.b
                    ivBluetooth.setImageResource(R.drawable.ic_bluetooth_inactive);
                    tvBluetoothDevice.setText(getString(R.string.device_connection_off_tex
                    break;
                case BluetoothAdapter.STATE_TURNING_OFF:
                    Snackbar.make(findViewById(android.R.id.content), getString(R.string.b
                    ivBluetooth.setImageResource(R.drawable.ic_bluetooth_inactive);
                    break;
                case BluetoothAdapter.STATE_ON:
                    Snackbar.make(findViewById(android.R.id.content), getString(R.string.b
                    ivBluetooth.setImageResource(R.drawable.ic_bluetooth_active);
                    tvBluetoothDevice.setText(getString(R.string.device_connection_on_text
                    checkDevice();
                    break;
                case BluetoothAdapter.STATE_TURNING_ON:
                    Snackbar.make(findViewById(android.R.id.content), getString(R.string.b
                    ivBluetooth.setImageResource(R.drawable.ic_bluetooth_inactive);
                    break;
            }
        }
    }
};
```

Fig. 2.11: Esempio di struttura del codice prima del refactor. E' possibile osservare come la classe di activity contenente tale codice si occupasse sia della parte di logica operativa che di quella di visualizzazione



```
private int[] ppgIr;
private int[] ppgR;

private SettingsManager settings;
private BleConnectionManager bleManager;
private final BluetoothViewHandler bluetoothViewHandler = new BluetoothViewHandler();

private final BroadcastReceiver bluetoothStatusReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();

        if (BluetoothAdapter.ACTION_STATE_CHANGED.equals(Objects.requireNonNull(action))) {
            final int state = intent.getIntExtra(BluetoothAdapter.EXTRA_STATE, BluetoothAda

            bluetoothViewHandler.handleBluetoothStatusChanged(state);
            if (state == BluetoothAdapter.STATE_ON)
                checkDevice();

        }
    }
};
```

Fig. 2.12: Codice presente nella Fig. 2.11 successivamente al refactor. Tutta la parte di visualizzazione viene astratta dal view handler

```
private class BluetoothViewHandler {
    private void handleBluetoothStatusChanged(int status) {
        doOnFragmentIfAvailable(GoToDevicesFragment.class, (f) -> {
            f.updateBluetoothStatus(status);
            return Unit.INSTANCE;
        });
    }

    private void handleBluetoothDisconnected() {
        doOnFragmentIfAvailable(GoToDevicesFragment.class, (f) -> {
            f.bluetoothDisconnected();
            return Unit.INSTANCE;
        });
    }

    private void handleBluetoothConnected(String deviceName) {
        doOnFragmentIfAvailable(GoToDevicesFragment.class, (f) -> {
            f.bluetoothConnected(deviceName);
            return Unit.INSTANCE;
        });
    }

    private void handleBleNotSupported() {
        doOnFragmentIfAvailable(GoToDevicesFragment.class, (f) -> {
            f.bleNotSupported();
            return Unit.INSTANCE;
        });
    }

    private void handleBatteryUpdate(int value) {
        doOnFragmentIfAvailable(GoToDevicesFragment.class, (f) -> {
            f.updateDeviceBattery(value);
            return Unit.INSTANCE;
        });
```

Fig. 2.13: Il view handler ha il compito di astrarre l'intervento sul fragment di interesse

```
fun updateBluetoothStatus(state: Int) {
    when (state) {
        BluetoothAdapter.STATE_OFF -> {
            showSnackbar(Snackbar.LENGTH_SHORT, "Bluetooth off")
            onBluetoothOff()
        }
        BluetoothAdapter.STATE_TURNING_OFF -> {
            showSnackbar(Snackbar.LENGTH_LONG, "Spegnimento bluetooth…")
        }
        BluetoothAdapter.STATE_ON -> {
            showSnackbar(Snackbar.LENGTH_SHORT, "Bluetooth on")
            onBluetoothOn()
        }
        BluetoothAdapter.STATE_TURNING_ON -> {
            showSnackbar(Snackbar.LENGTH_LONG, "Attivazione bluetooth…")
        }
    }
}


private fun onBluetoothOff() {...}

private fun onBluetoothOn() {...}

override fun onYesButton() {
    beforeBleDeviceFragment()
    navigateTo(BleDevicesFragment.newInstance())
}

private fun beforeBleDeviceFragment() {
    if (LocationClass.allowLocation(requireContext(), isGetLocation: false)) {
        if (!BleStatic.isBluetoothAvailable()) {
            val enableBtIntent = Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE)
            startActivityForResult(enableBtIntent, Constants.BLUETOOTH_PERMISSION_REQUEST_CODE)
            return
```

Fig. 2.14: Il fragment di interesse ha il compito di occuparsi dell'effettiva visualizzazione richiesta

Capitolo 3

3. User Experience di un’applicazione Mobile

3.1 Concetti riguardo *UX* e *UI*

Le persone trascorrono in media 4 ore al giorno (Chadwick) utilizzando le app sui propri smartphone. Gli utenti di app mobile cercano interazioni semplificate, si aspettano risultati immediati e sono orientati agli obiettivi. Questo comporta che la UX (User eXperience o, in italiano, esperienza utente) assume un’importanza fondamentale all’interno di un’applicazione.
L'esperienza utente, più nello specifico, rappresenta l'insieme di interazioni che gli utenti hanno con un’applicazione mobile.
La UX si differenzia dalla UI (User Interface, o, in italiano, interfaccia utente) in quanto quest’ultima riguarda l’insieme di elementi con cui un utente interagisce e la loro visualizzazione grafica.
L’UX è un aspetto essenziale di qualsiasi strategia di prodotto. Quest’ultima, però, necessita di particolari considerazioni per le applicazioni mobile; quali, ad esempio: vincoli di spazio, la tipologia di utente che è più probabile utilizzi l’applicazione e il contesto in cui quest’ultimo possa interagirci, lo stato d’animo in cui l’utente possa trovarsi nel momento di interazione, e via discorrendo.
Stabilire un linguaggio di navigazione chiaro e completo è essenziale per la progettazione di app mobili e per guidare facilmente gli utenti nel suo utilizzo.
La maggior parte delle interfacce mobili fanno molto affidamento su modelli visivi universali e riconoscibili per segnalare azioni su attività e funzionalità comuni (ad esempio un'icona a tre righe per indicare un menu compresso), tuttavia questi modelli possono non essere facilmente intuibili ad utenti altamente inesperti e poco familiari con l’utilizzo di smartphones.

La navigazione all’interno dell’applicazione rappresenta la totalità della sua usabilità e comprende il layout, il design, i contenuti e altri elementi che consentono agli utenti di raggiungere i propri obiettivi all'interno dell'app.
E’ essenziale che la navigazione e la visualizzazione rimangano coerenti all’interno di tutta l’applicazione con i gesti che l’utente può/deve compiere, al fine di ottimizzarne l'usabilità.
La personalizzazione della user interface, inoltre, aiuta a fornire all'utente un'esperienza più unica e pertinente, includendo pochi elementi e pochi messaggi, potenzialmente di effetto, sullo schermo. E’ necessario visualizzare solo contenuti rilevanti ad ogni singola schermata, evitando di sovraccaricarla di informazioni, cosa che potenzialmente potrebbe portare confusione nell’utilizzatore.
Sebbene l'UX non sia immediatamente visibile all'utente, il design dell'interfaccia, quindi la UI, è la prima cosa che vedranno, influenzando immediatamente la loro percezione dell'app. Il design di una buona UI aiuta a guidare gli utenti attraverso l'interfaccia utilizzando ausili visivi.
Gli elementi principali di una buona interfaccia utente possono essere riassunti, secondo ClearBridge, con i seguenti punti:

- Chiarezza: l'interfaccia deve evitare confusione o ambiguità rendendo tutto chiaro attraverso il linguaggio e le immagini. Le app con una buona interfaccia utente non richiedono un'ampia introduzione per mostrare all'utente come questa funziona;

- Familiarità: a molti utenti piace vedere funzioni familiari, pulsanti o inviti all'azione. La familiarità include simboli, icone o colori comuni per trasmettere un messaggio;

- Reattività: un'eccellente interfaccia utente non dovrebbe avere ritardi durante il suo utilizzo o risultare lenta. Fornire feedback durante il caricamento di una schermata migliorerà anche l'interfaccia utente tenendo l'utente informato su ciò che sta accadendo, ad esempio visualizzando maggiori dettagli sulle operazioni in corso di svolgimento;

- Coerenza: mantenere l'interfaccia coerente nell'applicazione è importante perché consente agli utenti di riconoscere i modelli di utilizzo. Una volta che gli utenti hanno appreso come funzionano

determinate parti dell'interfaccia, possono applicare questa conoscenza a nuove aree e funzionalità;

- Estetica: sebbene non sia necessario che l'interfaccia sia attraente per il suo corretto funzionamento, farlo renderà l'app più appetibile. Tale obiettivo può essere raggiunto mediante un'accurata scelta delle immagini da visualizzare e della loro posizione, nonché un'accurata scelta dei colori.

3.2 Il redesign effettuato

Il lavoro di redesign effettuato ha voluto tenere conto dei principali elementi chiave che possono cambiare la UX di un utente, portando alla personalizzazione delle applicazioni in base ad una serie di domande che ci si è posti.

3.2.1 Qual è il target dell'applicazione?

L'utente medio che ci si aspetta utilizzi quest'applicazione è una persona di età avanzata.
E' probabile che quest'ultima sia altamente inesperto nell'utilizzo degli smartphones e abbia poca familiarità nel riconoscere i modelli visivi comunemente utilizzati all'interno delle applicazioni; ovvero mancata capacità di associare le icone visualizzate alla loro funzione, nonché poca, se non assente, familiarità con le gesture più comuni.
Queste considerazioni hanno portato ad abbandonare l'utilizzo di icone o gesture, optando per pulsanti auto-descrittivi della loro funzione, come possibile osservare nelle Figg. 3.1, 3.2 e 3.3.

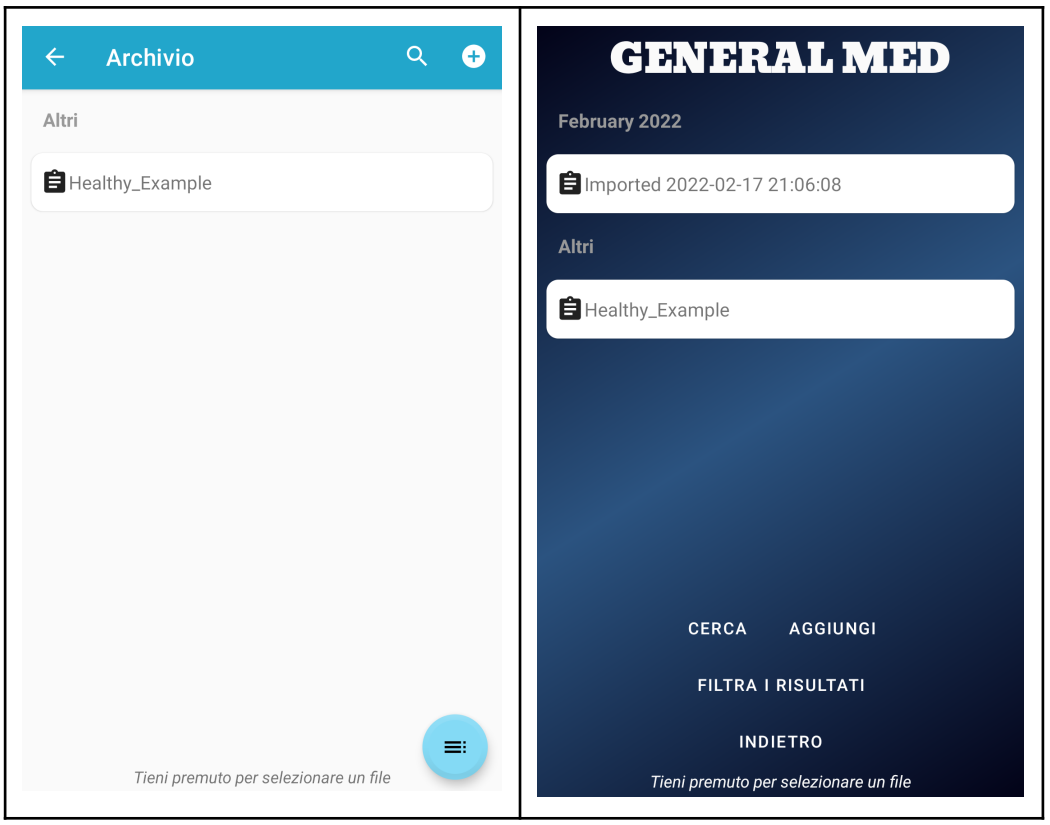


Fig. 3.1:
A sinistra la versione prima del redesign, a destra quella successiva. I pulsanti per la navigazione alla pagina precedente, nonché quelli per il filtraggio dei risultati, l'aggiunta di nuove acquisizioni oppure la ricerca tra i risultati sono stati sostituiti con pulsanti auto descrittivi, a discapito dell'utilizzo di icone comuni

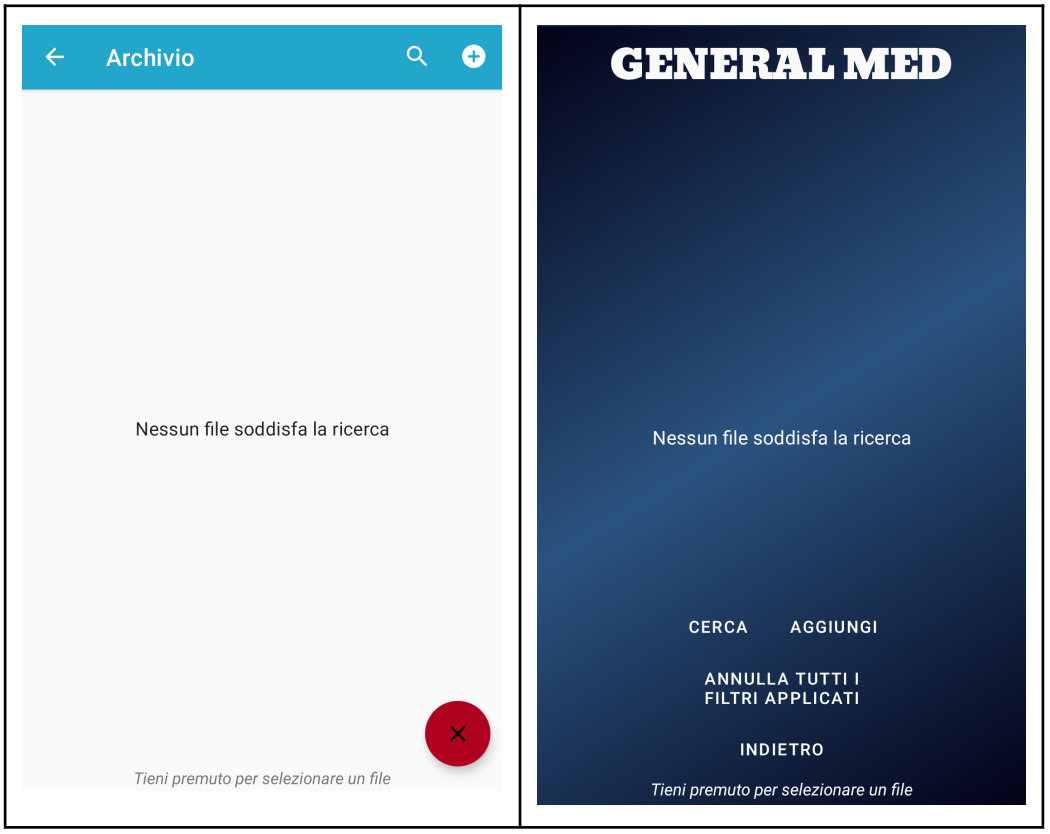


Fig. 3.2:
A sinistra la versione prima del redesign, a destra quella successiva. I pulsanti per la navigazione alla pagina precedente, nonché quelli per annullare i filtri applicati oppure l'aggiunta di nuove acquisizioni sono stati sostituiti con pulsanti auto descrittivi

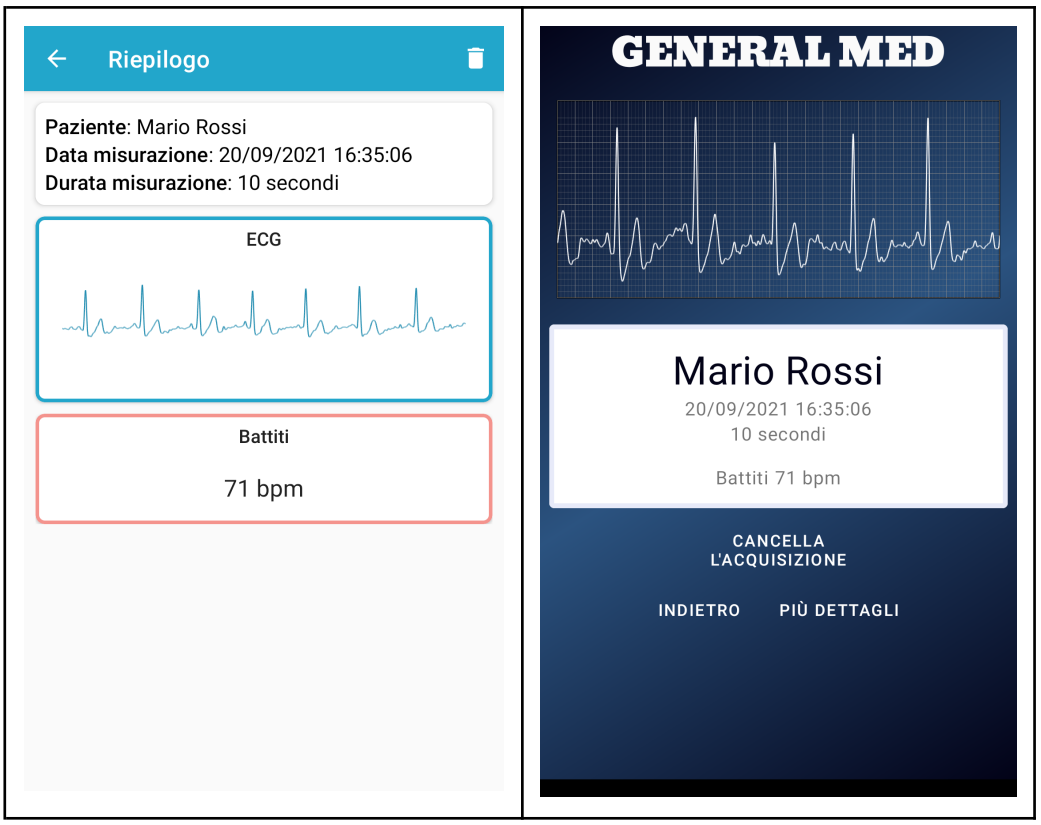


Fig. 3.3:
A sinistra la versione prima del redesign, a destra quella successiva. I pulsanti per la navigazione alla pagina precedente, nonchè quelli per l'eliminazione dell'acquisizione oppure la navigazione alla schermata di maggiori dettagli dell'acquisizione sono stati sostituiti con pulsanti auto descrittivi

3.2.2 Qual è il contesto di interazione?

Il contesto di interazione più comune con le applicazioni ci si aspetta sia in un momento in cui l'utente non abbia particolare fretta, dal momento che deve attendere il completamento di un'acqusizione la cui lunghezza può variare dai 10 ai 30 secondi.
Questa considerazione, unita alla necessità di guidare l'utente passo passo durante l'utilizzo dell'applicazione, data la sua potenziale scarsa familiarità con le applicazioni mobile, hanno portato ad abbandonare l'utilizzo di un menù ricco di icone, come rappresentato nella parte sinistra della Fig 3.4, a favore di un menù in cui ogni pulsante è

descrittivo dell'esito dell'interazione, portandolo passo dopo passo all'azione che intende portare a termine, come visibile nella parte destra della Fig. 3.4.

Fig. 3.4: A sinistra il menù prima del redesign,
a destra quello successivo.

3.2.3 Qual è lo stato d’animo durante l’interazione?

Prima della misurazione o durante lo svolgimento dell’esame, l’utente potrebbe potenzialmente trovarsi in uno stato ansioso, dovuto alla paura riguardo l’esito dell’acquisizione.
Tale valutazione ha guidato la scelta di immagini e formulazione di messaggi che tranquillizzassero l’utente, come è possibile visualizzare nella Fig. 3.5, nonché la scelta dei colori principali utilizzati all’interno dell’applicazione:

- Blu, richiama alla mente sentimenti di calma o serenità. È spesso descritto come pacifico, tranquillo, sicuro e ordinato;
  (“The Color Psychology of Blue”)

- Nero, che nel feng shui è associato all'elemento acqua ed evoca potere, mistero e calma. Per alcuni, il nero evoca associazioni positive, tra cui eleganza, oltre ad essere un colore che non stanca l’occhio, soprattutto se utilizzato su dispositivi mobili e di piccole dimensioni.
  (“The Color Psychology of Black”)

Fig. 3.5: Schermata di benvenuto dell’applicazione

Capitolo 4

4. Visualizzazione live dell'ecg

4.1 Obiettivi

Al fine di migliorare ulteriormente l'esperienza dell'utente all'interno delle applicazioni, in special modo nella fase di acquisizione dell'ecg, che poteva risultare come carente di feedback se composta da un solo timer o un solo messaggio scritto, si è deciso di implementare la visualizzare dell'ecg in corso di acquisizione.
Tale implementazione, tuttavia, ha richiesto particolari accorgimenti affinchè la qualità del grafico visualizzato fosse soddisfacente e la visualizzazione raggiungesse un update rate di 25 frame al secondo, valore minimo affinché l'occhio umano abbia la stessa sensazione di movimento che ha guardando la realtà, senza portare alla necessità di modifiche lato firmware o hardware.

4.2 Struttura generica

La struttura della soluzione adottata si articola in diverse fasi: i dati vengono ricevuti, raggruppati in batch di dimensione prefissata, filtrati, bufferizzati e visualizzati con una finestra di update di 40 ms.
Quando all'utente viene richiesto se desidera avviare un'acquisizione, l'applicazione si trova nel fragment `GoToAcquireFragment`.
Se l'utente decide di avviare l'acquisizione, come riassunto nella Fig. 4.1, si avvia prima di tutto la navigazione verso il fragment `ActionAcquireFragment`, il quale chiama il metodo statico `EcgLineCharter.liveChartUpdater` (descritto nel dettaglio all'interno del paragrafo 4.4) passando come parametro il chart su cui deve avvenire la visualizzazione live dell'ecg in corso di acquisizione.

Tale metodo ritorna un oggetto `LiveChartUpdater` che viene salvato nelle variabili del fragment.

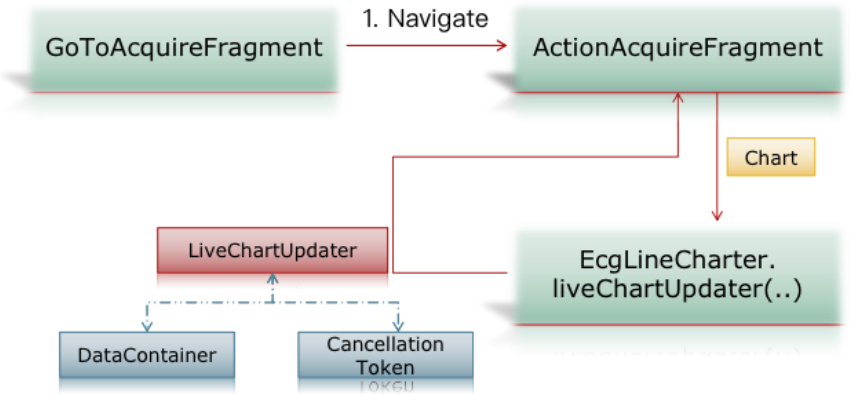


Fig. 4.1: Navigazione verso `ActionAcquireFragment` e inizializzazione del `LiveChartUpdater`

Successivamente, come rappresentato nella Fig 4.2, si avvia l'acquisizione: i dati vengono ricevuti con una frequenza di 500 Hz dal `BleService` il quale li raggruppa in batch di dimensione prefissata e li invia alla `MainActivity`. Quest'ultima li processa come descritto al paragrafo 4.3 e li invia al fragment `ActionAcquireFragment`.

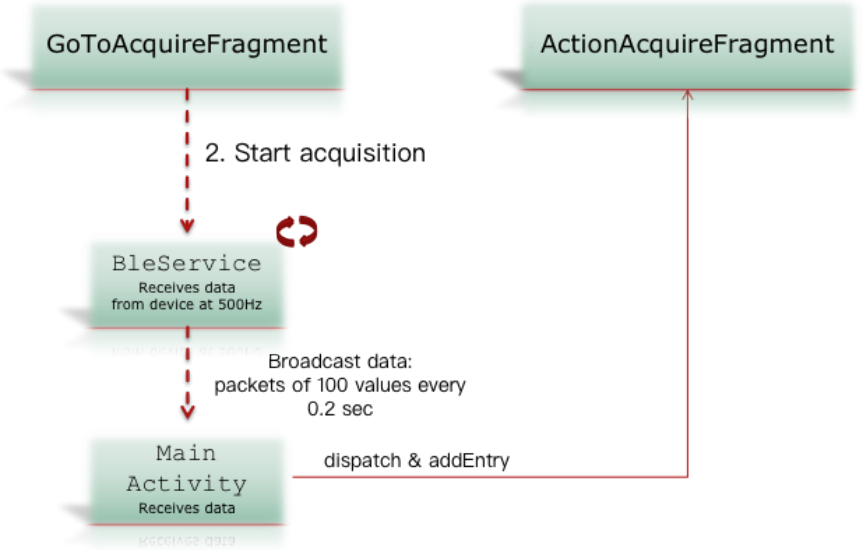


Fig. 4.2: Si avvia l'acquisizione e i dati vengono recapitati all'`ActionAcquireFragment`

Come rappresentato nello schema della Fig. 4.3, l'`ActionAcquireFragment`, quando riceve i dati, li inserisce in maniera thread safe (dettagli nel paragrafo 4.4) all'interno del data container contenuto nell'oggetto `LiveChartUpdater`. Quando il timer dell'acquisizione scade, l'`ActionAcquireFragment` chiama il metodo `complete()` sul cancellation token contenuto all'interno dell'oggetto `LiveChartUpdater`, il quale termina il thread avviato dall' `EcgLineCharter.liveChartUpdater`.

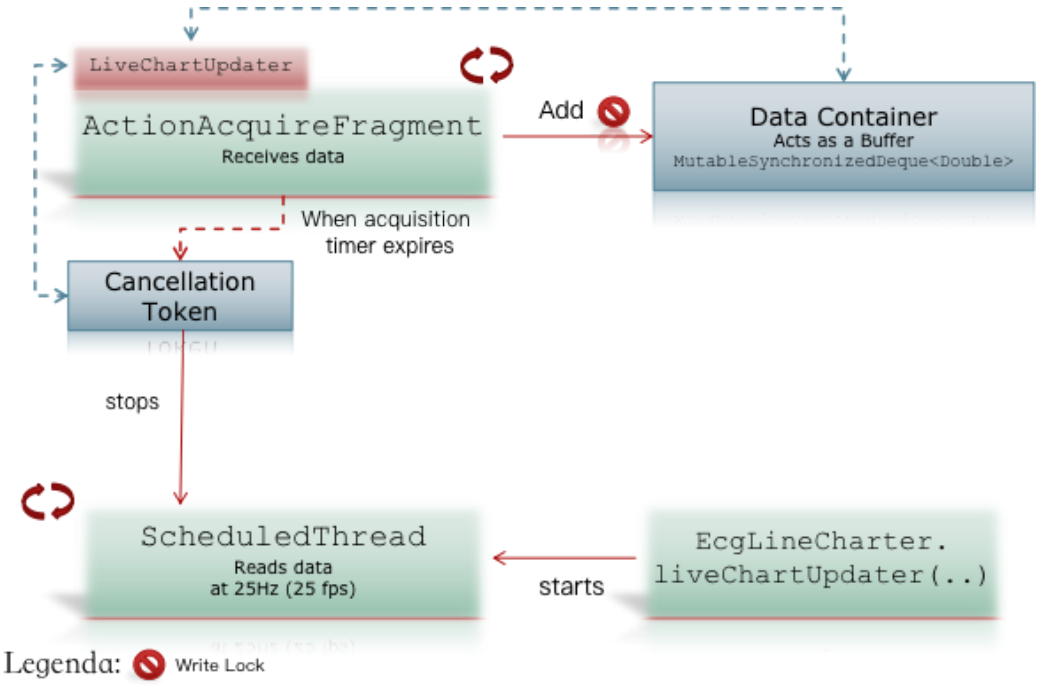


Fig. 4.3: I dati ricevuti vengono inoltrati nel data container contenuto nel `LiveChartUpdater`. Conclusa l'acquisizione, il cancellation token ferma il thread creato.

Contemporaneamente, il thread sopra citato controlla che il data container del `LiveChartUpdater` contenga nuovi dati. In caso affermativo, ne rimuove 20 e li inserisce nel data container del grafico, il quale si occupa di aggiornare la visualizzazione dell'ecg in corso di acquisizione (dettagli nel paragrafo 4.4). Quanto appena descritto può essere rappresentato nello schema della Fig. 4.4.

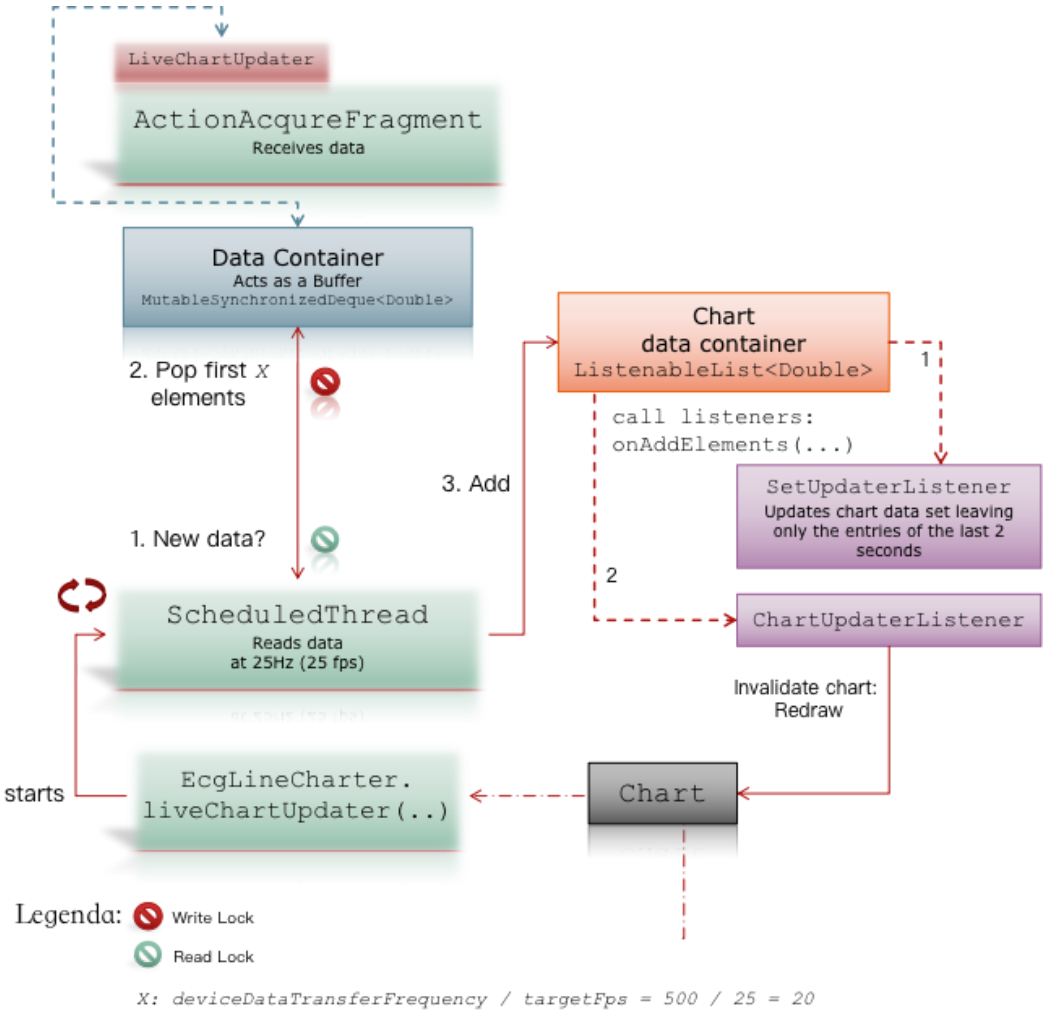


Fig. 4.4: Il thread esterno si occupa di aggiornare i due data container e scatenare l'aggiornamento del grafico

4.3 Filtraggio dei dati in live

Durante l'acquisizione, i campioni vengono memorizzati all'interno di un vettore dal `BleService`. Ogni 100 campioni acquisiti, i quali vengono ricevuti in una finestra di 200 ms, viene creato un pacchetto che viene inoltrato alla `MainActivity`, la quale, ogni volta, controlla il valore di offset in modo da scartare i primi 1500 campioni ricevuti e gli ultimi 500, dal momento che spesso sono soggetti ad un rumore troppo elevato.

I restanti campioni vengono inoltrati alla classe `ActionAcquireFragment` che si occupa di centrare i valori ricevuti sullo zero mediante una traslazione verso il basso, specchiarli intorno all'asse delle ascisse a seconda del polso su cui si trova il dispositivo e filtrarli attraverso una funzione di *notchFilter* e una serie di medie mobili con diverse finestre.
Per maggiori dettagli riguardo a tali tecniche si rimanda alla tesi di Giacomo Gorga.
I dati così filtrati vengono aggiunti ad un data container che si occupa della loro bufferizzazione e visualizzazione.

4.4 Bufferizzazione e visualizzazione

La struttura dell'API atta alla visualizzazione è stata studiata affinchè tale lavoro sia del tutto trasparente al chiamante.
Essa si articola di un metodo statico della classe `EcgLineCharter`

```
fun liveChartUpdater(lineChart: LineChart):
LiveChartUpdater
```

che riceve in input il chart su cui visualizzare l'ECG live.
Internamente, il lavoro svolto è strutturato in diverse fasi.
Dopo aver effettuato l'inizializzazione del grafico e del relativo dataset, viene istanziato il data container del grafico (una collection la cui implementazione si basa sul pattern *Delegate* e sul pattern *Observer*, rispettivamente descritti all'interno del paragrafo 4.4.1 e del paragrafo 4.4.2) e vengono registrati su quest'ultimo due observers: uno il cui compito è quello di allineare il dataset del grafico, l'altro il cui scopo è quello di aggiornare adeguatamente il grafico stesso a partire dai nuovi dati disponibili.
Questo data container viene nascosto al chiamante dell'API, a cui viene ritornato un oggetto `LiveChartUpdater` che internamente contiene due campi:

1. `dataContainer`: una `Collection<Double>`, all'interno della quale il chiamante deve inserire i dati, implementata come `MutableSynchronizedDeque<Double>`, estensione custom di `Deque<Double>` che presenta thread safeness basata su reentrant read-write lock;
   (dettagli nel paragrafo 4.4.3)
2. `cancellationToken`: oggetto che espone al chiamante un metodo per terminare la visualizzazione live, basato sul pattern *cancellation token*, descritto nel paragrafo 4.4.4.

L'allineamento tra il `dataContainer` esposto al chiamante dell'API e quello che fa avviare l'aggiornamento del grafico tramite i suoi observers viene portato a termine da un thread istanziato all'interno dell'API stessa, seguendo il pattern *producer-consumer* (illustrato al paragrafo 4.4.5). Quest'ultimo, con un periodo di 40 ms, provvede a leggere e rimuovere, se presenti, 20 elementi dalla prima struttura dati, riportandoli nella seconda.
I motivi di tali valori sono i seguenti:

- 40ms: valore ottenuto dalla necessità di effettuare 25 aggiornamenti del grafico al secondo, al fine di ottenere un framerate di 25 fps;
- 20 elementi: numero di elementi necessari a visualizzare, con un update rate di 40ms, tutti i 500 dati ricevuti dal dispositivo all'interno di un secondo.

In questo modo, si è riusciti a disaccoppiare i due rate di aggiornamento dei dati:

- 500 Hz la prima, frequenza di invio dei dati da parte dell'hardware;
- 25 Hz la seconda, frequenza di visualizzazione.

4.4.1 Note sul pattern *Delegate*

Il pattern *Delegate* consente di ottenere la stessa riutilizzabilità del codice ottenibile con l'inheritance, attraverso la composizione di oggetti.
In questo pattern, un oggetto gestisce una richiesta delegandola a un secondo oggetto (il delegato). Il delegato è un oggetto di supporto, contenente spesso una logica che si vuole estendere.
Questo pattern è stato largamente adottato all'interno dei progetti, in particolare nella creazione di data structures che svolgessero compiti particolari affinché non ci fosse la necessità di dover gestire direttamente le operazioni di aggiunta/rimozione/modifica, ma delegandole ad una struttura dati già esistente.
Le operazioni sopra descritte, dunque, vengono semplicemente inoltrate alla struttura dati sottostante, effettuando eventuali altre operazioni di contorno necessarie.

Esempio:

```
override fun add(element: T): Boolean =

    delegate.add(element)
     .also { result ->
         ...doOtherStuff...
    }
```

4.4.2 Note sul pattern *Observer*

Il pattern *Observer*, noto anche come *Publish-Subscribe*, permette di definire un subject i cui cambiamenti vengono osservati da molteplici observers, introducendo una dipendenza *1:n* (uno a molti) fra oggetti e mantenendo un alto livello di consistenza fra classi correlate, senza produrre situazioni di accoppiamento elevato.
All'interno di questo pattern, gli attori principali sono dunque di due tipi:

1. Subject, la classe Observable, che ha conoscenza dei propri Observer, fornisce operazioni per la loro aggiunta e la loro rimozione nonché fornisce operazioni per la loro notifica;

2. Observer, che specifica un'interfaccia per la notifica di eventi agli oggetti interessati in un Subject.

Nelle Figg. 4.5 e 4.6 è possibile osservare la struttura di tale pattern.

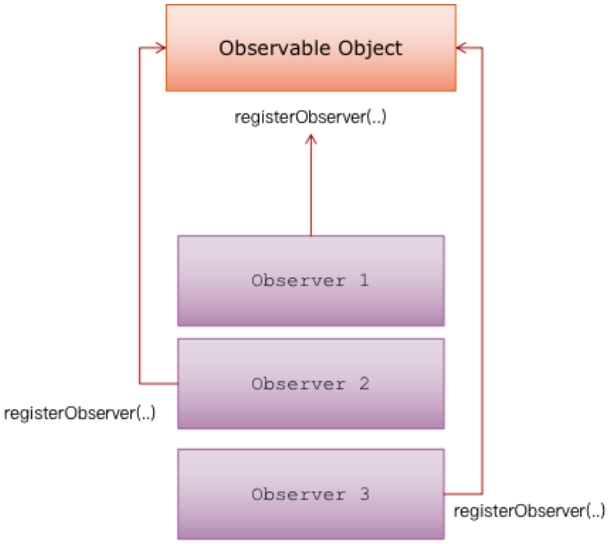


Fig. 4.5: Vengono registrati tutti gli *observers* presso l'oggetto *observable*

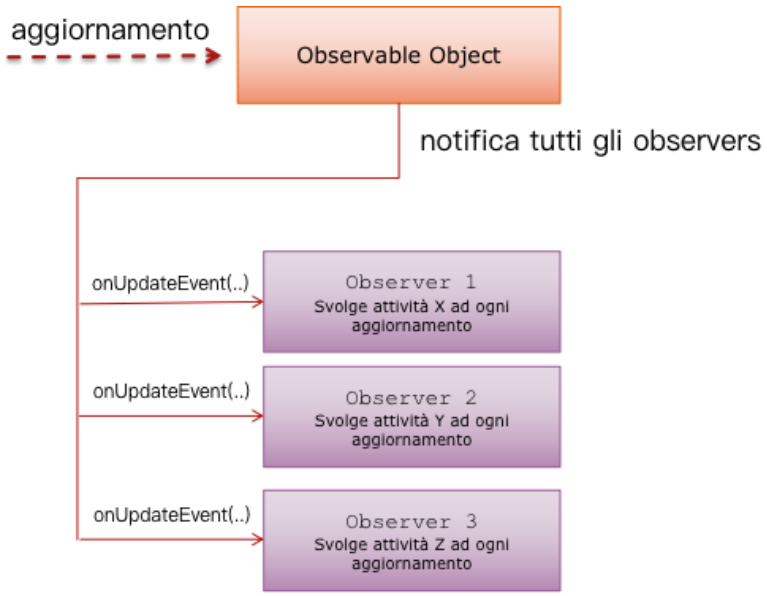


Fig. 4.6: Quando l'oggetto *observable* viene aggiornato, tutti gli *observers* vengono notificati

All'interno delle applicazioni, tale pattern è stato implementato mediante la creazione della classe `ObservableList<T>`. Quest'ultima implementa `MutableList<T>` di Kotlin, delegando tutte le implementazioni dei suoi metodi ad una lista interna ricevuta o creata in fase di istanziazione (tramite il pattern *delegate* descritto nel paragrafo 4.4.1), ed estende `Observable<ListObserver<T>>`, ereditando automaticamente la possibilità di registrare o rimuovere observers.
La classe `ListObserver<T>`, che è un'interfaccia che tutti gli observers di questo tipo di lista devono implementare, consta dei seguenti metodi:

- `fun onAddElements(elements: Collection<T>)`
  Viene chiamato dalla lista successivamente all'aggiunta di elementi nuovi all'interno di quest'ultima. Il parametro `elements` rappresenta gli elementi aggiunti;
- `fun onRemoveElements(elements: Collection<T>)`
  Viene chiamato dalla lista successivamente alla rimozione di elementi

da quest'ultima. Il parametro `elements` rappresenta gli elementi rimossi;

- `fun onElementChanged(previous: T, current: T)`
  Viene chiamato dalla lista successivamente alla sostituzione di un elemento all'interno quest'ultima. Il parametro `previous` rappresenta l'elemento sostituito, il parametro `current` rappresenta l'elemento aggiunto;
- `fun onClearList()`
  Viene chiamato dalla lista successivamente allo svuotamento di quest'ultima.

Nel delegare le varie chiamate dei metodi ereditati dall'interfaccia `MutableList<T>` di Kotlin alla lista sottostante, dunque, `ObservableList<T>` chiama opportunamente i vari metodi di tutti gli observers registrati sulla data istanza.

Esempio:

```
override fun add(element: T): Boolean =

    innerList.add(element)
    .also { result ->
        if (result)
           this.mObservers
            .forEach { o ->
               o.onAddElements(listOf(element))
            }
    }
```

4.4.3 Note sulla thread safety

I problemi principali quando si parla di concorrenza e di sviluppo multi threading sono quelli inerenti alle cosiddette *sezioni critiche*, ovvero quelle parti di codice che non possono essere eseguite da più thread in parallelo, a causa di risorse condivise.
Nel nostro specifico caso, avendo dovuto utilizzare una collection che veniva letta e modificata da un thread e aggiornata da un altro, le sezioni critiche erano quelle relative all'aggiornamento della collection e alla sua lettura.
Il problema principale da affrontare, in queste situazioni, sono le cosiddette *race conditions*, le quali si verificano quando due o più thread possono accedere a dati condivisi e tentano di modificarli contemporaneamente. Poiché l'algoritmo di thread-scheduling può passare da un thread all'altro in qualsiasi momento, non si conosce l'ordine in cui questi tenteranno di accedere ai dati condivisi. Pertanto, il risultato della modifica dei dati, e quindi lo stato di consistenza della collection, dipende dall'algoritmo di thread-scheduling.
Tale problema può essere risolto con la *mutua esclusione*, ovvero una tecnica che permette ad un dato thread di avere accesso esclusivo ad una data sezione critica, senza che altri possano eseguirla nello stesso momento.
Tale tecnica può essere ottenuta in molteplici modi, tuttavia può portare ad altrettanti problemi quali, ad esempio:

- *Deadlocks*: si verificano quando un thread attende la risorsa B, avendo acquisito la risorsa A, e un altro attende la risorsa A, avendo acquisito la risorsa B. Ergo sono reciprocamente dipendenti e si trovano in una situazione di stallo;

- *Resource starvation*: si verifica quando ad un processo vengono perennemente negate le risorse necessarie per elaborare il suo compito. Potrebbe essere causato da errori di scheduling o errori nell'algoritmo di mutua esclusione.

Per tale motivo si tende ad evitare gestioni ed implementazioni custom di questa tecnica, avvalendosi sempre di utilities messe a disposizione dal linguaggio.
Nel nostro particolare caso, volendo implementare una soluzione elastica e che potesse essere riutilizzata in maniera efficiente anche in situazioni più complesse, ovvero con più attori che leggono/aggiornano la struttura dati, abbiamo deciso di avvalerci del *ReentrantReadWriteLock* messo a disposizione da Java.
Quest'ultimo è caratterizzato da due concetti molto importanti:

- L'essere reentrant: permette di essere acquisito più volte dallo stesso thread senza che quest'ultimo blocchi se stesso. È utile in situazioni in cui non è facile tenere traccia se si è già acquisito il lock o meno, nel nostro particolare caso permette un disaccoppiamento facile per l'iteratore della collection, pur avendo possibilità di utilizzare lo stesso lock;

- L'essere read-write: consente l'accesso simultaneo per le operazioni di sola lettura, mentre le operazioni di scrittura richiedono l'accesso esclusivo. Ciò significa che più thread possono leggere i dati in parallelo ma è necessario un blocco esclusivo per la scrittura o la modifica dei dati. Quando uno scrittore scrive i dati, tutti gli altri scrittori o lettori verranno bloccati fino a quando il primo non avrà terminato le sue operazioni.

Quanto descritto, dunque, ha portato alla nascita di diverse classi.
Prima tra tutte, `SynchronizedDataStructure<DS>`, dove `DS` è il tipo di data structure su cui ci si vuole appoggiare tramite il pattern *delegate* (descritto al paragrafo 4.4.1), è l'interfaccia contenente il contratto a cui le diverse implementazioni devono sottostare.
Più nel dettaglio, consta dei seguenti metodi:

- `fun removeSync(): DS`
  Permette di rimuovere l'involucro di gestione della sincronizzazione, accedendo in maniera diretta alla data structure sottostante, ovvero alla struttura dati delegata;

- `fun <Q> doReadLocking(action: DS.() -> Q): Q`
  Permette di eseguire l'azione ricevuta come parametro avendo acquisito un lock di tipo read;
- `fun <Q> doWriteLocking(action: DS.() -> Q): Q`
  Permette di eseguire l'azione ricevuta come parametro avendo acquisito un lock di tipo write.

`MutableSynchronizedCollection<DS, T>`, dove `DS` è il tipo di data structure su cui ci si vuole appoggiare (ovvero il tipo di struttura dati delegata) e `T` è il tipo generico di tale data structure , è una sua implementazione, il cui scopo è quello di aggiungere un layer d'astrazione intermedio, implementando tramite il pattern *delegate* l'interfaccia di Kotlin `MutableCollection<T>` e gestendo opportunamente i lock nelle fasi di lettura/scrittura.
Infine, `MutableSynchronizedDeque<T>`, estensione di `MutableSynchronizedCollection<Deque<T>, T>`, ed implementazione di `Deque<T>`, ricalca l'ideologia della prima aggiungendole i metodi messi a disposizione dalla seconda tramite un'opportuna gestione dei locks nelle fasi di lettura/scrittura.

4.4.4 Note sul pattern *Cancellation Token*

Il pattern del *Cancellation Token* viene ampiamente utilizzato all'interno della programmazione concorrente con lo scopo di permettere ad agenti esterni (thread terzi) l'interruzione di un task in corso di svolgimento su un dato thread.
Tale pattern si basa su un oggetto, appunto chiamato `CancellationToken`, il quale viene restituito o passato come parametro al momento del lancio di un'operazione asincrona.
Il chiamante di tale operazione, dunque, avrà la possibilità di annullarla chiamando un metodo sul token in questione. Sarà premura dell'operazione asincrona controllare periodicamente lo stato del token e, quando quest'ultimo è marcato come concluso, terminare le operazioni in corso di svolgimento.

Nella Fig. 4.7 è possibile visualizzare ad alto livello la struttura sopra descritta.

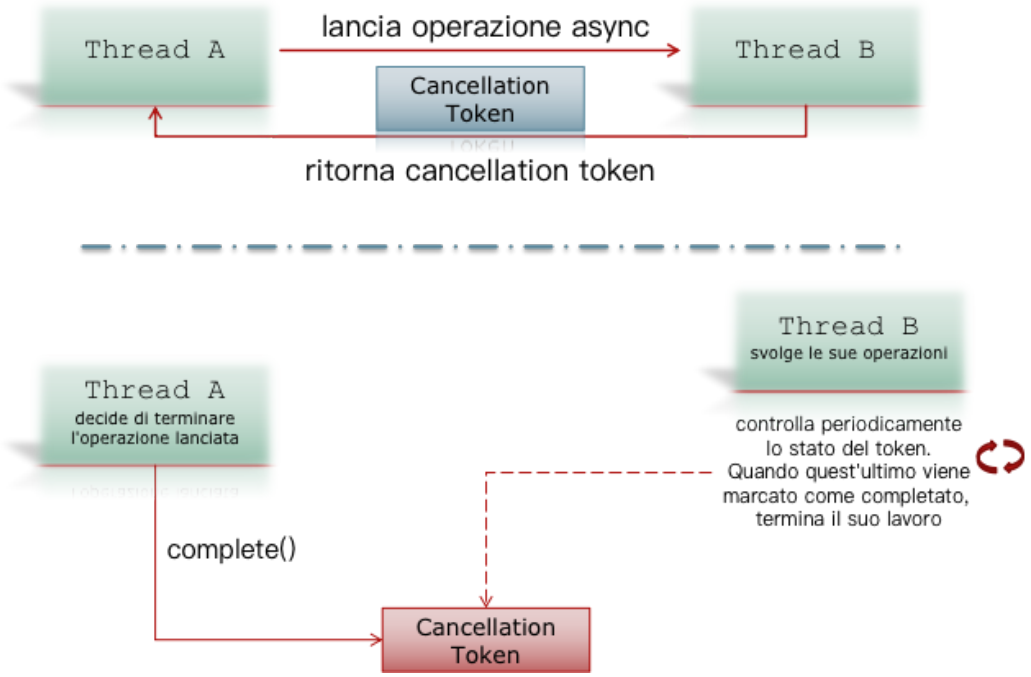


Fig. 4.7: Struttura del pattern *cancellation token*

4.4.5 Note sul pattern *Producer-Consumer*

Il modello *Producer-Consumer* permette di separare l'individuazione di un compito da svolgere dal suo svolgimento vero e proprio; più comunemente l'individuazione di dati da elaborare dalla loro elaborazione. Tale modello prevede due attori principali, come possibile visualizzare all'interno della Fig. 4.8: il *producer*, il cui compito è quello di generare/individuare i dati ed inserirli in una struttura dati comune, spesso una coda, ed il *consumer*, il cui compito è quello di leggere ed elaborare i dati presenti su quest'ultima.
Questo disaccoppiamento implica che i produttori non hanno necessità di conoscere come verrà elaborato ogni elemento, quanti consumatori lo elaboreranno o quanti altri produttori sono presenti. Il loro compito è quello di generare i dati con la loro frequenza. Allo stesso modo i consumatori non hanno bisogno di sapere da dove provengano i dati e quanti altri produttori e consumatori ci sono; il loro compito è prendere i dati dalla coda ed elaborarli, con la loro frequenza.

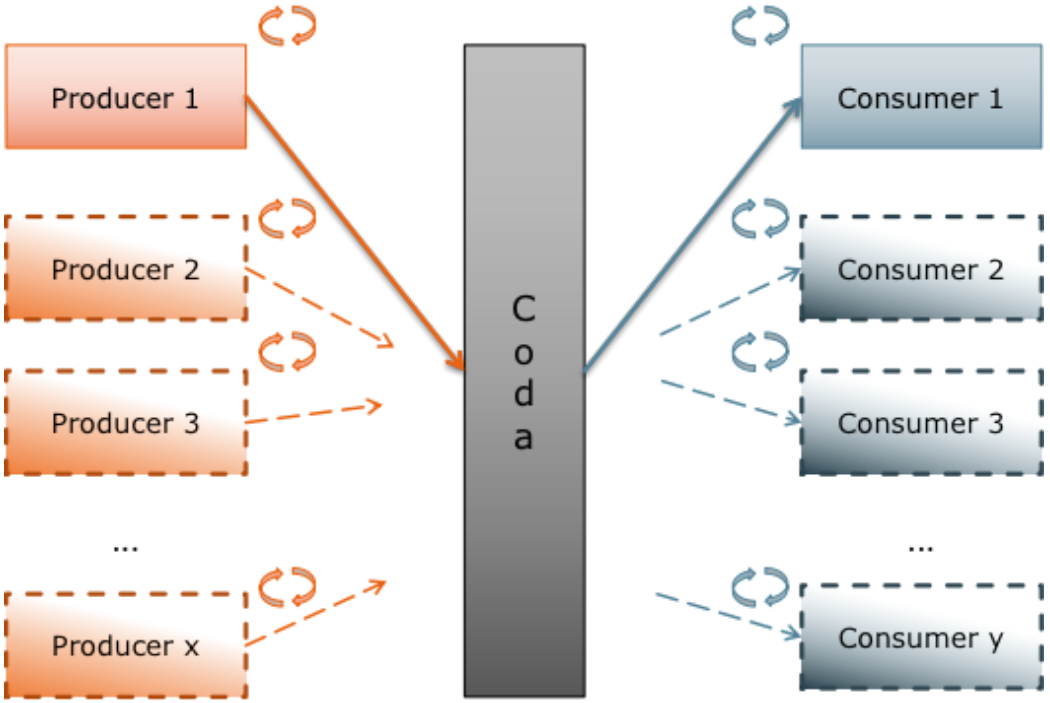


Fig. 4.8: Struttura del pattern *producer-consumer*, dove i producers/consumers tratteggiati sono ipotetici di un scenario ad *x* producers ed *y* consumers

Tale pattern, è stato implementato all'interno dei progetti appoggiandosi sulla coda thread safe descritta nel paragrafo 4.4.3, e sui due attori, rispettivamente producer e consumer, `ActionAcquireFragment` e `ScheduledThread`, così come descritto nel paragrafo 4.2.

Capitolo 5

5. Calcolo ecg medio

5.1 Elettrocardiogramma medio

L'elettrocardiogramma medio (SAECG, Signal averaged ECG), di cui è visionabile un esempio nella Fig. 5.1, è una tecnica di elaborazione del segnale per rilevare sottili anomalie nell'ECG, non visibili ad occhio nudo. Basata sul calcolo della media di più segnali elettrici provenienti dal cuore al fine di eliminare potenziali interferenze, questa tecnica è stata spesso utilizzata per rilevare piccole variazioni nel complesso QRS, le quali possono rappresentare una predisposizione verso tachiaritmie ventricolari potenzialmente pericolose.
("Signal-averaged electrocardiogram")

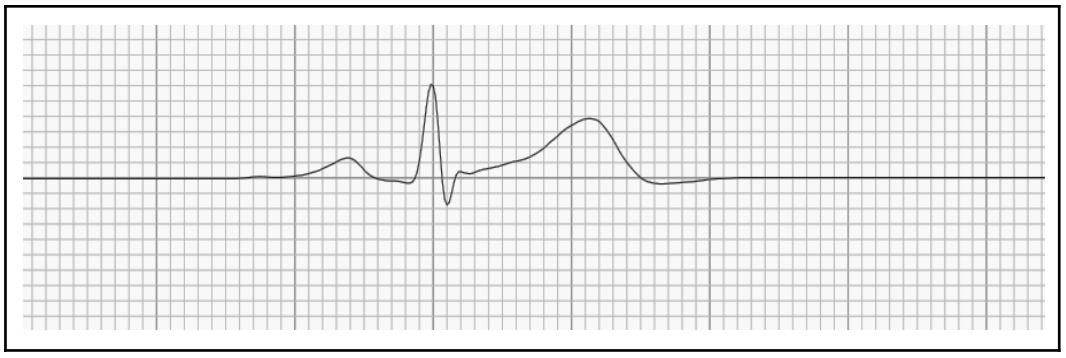


Fig. 5.1: Esempio di un ECG medio. Immagine prelevata da un'acquisizione effettuata con Kardia

5.2 Implementazione

L'attività di implementazione dell'elettrocardiogramma medio all'interno delle applicazioni si è prefissa l'idea di voler estrapolare quest'ultimo a partire da tracciati ecg completi.
Al fine di adempiere a tale ambizioso obiettivo, il lavoro è stato strutturato in diverse fasi.
La prima fase ha richiesto l'ideazione di un algoritmo che permettese di estrapolare l'ecg medio a partire da un elettrocardiogramma completo. Per fare ciò, è stato implementato un algoritmo flessibile, basato sulla programmazione funzionale, in grado di lavorare con una funzione di calcolo della media data, al fine di poter testare molteplici funzioni matematiche per poter trovare il best-fit.
Successivamente ci si è concentrati sul fine-tuning del risultato, sulla sua validazione e sulla sua corretta visualizzazione, mediante tecniche di padding dei dati ottenuti fino ad una lunghezza prefissata.

5.2.1 Algoritmo

5.2.1.1 Struttura dell'API

Per la sua implementazione si è deciso di avvalersi della cosiddetta "programmazione funzionale", al fine di poterlo rendere indipendente da una prefissata implementazione di funzione che calcoli il valor medio di un insieme di dati.
La programmazione funzionale (FP, functional programming) è un approccio di tipo dichiarativo, all'interno del quale il focus principale è su *cosa* va ottenuto piuttosto che sul *come* ottenerlo (tipico invece della programmazione imperativa). La FP delega dunque i dettagli a particolari funzioni o implementazioni di interfacce. Questo stile di programmazione permette di scorporare in maniera più puntuale le varie azioni da eseguire, dando frutto ad un codice ben organizzato e atomico, capace di cambiare del tutto risultato a seconda della particolare implementazione che si dà ad una sua precisa parte.

Il prototipo risultante dell'API, dunque, è il seguente

```
public static double[]ecgMedio(
          double[] input,
          boolean skipPartial,
          int minimumSize,
          AveragingFunction averagingFunction
)
```

Dove AveragingFunction è una generica interfaccia rappresentante una funzione matematica in grado di calcolare il valor medio di un insieme di dati.
Tale interfaccia presenta tre metodi:

1. `public String getName()`
   Ritorna il nome della funzione di averaging in questione, utilizzato per essere visualizzato sopra il relativo grafico durante le fasi di test;
2. `public Double getNeutralValue()`
   Ritorna il valore neutrale dell'algoritmo di averaging, se presente;
3. `public Double invoke(List<Double> values)`
   Invoca l'algoritmo di averaging che ritorna un Double come risultato.

5.2.1.2 Struttura dell'algoritmo

L'algoritmo implementato è nato intorno all'idea di un terzo tesista, Roberto Bruno, che, nell'ambito di questo progetto, ha lavorato su altri temi.
L'algoritmo si occupa innanzitutto dell'individuazione dei picchi R all'interno dell'elettrocardiogramma completo.
Si procede dunque all'identificazione di un intorno di questi ultimi, al loro allineamento insieme ai relativi intorni e al calcolo del valor medio dei dati allineati tramite una specifica funzione matematica.

Nella Fig. 5.2, composta da tre parti, è possibile visualizzare una rappresentazione grafica concettuale delle procedure che vengono eseguite dall'algoritmo.

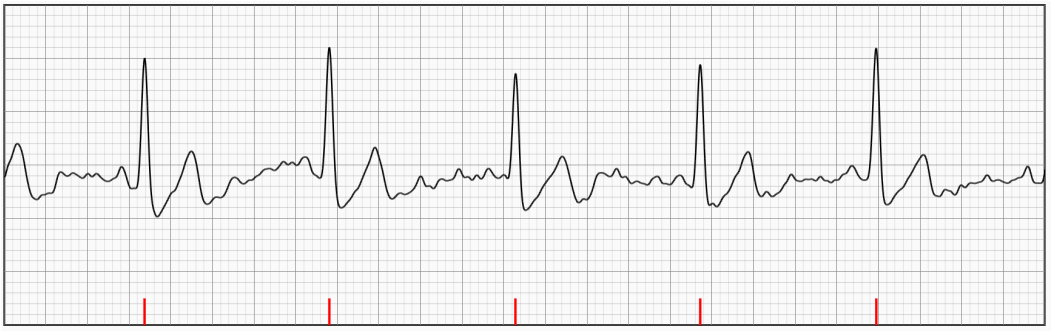


Fig. 5.2.1: Individuazione dei picchi R

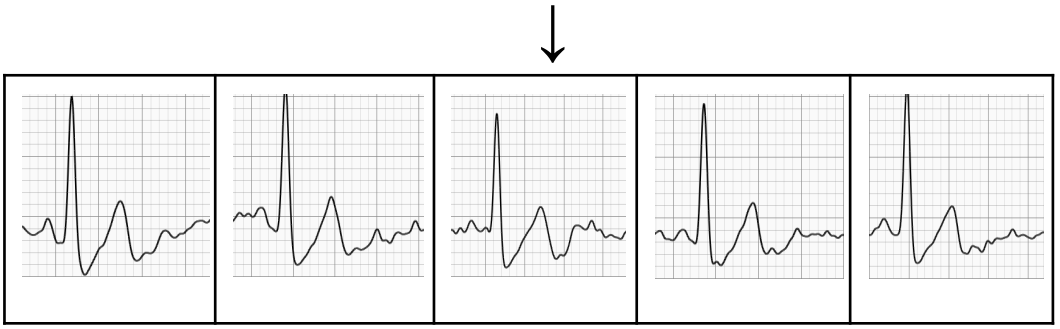


Fig. 5.2.2: Individuazione degli intorni dei picchi

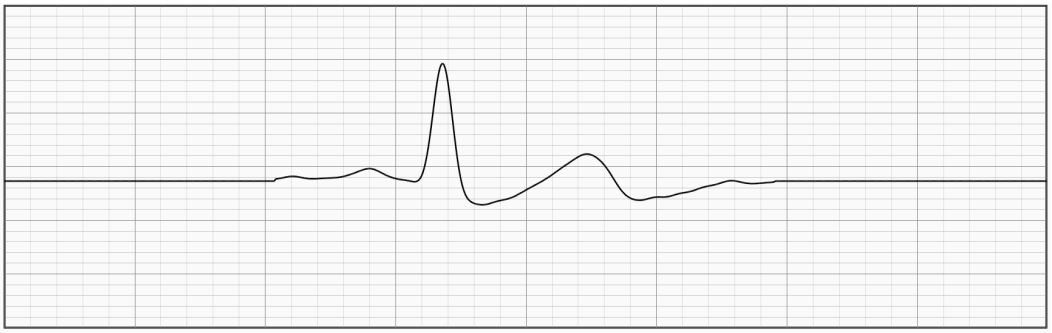


Fig. 5.2.3: Risultato della media degli intorni individuati

L'individuazione dei picchi R viene effettuata basandosi sull'identificazione dei massimi e dei minimi locali, sul confronto della loro distanza, dei valori rispetto ad una soglia media e sul confronto della pendenza del loro tratto. Per maggiori dettagli riguardo a tale tecnica si rimanda alla tesi di Giacomo Gorga.

Successivamente alla loro individuazione, viene calcolata la distanza media tra i picchi R, individuando un raggio intorno a questi ultimi:

- $\frac{1}{3}$ della distanza media a sinistra del picco;
- $\frac{2}{3}$ della distanza media a destra del picco.

La scelta dei suddetti rapporti si basa sul rapporto medio tra la lunghezza dell'intervallo PR e la lunghezza dell'intervallo RT.
Se l'intorno del picco non è sufficientemente grande, a seconda del valore del flag *skipPartial*, il picco viene saltato o, in alternativa, viene effettuato un padding. Il padding viene effettuato a sinistra in caso di un intorno sinistro, oppure a destra nel caso di un intorno destro, con il valore neutro della funzione di media utilizzata, se presente, oppure con il valore 1.0, individuato sperimentalmente come valore medio delle acquisizioni effettuate con successo.
Si procede, in seguito, alla trasformazione dell'elettrocardiogramma completo in una matrice rappresentata nella Fig. 5.3. Più nel dettaglio: per ogni picco vengono presi i sopracitati intorni e vengono inseriti in una matrice organizzata per colonne, portando l'ECG ad assumere la seguente organizzazione, dove $P_{k,x}$ è il valore x dell'intorno del picco P_k.

$$\begin{bmatrix} P_{0-0} & \cdots & P_{k-0} \\ \vdots & \ddots & \vdots \\ P_{0-x} & \cdots & P_{k-x} \end{bmatrix}$$

Fig. 5.3: Organizzazione matriciale dell'ECG

Viene dunque effettuato il calcolo del valor medio per riga, come rappresentato nella Fig. 5.4, mediante la particolare implementazione della funzione di media utilizzata.

Il valore ottenuto, chiamato qui per comodità *ecgMedio*, è il risultato temporaneo dell'ecg medio.

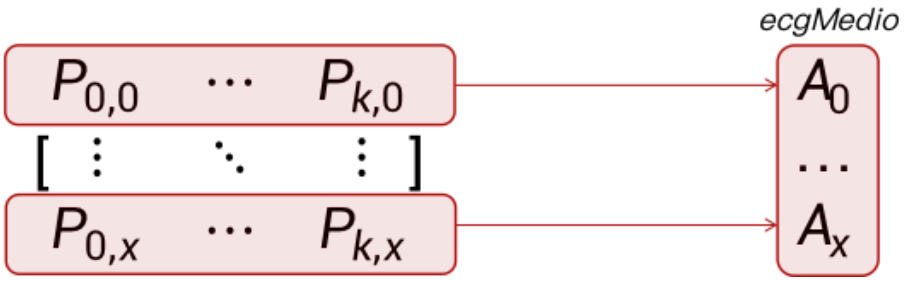


Fig. 5.4: Calcolo del valor medio della componente *x* dell'ECG a partire dalla sua organizzazione matriciale

5.2.2 Fine tuning del risultato

Una volta ottenuto il risultato temporaneo precedentemente illustrato, l'algoritmo procede ad un suo fine tuning. Più nel dettaglio, viene calcolato, con la specifica funzione di media, il valore medio di *ecgMedio*, chiamato qui per comodità *ema (ecgMedioAverage)*.
Si ricerca, all'interno di *ecgMedio*, valori non validi (*NaN*), potenzialmente ottenuti con particolari operazioni della funzione di media utilizzata (quali divisioni per zero, radici a indice pari di numeri negativi, etc) e li si sostituisce con *ema*.
Il valore ottenuto è il risultato temporaneo dell'ecg Medio ed è il centro del grafico, evidenziato in rosso, della Fig. 5.5.

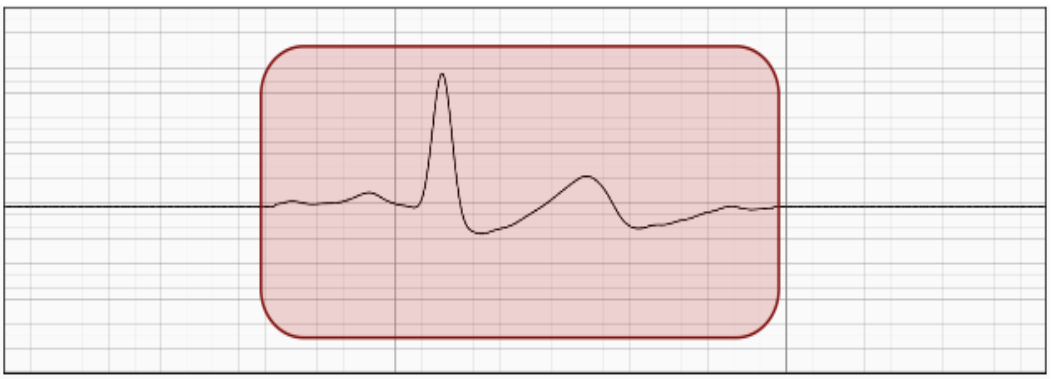


Fig. 5.5: Risultato dell'ECG medio dopo fine tuning

5.2.3 Validazione del risultato

Il risultato così ottenuto necessita di una validazione al fine di poter individuare potenziali casi in cui il risultato possa essere privo di significato, dovuto ad acquisizioni non totalmente riuscite (ad esempio il paziente ha tolto le dita dagli elettrodi a metà acquisizione, oppure si è mosso durante l'acquisizione, etc) o con eccessivo disturbo dei dati.
In particolare, ci si aspetta che il risultato dell'ECG medio abbia un unico picco. Per verificare che il risultato ottenuto sia conforme, si utilizza la funzione di individuazione dei picchi citata al paragrafo 5.2.1.2.
L'ECG medio, inoltre, non deve presentare scostamenti troppo grandi o divergenze. Per controllare che il risultato aderisca a tale caratteristica

- si calcola la sua moda e si controlla che il valore di *ema* non discosti da quest'ultimo più di 0.15;
- si calcola il suo minimo e si controlla che il valore non disti da *ema* più di 1.0;
- si calcola il suo massimo e si controlla che il valore non disti da *ema* più di 3.0.

Le costanti di qui sopra sono state individuate sperimentalmente con un gran numero di acquisizioni da parte di più individui.
Se il risultato ottenuto non è valido, l'API dell'algoritmo ritorna *null*, valore che viene controllato per mostrare eventuali messaggi di errore.

5.2.4 Visualizzazione del risultato

Come illustrato nella Fig. 5.5, il risultato finora ottenuto rappresenta la parte centrale del grafico a cui si vuole ambire.
Seppur sia quest'ultima la parte di maggior interesse, in quanto contiene le maggiori informazioni sull'andamento medio dell'elettrocardiogramma, incluso il discostamento da un valore di ground rappresentato dai valori iniziali e quelli finali, non bisogna tralasciare un importante aspetto: la facilità di visualizzazione di tali dati. Visualizzando soltanto un grafico come quello rappresentato nella Fig. 5.6, risulta infatti difficoltoso individuare il grado di scostamento dei picchi da un valore di ground, nonché apprezzare in maniera immediata il movimento medio dell'ECG.

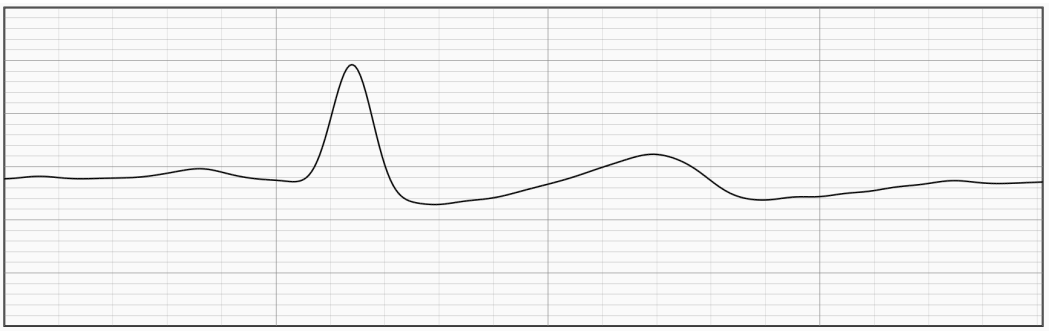


Fig. 5.6: Risultato dell'ECG medio senza padding

Si è rivelato dunque necessario effettuare un padding del risultato, in misura differente a seconda dello scenario di visualizzazione dell'ECG medio (in app, visibile nella Fig. 5.7, oppure in pdf, visibile nella Fig. 5.8).
Tale padding viene effettuato inserendo metà dei valori necessari all'inizio del vettore del risultato e metà alla fine, utilizzando *ema* come valore di padding, portando ai due risultati sotto rappresentati.
In caso l'API di calcolo dell'ECG medio ritorni il valore *null*, segno che il grafico ottenuto non sia risultato valido, viene mostrato un messaggio di errore.

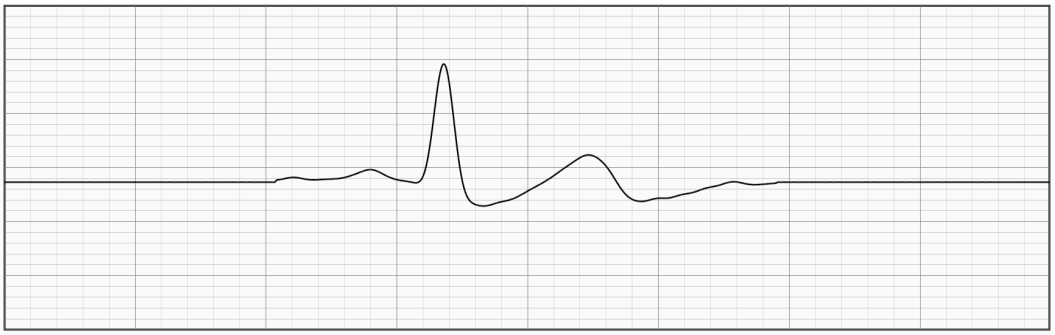


Fig. 5.7: Risultato in app dell'ECG medio con padding

Fig. 5.8: Risultato in pdf dell'ECG medio con padding

5.2.5 Funzioni di calcolo del valor medio

Come descritto al paragrafo 5.2.1.1, l'algoritmo precedentemente illustrato e la relativa API sono stati studiati ed implementati per essere indipendenti da una particolare implementazione di una funzione di calcolo del valor medio, con lo scopo di poter testare molteplici funzioni di averaging, mettendo a confronto il loro comportamento in diverse situazioni.
In particolare, le funzioni implementate, che verranno descritte in seguito, sono:

1. Media aritmetica
2. Media geometrica
3. Media armonica
4. Media aritmetico geometrica
5. Mediana
6. Moda

5.2.5.1 Note sulla Media aritmetica

La media aritmetica di un vettore *a* di *n* valori è definita nel seguente modo:

$$M_a = \frac{1}{n} \sum_{i=0}^{n} a_i$$

Benché la media aritmetica venga spesso utilizzata nell'ambito statistico per far riferimento alle tendenze, non è in grado di fornire un risultato robusto in quanto risente fortemente dei valori anomali (outliers).

5.2.5.2 Note sulla Media geometrica

La media geometrica di un vettore *a* di *n* valori è definita nel seguente modo:

$$M_g = \sqrt[n]{\prod_{i=1}^{n} a_i}$$

La media geometrica può portare come risultato *NaN* qualora *n* sia pari e la moltiplicatoria porti ad un valore negativo.

5.2.5.3 Note sulla Media armonica

La media armonica di un vettore *a* di *n* valori è definita come il reciproco della media dei reciproci:

$$M_h = \left(\frac{1}{n}\sum_{i=0}^{n}\frac{1}{a_i}\right)^{-1}$$

La media geometrica può portare come risultato *NaN* qualora un qualsiasi elemento del vettore abbia valore 0.

5.2.5.4 Generalizzazione di funzioni a due parametri

Tra le funzioni testate che verranno descritte in seguito, vi sono alcune che, di base, lavorano solamente su due parametri; ovvero sono in grado di calcolare il valor medio di due parametri *a* e *b*. Lo scenario di utilizzo, tuttavia, richiede la possibilità di calcolare il valor medio su un insieme di *n* parametri. Al fine di adattare in maniera semplice e unificata tale famiglia di funzioni, è stata creata una classe astratta che, mediante un approccio ricorsivo basato su “divide and conquer”, permette di raggrupparle in un’unica generalizzazione.
La classe in questione è *AbstractTwoValuesAveragingFunction* e l’approccio che segue può essere rappresentato mediante il diagramma presente nella Fig. 5.9, dove la parte tratteggiata rappresenta il comportamento nel caso di un vettore con un numero di elementi dispari.

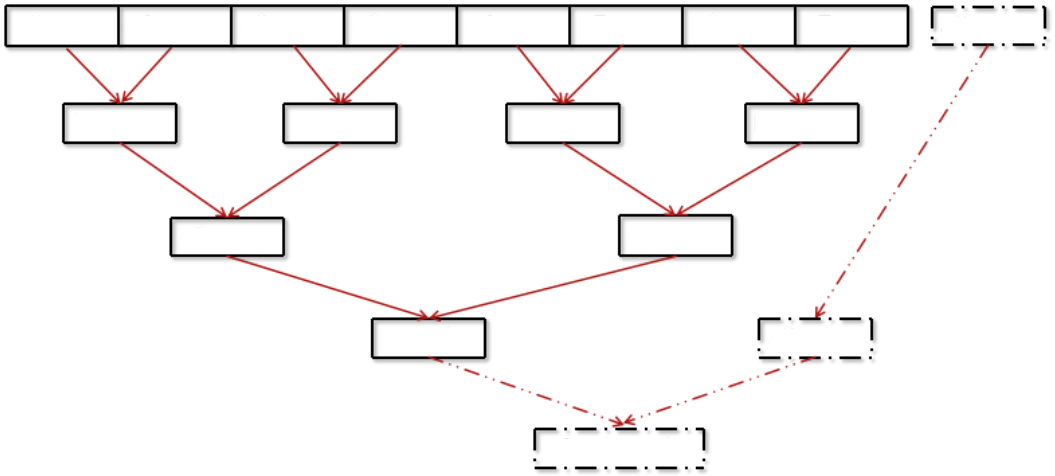


Fig. 5.9: Struttura dell'algoritmo di generalizzazione delle funzioni di calcolo della media a due parametri

5.2.5.5 Note sulla media aritmetico-geometrica

La media aritmetico-geometrica di due valori *x* ed *y* è definita nel seguente modo:

- Chiamati *x* ed *y* come a_0 e g_0;
- Vengono definite due sequenze indipendenti a_n e g_n come segue:
  - $a_{n+1} = \frac{1}{2}(a_n + g_n)$
  - $g_{n+1} = \sqrt{a_n g_n}$
- Queste sequenze convergono allo stesso numero, la media aritmetico-geometrica di *x* ed *y*.

Le proprietà di convergenza di queste due sequenze sono ben note e sono estremamente rapide: in media dopo 5 iterazioni si converge fino alla ventesima cifra decimale.
("Arithmetic–geometric mean")

L'implementazione di questa funzione è stata effettuata basandosi sull'algoritmo descritto nel paragrafo 5.2.5.7 e contenuto nella classe *AbstractTwoValuesAveragingFunction*.
Il numero di iterazioni utilizzato è stato fissato a 5 e come risultato unico è stato preso il valor medio delle due sequenze sopra descritte.

5.2.5.6 Note sulla Mediana

Per calcolare la mediana di un vettore *a* di *n* valori, si procede ordinando il vettore in ordine crescente.
Successivamente, se il numero di dati *n* è dispari, la mediana corrisponde al valore centrale, ovvero al valore che occupa la posizione $\frac{n+1}{2}$, altrimenti corrisponde alla media aritmetica dei valori che occupano la posizione $\frac{n}{2}$ ed $\frac{n+1}{2}$.

5.2.5.7 Note sulla Moda

La moda in un vettore *a* di *n* elementi viene definita come il valore caratterizzato dalla massima frequenza, ovvero quel valore che viene ripetuto, all'interno di *a,* il maggior numero di volte.
Una possibile implementazione del calcolo della moda, nonché quella che è stata utilizzata all'interno del nostro studio, prevede la trasformazione del vettore *a* in una struttura dati a mappa, all'interno della quale le chiavi rappresentano gli elementi del vettore, mentre i valori sono dei contatori della relativa frequenza. La moda sarà data dall'elemento che avrà il valore del contatore massimo.

5.3 Risultati e conclusioni

Di seguito vengono illustrati alcuni degli esperimenti effettuati.
Si può notare come le diverse funzioni di averaging si comportino in maniera abbastanza simile in caso di acquisizioni che non presentano anomalie (tabella 5.1 e 5.2).
Le funzioni di moda e mediana, tuttavia, portano a un grafico con una forma meno regolare rispetto agli altri.
Il comportamento delle funzioni, invece, cambia molto in caso di acquisizioni anomale (tabelle dalla 5.3 alla 5.6).
Le funzioni di media aritmetica, moda e mediana spesso possono portare a risultati la cui validazione ha successo ma che, analizzati da un essere umano, sono privi di senso (tabelle 5.4, 5.5 e 5.6).
Le funzioni di media geometrica e media armonica, d'altra parte, spesso possono portare ad operazioni in cui il risultato non è calcolabile (quali radici pari di numeri negativi e divisioni per zero).
In base a quanto appena descritto, le funzioni di media quadratica e media aritmetico-geometrica sono, a nostro avviso, le candidate migliori in qualità di stabilità dei calcoli effettuati nonché qualità del grafico ottenuto.

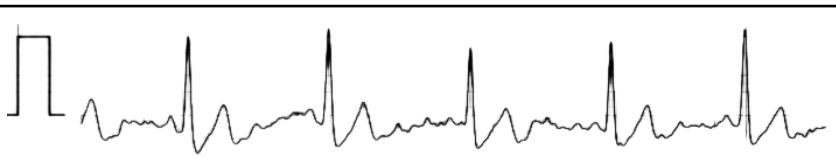
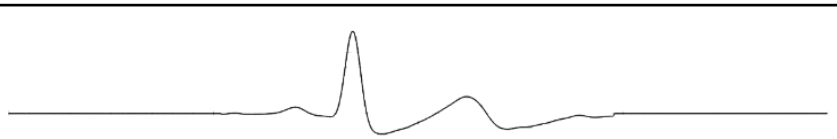
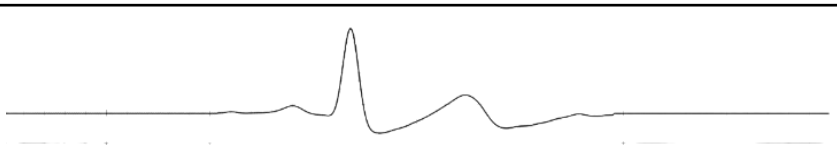
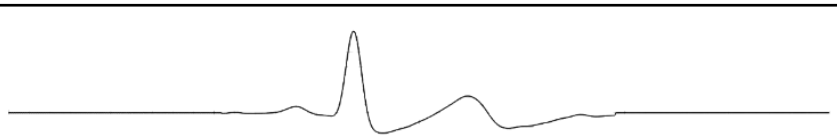
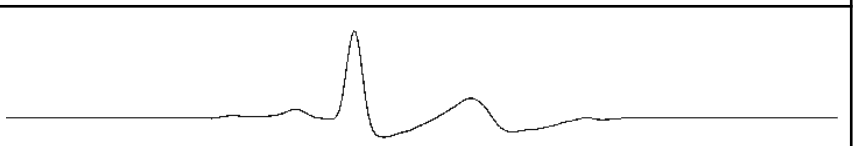
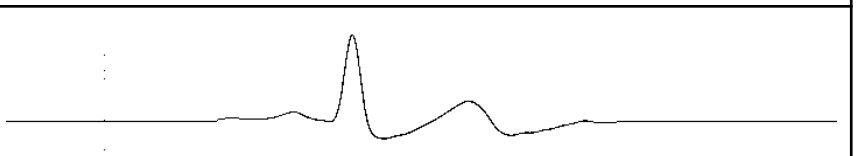
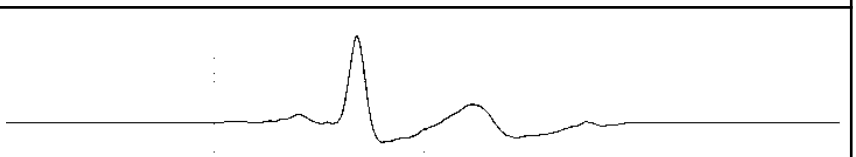
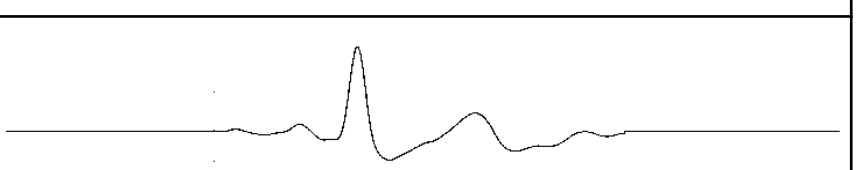
Ecg di riferimento	
Media aritmetica	
Media geometrica	
Media quadratica	
Media armonica	
Media aritmetico geometrica	
Mediana	
Moda	

Tabella 5.1: Confronto degli algoritmi di calcolo della media su un ecg corretto

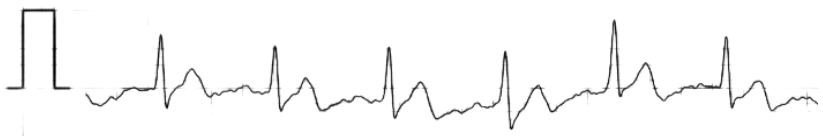
Ecg di riferimento	
Media aritmetica	
Media geometrica	
Media quadratica	
Media armonica	
Media aritmetico geometrica	
Mediana	
Moda	

Tabella 5.2: Confronto degli algoritmi di calcolo della media su un ecg corretto

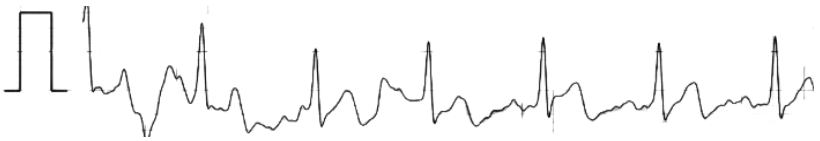
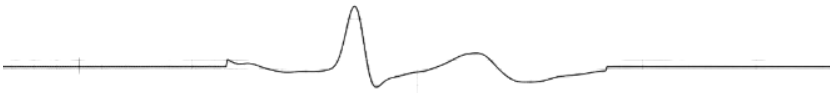
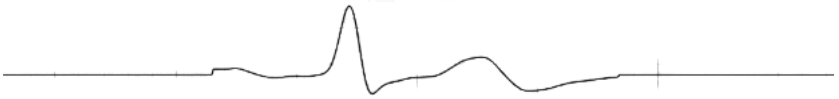
Ecg di riferimento	
Media aritmetica	
Media geometrica	
Media quadratica	
Media armonica	
Media aritmetico geometrica	
Mediana	
Moda	Grafico non valido

Tabella 5.3: Confronto degli algoritmi di calcolo della media su un ecg scostante

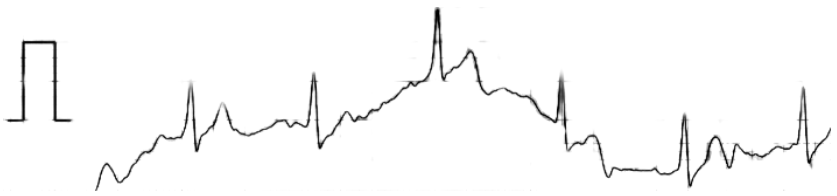
Ecg di riferimento	
Media aritmetica	
Media geometrica	
Media quadratica	
Media armonica	
Media aritmetico geometrica	
Mediana	
Moda	

Tabella 5.4: Confronto degli algoritmi di calcolo della media su un ecg fortemente scostante

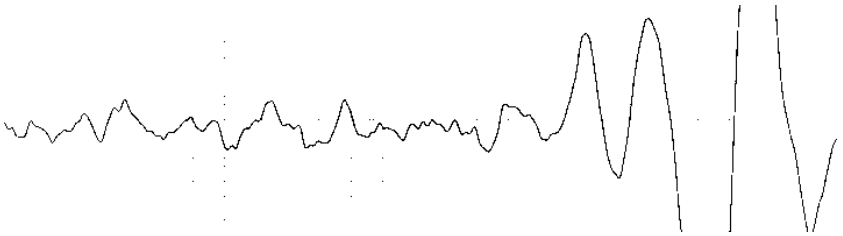
Ecg di riferimento	
Media aritmetica	
Media geometrica	Grafico non valido
Media quadratica	Grafico non valido
Media armonica	Grafico non valido
Media aritmetico geometrica	Grafico non valido
Mediana	Grafico non valido
Moda	

Tabella 5.5: Confronto degli algoritmi di calcolo della media su un ecg errato

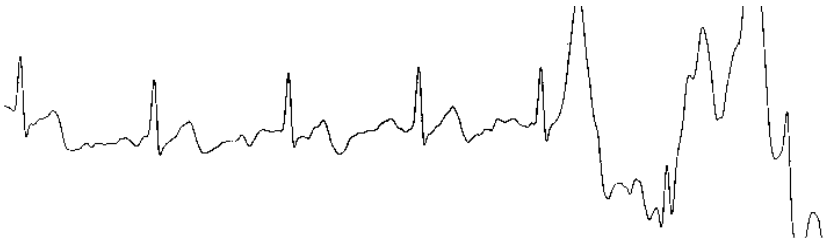
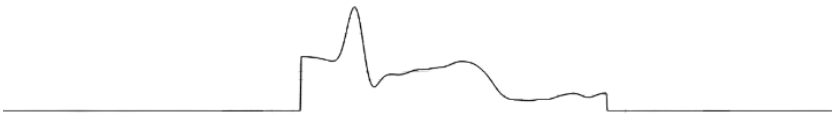
Ecg di riferimento	
Media aritmetica	
Media geometrica	Grafico non valido
Media quadratica	Grafico non valido
Media armonica	Grafico non valido
Media aritmetico geometrica	Grafico non valido
Mediana	
Moda	

Tabella 5.6: Confronto degli algoritmi di calcolo della media su un ecg parzialmente errato

Capitolo 6

6. Riallineamento dell'app PulsEcg

6.1 Dispositivo PulsEcg

Il dispositivo PulsEcg presente nella Fig. 6.1, è caratterizzato principalmente da due elettrodi, da una batteria e da due circuiti, uno per la misurazione dell'elettrocardiogramma e l'altro per la misurazione della fotopletismografia, il tutto montato su un mockup stampato in 3D.
Il primo circuito è costituito da:

- un connettore degli elettrodi, un connettore USB, un connettore piatto e un connettore della batteria che formano l'interazione tra il PCB e il mondo esterno;
- un modulo Bluetooth 4.0 per la comunicazione dei dati ad altri dispositivi, in particolare per inviare i dati all'applicazione PulsEcg sviluppata da questa tesi;
- blocchi di alimentazione:
    - Battery Charger e Battery Gauge entrambi usati per controllare lo stato della batteria e la sua modalità di carica
    - Voltage Regulator e Analog Power Domain, che generano una tensione di alimentazione stabile e una tensione di riferimento per tutti gli altri componenti
- filtri utilizzati per il condizionamento del segnale elettrocardiografico;
- un microcontrollore Texas Instruments CC2640R2F per controllare tutti gli altri blocchi e raccogliere i dati dell'elettrocardiogramma attraverso il suo convertitore analogico-digitale (ADC) interno e il valore di carica della batteria attraverso la comunicazione I2C.

Il secondo circuito è costituito da:

- un connettore piatto che viene usato per collegare il circuito stampato al circuito principale attraverso il cavo piatto che porta le linee di alimentazione, le linee I2C e la linea legata all'interrupt;
- un sensore Maxim Integrated MAXM86161 per ricavare i dati utili per la fotopletismografia, utilizzando la luce rossa e infrarossa.

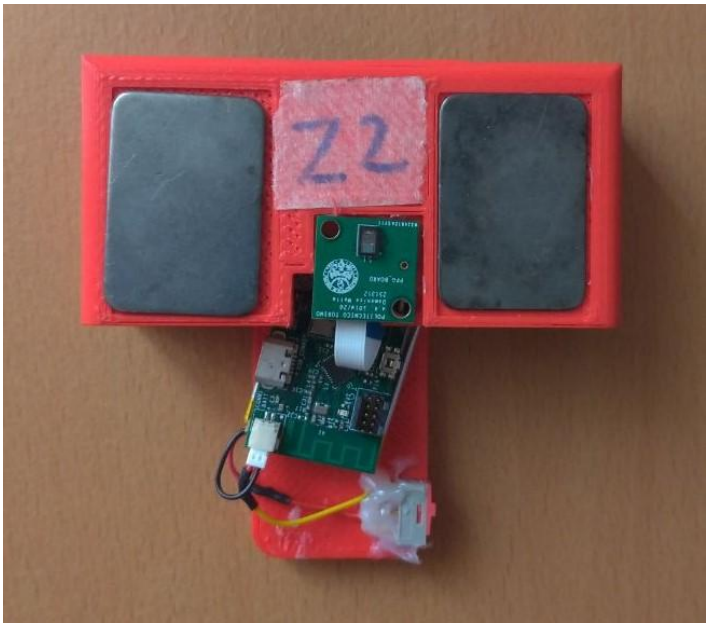


Fig. 6.1: Dispositivo PulsEcg

6.2 Riallineamento UI

Nell'applicazione PulsEcg, della quale era stata effettuata una copia per creare l'applicazione EcgMed, sono state importate le modifiche apportate su quest'ultima, una volta completati gli sviluppi.

In questa fase è stato importato il codice relativo al redesign (trattato nel paragrafo 3.2), quindi sono state modificate le schermate dell'applicazione e la relativa navigazione.
Nelle Figg. 6.2 e 6.3 è possibile osservare le differenze grafiche tra le due applicazioni.

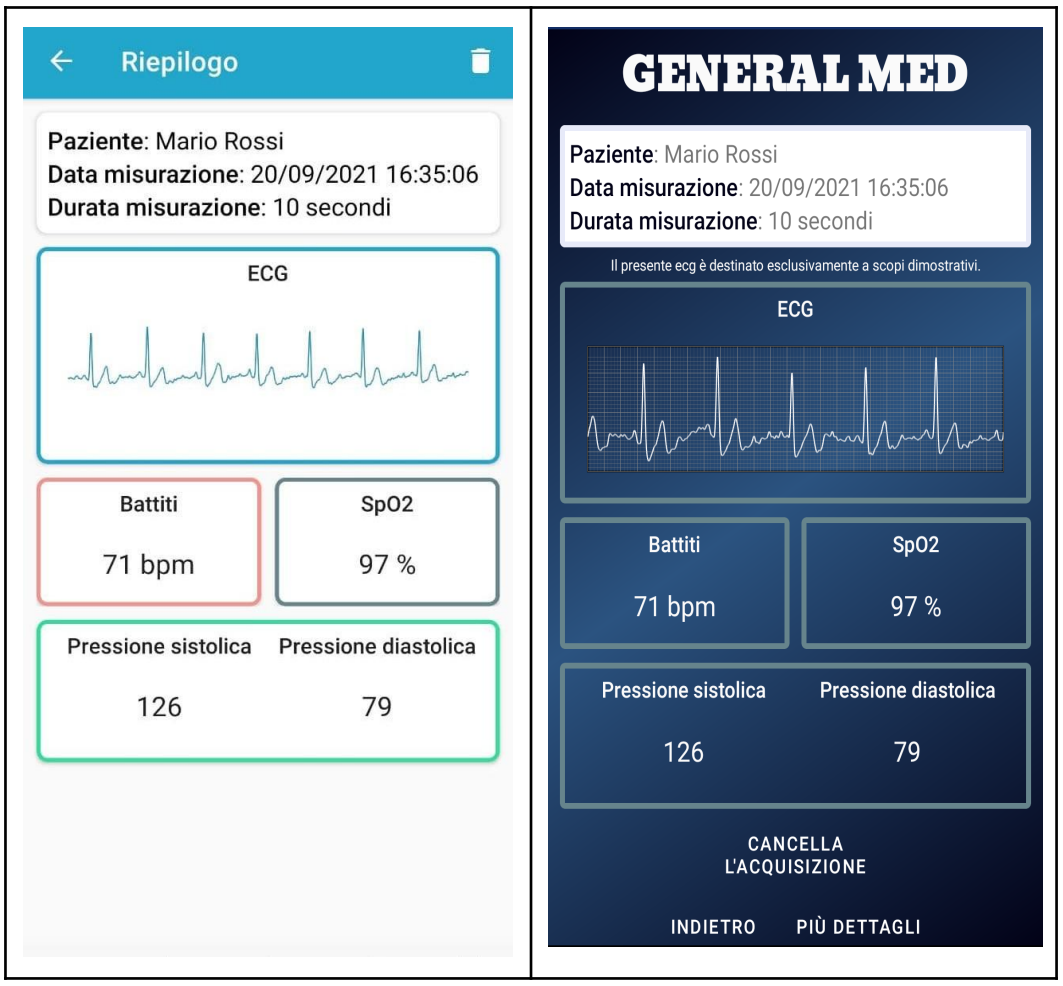


Fig. 6.2:
A sinistra la schermata di riepilogo dell'applicazione PulsEcg prima del riallineamento, a destra la schermata di riepilogo dell'applicazione PulsEcg dopo il riallineamento

Fig. 6.3:
A sinistra sono mostrate le tre schermate di visualizzazione dei dettagli dell'applicazione PulsEcg prima del riallineamento.
A destra sono mostrate le tre schermate di visualizzazione dei dettagli dell'applicazione PulsEcg dopo il riallineamento

6.3 Riallineamento funzionalità

In questa fase sono state importate le funzionalità aggiuntive ed in particolare è stata aggiunta l'implementazione della finestra di dialog all'apertura dell'applicazione, l'implementazione della visualizzazione live dell'ecg in fase di acquisizione (trattato al capitolo 4) e l'implementazione dell'algoritmo per il calcolo e la visualizzazione dell'ecg medio (trattato al capitolo 5).

Nelle Figg. 6.4, 6.5 e 6.6 è possibile osservare le schermate dell'applicazione PulsEcg con l'aggiunta delle nuove funzionalità.

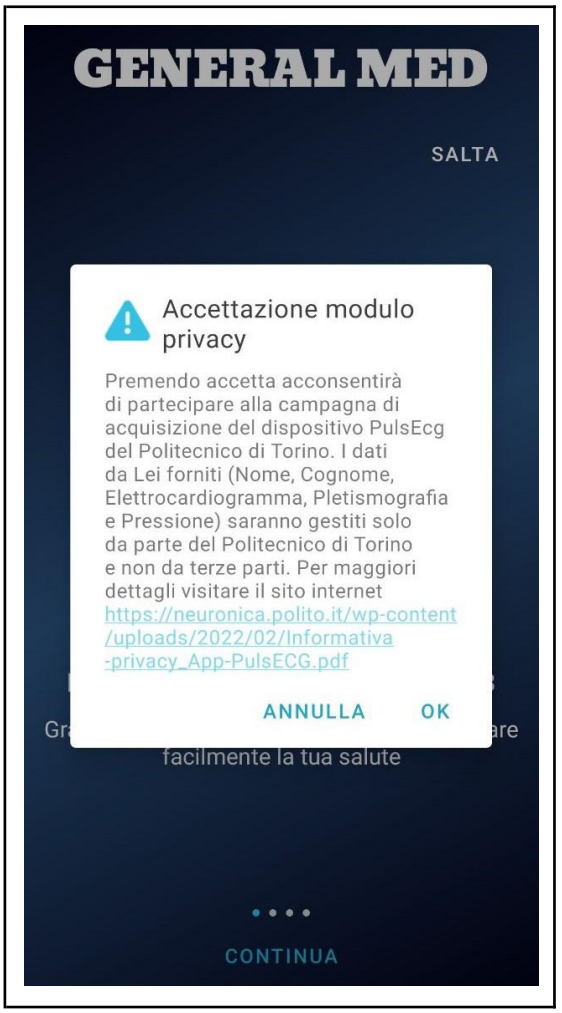


Fig. 6.4: Finestra di dialog riguardo l'informativa sulla privacy

Fig. 6.5: Schermata di acquisizione con visualizzazione live dell'ecg

Fig. 6.6: Schermata di visualizzazione dell'ecg medio

Per quanto riguarda il file PDF generato dall'applicazione, al suo interno, alle informazioni già presenti, è stato aggiunto il grafico dell'ecg medio. Nelle Figg. 6.7 e 6.8 è possibile vedere le modifiche effettuate.

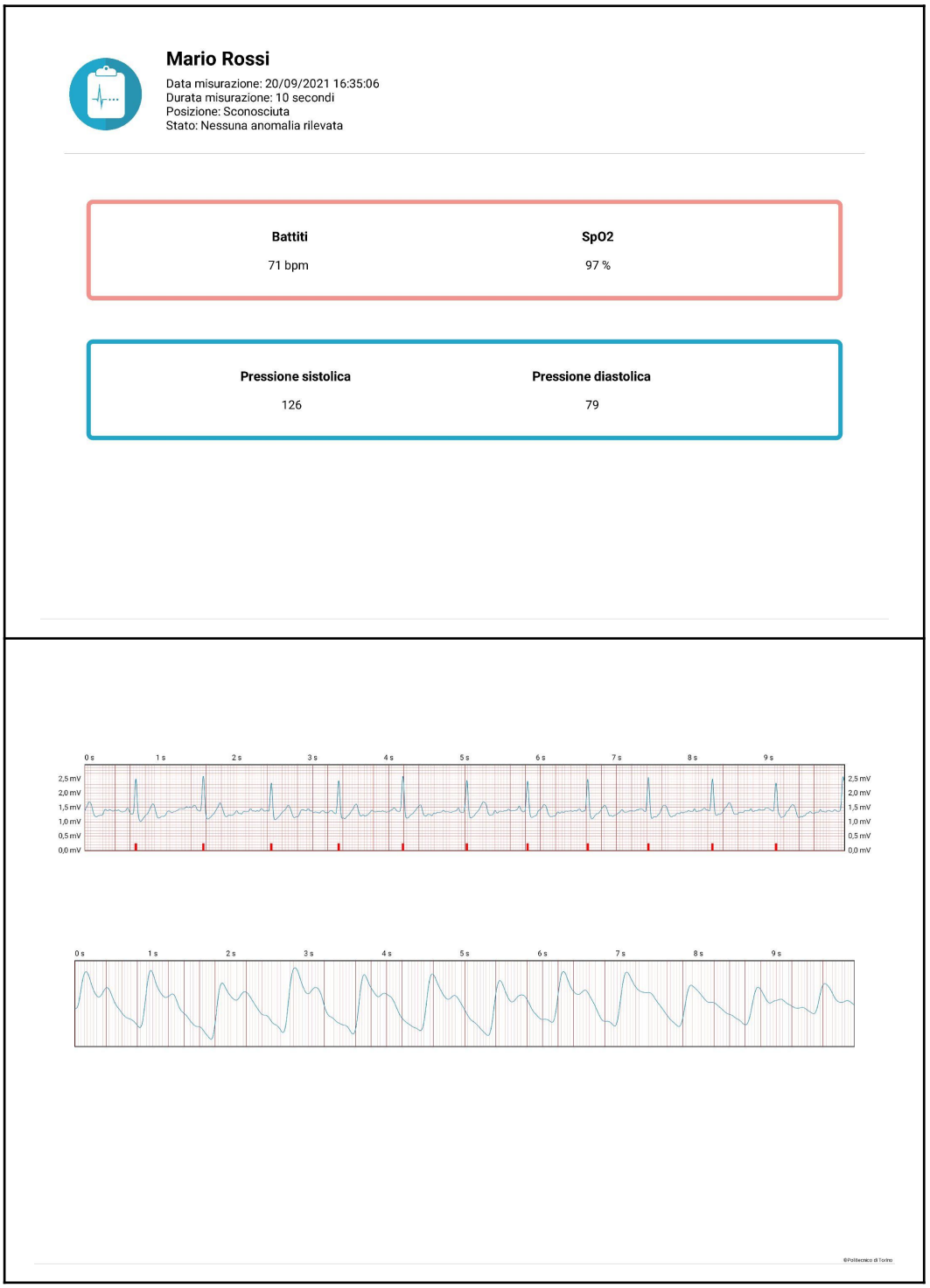


Fig. 6.7: PDF generato dall'applicazione PulsEcg prima del riallineamento

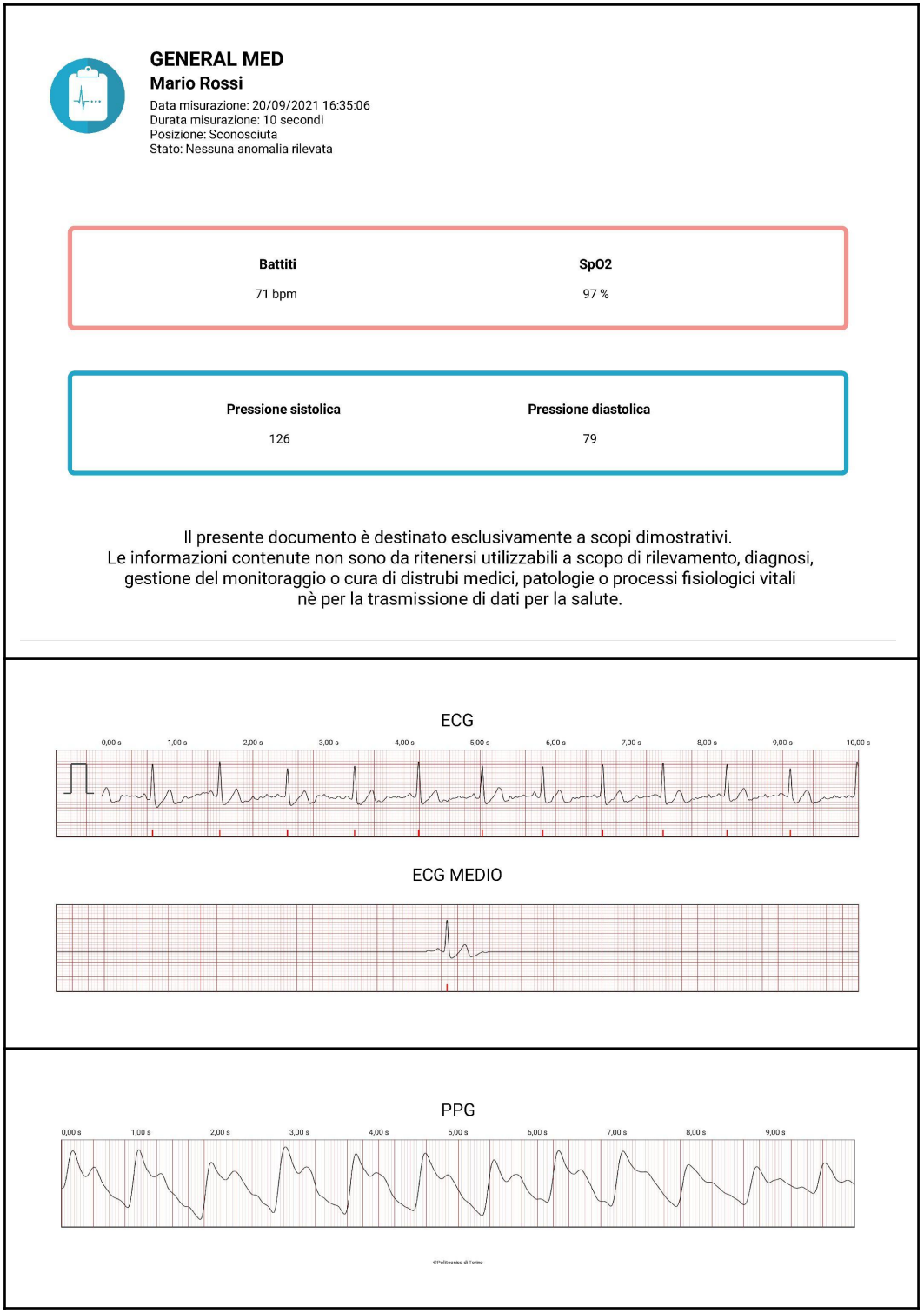


Fig. 6.8: PDF generato dall'applicazione PulsEcg dopo il riallineamento

Capitolo 7

7. Conclusioni

Per quanto descritto nei capitoli precedenti, il lavoro svolto si può riassumere nei seguenti punti:

1. Studio del progetto e dell'applicazione Android PulsEcg preesistente, comprendendo la sua struttura complessiva, l'obiettivo e le scelte tecniche effettuate, nonché le loro motivazioni;

2. Creazione dell'applicazione EcgMed a partire dall'applicazione PulsEcg, comportando la necessità di modificare tutti gli identificativi dell'applicazione per consentire la loro coesistenza, nonché la modifica di ogni riferimento al nome dell'applicazione;

3. Refactor e riorganizzazione del codice al fine renderlo più mantenibile e facilmente leggibile: comprende la scissione tra la logica operativa (aka business logic) implementata nelle activities e la parte di visualizzazione implementata nei fragment, eliminazione della parte inerente alla misurazione e visualizzazione del ppg, introduzione di una nuova gerarchia di componenti atti ad evitare la duplicazione continua di codice nonché l'introduzione di nuovi componenti con lo scopo di scindere meglio i ruoli delle componenti logiche del codice;

4. Redesign completo dell'applicazione con lo scopo di modernizzare il suo aspetto e il modo in cui l'utente deve interagire con quest'ultima, guidandolo in ogni step all'interno della navigazione e fornendogli una descrizione su cosa può fare in ciascuna schermata; approccio adatto ad un utente altamente inesperto (platea principale dell'applicazione);

5. Implementazione della visualizzazione live dell'ECG, sempre facente parte del piano di redesign completo dell'applicazione, avente come scopo quello di fornire un feedback visivo all'utente durante l'acquisizione;

6. Implementazione dell'algoritmo per il calcolo e la visualizzazione

dell'ECG medio, comprendendo lo studio e l'implementazione delle varie funzioni matematiche utilizzate, nonché l'acquisizione di molteplici ECG da numerose persone, al fine della validazione dei risultati ottenuti;

7. Riallineamento dell'applicazione PulsEcg ad EcgMed, allineando l'implementazione delle nuove funzionalità della seconda a quelle aggiuntive della prima.

7.1 Sviluppi futuri

I dispositivi EcgMed e PulsEcg con relative applicazioni hanno ampi margini di miglioramento.
I possibili miglioramenti da effettuare sono:

1. rendere possibile da parte del dispositivo l'invio dell'informazione sulla percentuale di batteria non appena si effettua la connessione all'applicazione, mentre nello stato attuale la suddetta informazione viene mandata solo in fase di acquisizione;

2. rendere possibile l'interruzione dell'invio dei dati da parte del dispositivo in caso di acquisizione interrotta dall'utente tramite l'applicazione;

3. rivedere i filtri in modo da ottenere un segnale meno rumoroso e più preciso;

4. rivedere l'elaborazione del segnale in fase di acquisizione in modo da ottenere un andamento del segnale più simile a quello calcolato dopo aver terminato l'acquisizione;

5. aggiungere un sensore per misurare la temperatura in modo da segnalare l'utente in caso di valori elevati;

6. aggiungere un sensore di movimento in grado di rilevare una caduta da parte dell'utente e permettere all'applicazione, nel caso in cui l'utente non si muova per un significativo lasso di tempo, di far partire automaticamente una chiamata ai servizi di emergenza e mandare un messaggio ai propri contatti di emergenza includendo la sua posizione.

Bibliografia

“Android Studio.” *Wikipedia*, https://it.wikipedia.org/wiki/Android_Studio. Accessed 17 March 2022.

“Arithmetic–geometric mean.” *Wikipedia*, https://en.wikipedia.org/wiki/Arithmetic%E2%80%93geometric_mean. Accessed 22 February 2022.

“Arithmetic mean.” *Wikipedia*, https://en.wikipedia.org/wiki/Arithmetic_mean. Accessed 21 March 2022.

Bertelli, Giulia. “Saturazione di Ossigeno.” *My Personal Trainer*, https://www.my-personaltrainer.it/bpco/saturazione-di-ossigeno.html. Accessed 6 March 2022.

Borgacci, Riccardo. “Frequenza Cardiaca.” *My Personal Trainer*, https://www.my-personaltrainer.it/allenamento/frequenza-cardiaca/frequenza-cardiaca.html. Accessed 6 March 2022.

“Bradicardia - Frequenza cardiaca.” *My Personal Trainer*, https://www.my-personaltrainer.it/salute/bradicardia.html. Accessed 17 March 2022.

Chadwick, Jonathan. “People spend 4.8 hours per day on mobile apps, research shows.” *Daily Mail*, 12 January 2022, https://www.dailymail.co.uk/sciencetech/article-10394221/People-spend-4-8-hours-day-mobile-apps-research-shows.html. Accessed 6 March 2022.

“The Color Psychology of Black.” *Verywell Mind*, 13 October 2020, https://www.verywellmind.com/the-color-psychology-of-black-2795814. Accessed 7 March 2022.

“The Color Psychology of Blue.” *Verywell Mind*, 21 February 2020, https://www.verywellmind.com/the-color-psychology-of-blue-2795815. Accessed 7 March 2022.

“Configure the app module.” *Android Developers*, 25 January 2022, https://developer.android.com/studio/build/configure-app-module. Accessed 22 March 2022.

“Delegation pattern.” *Wikipedia*, https://en.wikipedia.org/wiki/Delegation_pattern. Accessed 21 March 2022.

“ECG/PPG Measurement Solution.” *Richtek Technology*, https://www.richtek.com/Design%20Support/Technical%20Document/AN057. Accessed 6 March 2022.

Einthoven, Willem, et al. “Elettrocardiogramma.” *Wikipedia*, https://it.wikipedia.org/wiki/Elettrocardiogramma. Accessed 21 February 2022.

“Geometric mean.” *Wikipedia*, https://en.wikipedia.org/wiki/Geometric_mean. Accessed 21 March 2022.

“Git Feature Branch Workflow.” *Atlassian*, https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow. Accessed 17 March 2022.

“Git Guides.” *GitHub*, https://github.com/git-guides. Accessed 17 March 2022.

“GitHub.” *Wikipedia*, https://it.wikipedia.org/wiki/GitHub. Accessed 22 March 2022.

Gorga, Giacomo. *Sviluppo di un’applicazione Android con l’integrazione di una rete neurale per il supporto a dispositivi medicali indossabili*. Ottobre 2021, Torino, Piemonte, Italia. Politecnico di Torino.

Griguolo, Antonio. “Elettrocardiogramma.” *My Personal Trainer*, https://www.my-personaltrainer.it/salute-benessere/elettrocardiogramma.html. Accessed 21 February 2022.

“Harmonic mean.” *Wikipedia*, https://en.wikipedia.org/wiki/Harmonic_mean. Accessed 21 March 2022.

Hughes, Roger. “The Producer Consumer Pattern.” *DZone*, 26 February 2013, https://dzone.com/articles/producer-consumer-pattern. Accessed 21 March 2022.

Ilaria, Randi. “Misuratore di Pressione: Cos'è? Come si Usa? Quale Scegliere e Costi.” *My Personal Trainer*, https://www.my-personaltrainer.it/benessere/misuratore-pressione.html. Accessed 6 March 2022.

“Median.” *Wikipedia*, https://en.wikipedia.org/wiki/Median. Accessed 21 March 2022.

“Mobile App Design Fundamentals: User Experience vs. User Interface.” *Clearbridge Mobile*, https://clearbridgemobile.com/mobile-app-design-fundamentals-user-experience-user-interface/. Accessed 6 March 2022.

“Mode (statistics).” *Wikipedia*, https://en.wikipedia.org/wiki/Mode_(statistics). Accessed 21 March 2022.

Mor, Einav. “App UX: All the basics you need to know in 2021.” *AppsFlyer*, 11 August 2021, https://www.appsflyer.com/blog/tips-strategy/mobile-app-ux/. Accessed 6 March 2022.

“Observer pattern.” *Wikipedia*, https://it.wikipedia.org/wiki/Observer_pattern. Accessed 21 March 2022.

“Pressione arteriosa, cos'è e come si misura.” *My Personal Trainer*, https://www.my-personaltrainer.it/ipertensione/misurare-pressione.html. Accessed 6 March 2022.

“Privacy policy per app Android.” *Iubenda*, https://www.iubenda.com/it/help/11583-privacy-policy-per-app-android. Accessed 22 March 2022.

“Producer–consumer problem.” *Wikipedia*, https://en.wikipedia.org/wiki/Producer%E2%80%93consumer_problem. Accessed 21 March 2022.

“Refactoring.” *Wikipedia*, https://it.wikipedia.org/wiki/Refactoring. Accessed 22 March 2022.

"Signal-averaged electrocardiogram." *Wikipedia*, https://en.wikipedia.org/wiki/Signal-averaged_electrocardiogram. Accessed 22 February 2022.

"Tachicardia: quando può essere pericolosa?" *Cardio Center Napoli*, 30 September 2020, https://www.cardiocenternapoli.it/cardiologia/tachicardia-quando-puo-essere-pericolosa.html. Accessed 17 March 2022.

"10 Best Practices to Enhance Your Mobile App User Experience." *Clearbridge Mobile*, https://clearbridgemobile.com/best-practices-to-enhance-your-mobile-app-user-experience/. Accessed 6 March 2022.

"Thread safety." *Wikipedia*, https://en.wikipedia.org/wiki/Thread_safety. Accessed 21 March 2022.