POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Sviluppo di un'interfaccia iOS per device medicali per la misurazione non-invasiva e senza cuffia della pressione arteriosa

Relatori

Prof. Eros PASERO

Ing. Vincenzo RANDAZZO

Candidato

Silvia FOIS

Aprile 2021

Sommario

L'evoluzione tecnologica nell'ambito dell'ingegneria biomedica e dell'intelligenza artificiale apre, nello scenario odierno, nuove possibilità di supporto alla sanità, rendendo le procedure medicali più rapide, precise e semplici.

Lo scopo di questa tesi è lo sviluppo di un'interfaccia iOS per semplificare la misurazione di alcuni importanti parametri vitali, al fine di monitorare in qualunque momento la salute del cuore, mediante un dispositivo chiamato PulsEcg. Lo strumento in questione è un bracciale dotato di elettrodi e sensori attraverso i quali è possibile acquisire rapidamente e con precisione il segnale elettrocardiografico (ECG), che descrive l'attività cardiaca del paziente, e quello pletismografico (PPG), che indica il livello di saturazione dell'ossigeno nel sangue. Grazie all'impiego dell'intelligenza artificiale, lo studio dell'andamento di questi segnali permette di stimare in maniera non invasiva i valori di pressione arteriosa del sangue, sia diastolica che sistolica, senza dover gonfiare una cuffia.

Il dispositivo è in grado di comunicare con applicazioni mobili, negli ambienti iOS e Android, sfruttando una connessione Bluetooth Low Energy. I dati raccolti dal dispositivo vengono elaborati dall'applicazione, favorendo l'identificazione di situazioni critiche in cui è necessaria un'analisi più accurata da parte del medico. Lo stesso medico potrà rilevare la presenza di malattie cardiovascolari più facilmente, grazie ai documenti prodotti dall'applicazione e condivisi dal paziente. L'interfaccia iOS è stata sviluppata in modo da poter avviare l'acquisizione da dispositivo, ricevere ed elaborare i dati, visualizzarli attraverso opportuni grafici, immagazzinarli e renderli condivisibili sui principali canali di comunicazione, quali la posta elettronica o i servizi di messaggistica istantanea. L'applicazione dispone di algoritmi di filtraggio dei segnali ECG e PPG, di una rete neurale in grado di derivare la pressione arteriosa e di meccanismi di segnalazione di eventuali anomalie cardiache. La grande novità di PulsEcg è rappresentata dall'abbandono dello sfigmomanometro, rendendo così il processo di misurazione della pressione alla portata di tutti, sia in termini di comodità che di costo. L'applicazione iOS possiede un'interfaccia molto intuitiva che garantisce una user experience adattabile, per tutte le età e per tutti i livelli di familiarità con il mondo smart. Particolare attenzione è stata posta nell'efficienza, robustezza e velocità. Infine, guardando al

futuro, il software è progettato in modo scalabile e mantenibile, in modo da poter integrare facilmente nuove funzionalità e nuovi sensori lato hardware, ad esempio sensori di temperatura e di movimento della persona.

Ringraziamenti

Prima di procedere nella trattazione dell'elaborato, vorrei spendere qualche parola per ringraziare coloro senza i quali questo traguardo sarebbe stato difficile, se non impossibile, da raggiungere.

Prima di tutto ringrazio il mio relatore, il professor Pasero, la cui proposta di tesi mi ha riempita d'entusiasmo, sia perché riguarda un ambito di mio estremo interesse, sia perché ha rappresentato l'occasione per entrare in contatto con un ambiente lavorativo piacevolissimo. Un ringraziamento va poi al mio corelatore, Vincenzo, per il sostegno costante e per la precisione che, se a volte può essere stata frustrante, mi ha permesso di raggiungere un risultato così impegnativo. Grazie anche a Simone, che è stato una spalla in questo percorso, a Jacopo e Nancy, che hanno ampliato le mie conoscenze su argomenti a me sconosciuti, e a tutti gli altri ragazzi del laboratorio, sempre disponibili a dare una mano.

Quando mi sono iscritta al Politecnico dubitavo fortemente delle mie capacità. Durante i primi anni, ad ogni sessione venivo assalita dai dubbi. Ho scelto la cosa giusta? Sarò all'altezza di questo percorso? Riuscirò a portarlo a termine? I fallimenti, i pianti, la paura, sono stati controbilanciati da tutte le gioie e dagli insegnamenti che questi anni mi hanno regalato. Con la specializzazione, ho avuto modo di scoprire gli aspetti del mio ambito di studio che più mi appassionano e sui quali voglio continuare ad investigare. Ciò che però è stato determinante al fine di raggiungere i miei obiettivi, è stato riconoscere che la vita professionale e la vita privata hanno pari valore. In una società che ci spinge alla perfezione ed alla competizione continua, alimentare i legami interpersonali ed i propri interessi al di fuori dell'università è essenziale per poter raggiungere una serenità emotiva, che di riflesso gioverà anche al lavoro. La famiglia, gli amici, gli amori e le passioni sono stati motivo di turbamento, ma sono anche ciò che ogni giorno mi ha ricordato quanto sia piacevole esplorare il mondo che mi circonda, e prendermene cura.

Vorrei ringraziare la mia famiglia, che mi ha cresciuta in un ambiente orientato alla curiosità entusiasta della vita. Un grazie a mia madre, per essere stata il mio modello e la mia insostituibile roccia, e per avermi insegnato che per rendere possibile qualcosa, basta provare a farla. Grazie a mio padre, che mi ha sempre spinta a pormi le giuste domande su quale tipo di persona io voglia essere, e

che mi ha ricordato che Silvia Fois non è, e non sarà mai, solo una laurea in ingegneria. Ringrazio mio fratello Nicolò, che spero veda in me un esempio di cui essere fiero, tanto quanto lui lo è per me, con la sua costante ludicità trasfigurata in esilarante cinismo. Un grazie a Talello, che mi ha scelta come parte della sua vita senza che nessuno glielo richiedesse, e che mi ha spiegato come il rispetto vada conquistato assumendosi le proprie responsabilità, condendo il tutto con qualche balletto. Ringrazio Franca, che nonostante abbia dovuto accarezzarmi l'ala all'infinito e subire piedi puzzolati, è restata comunque, sino alla fine, una delle mie migliori fan. Ringrazio Luisa, che ha saputo rinforzare un collante apparentemente invisibile, l'amore, e che mi è stata amica sincera.

Un enorme grazie alla mia compagna d'avventura e ormai sorella, Giulia. Mi ha mostrato che l'amicizia è comprensione, è risate, è darsi una mano e per errore nascondere l'altra, ma poi pentirsene. E' riconoscere i propri limiti e cercare di superarli, per se stessi, ma anche per l'altro.

Grazie agli amici con cui sono cresciuta. Ringrazio Viky, perchè è la dimostrazione che un sorriso e un abbraccio sono sempre la soluzione giusta, anche se spesso me ne dimentico. Grazie a Beatrice, per essere stata un esempio di come si debba guardare in faccia la paura per mandarla a quel paese, ed essere finalmente quel che si vuole essere. Dico grazie a Saretta, a cui i fronzoli piacciono solo da indossare, perché quello che mi da e quello che vede in me è l'unica cosa importante, la sostanza.

Ringrazio anche Fiorenza, che ha saltato tutte le tappe, ed è riuscita a diventare per me come un'amica d'infanzia. La vera donna, permettetemi la licenza vagamente discriminatoria, ha la sua tenacia alternata ai piantini di fronte a qualsiasi film.

Infine ringrazio Nicola, che crede in me come io non ho mai fatto. Grazie perché mi dimostra costantemente come, anche a venticinque anni, con la persona giusta, si possa piacevolmente regredire a quando se ne avevano cinque. Inoltre, con lui ho capito che la via del dialogo è realizzabile, ed è bellezza.

Torino può far paura all'inizio, quando arrivi con il tuo bagaglietto da una piccola città e non conosci nessuno, ma poi scopri che ogni persona è un'occasione, per liberarti della paura del giudizio, e per trovare un amico che ti apprezzi per quello che sei. Grazie ai miei spinazzolesi preferiti: Roberto, Nicola, Sebastiano, Erasmo e Yuto. Con voi finalmente non ho mai bisogno di nascondermi.

Grazie perché mi avete fatto bene al cuore.

Indice

Elenco delle tabelle x

Elenco delle figure xi

Glossario xiv

1 Introduzione 2
- 1.1 Background Medico . . . 2
  - 1.1.1 Elettrocardiogramma . . . 2
  - 1.1.2 Pulsossimetria e Fotopletismografia . . . 6
  - 1.1.3 Pressione Arteriosa e Sfigmomanometro . . . 7
- 1.2 Dispositivi Wearable . . . 8
  - 1.2.1 Requisiti . . . 9
  - 1.2.2 Obiettivi del sistema PulsEcg . . . 9

2 Hardware 12
- 2.1 ECG Watch . . . 12
- 2.2 PulsEcg . . . 14

3 Rete Neurale 18
- 3.1 Fondamenti di Deep Learning . . . 18
  - 3.1.1 Intelligenza Artificiale e Machine Learning . . . 18
  - 3.1.2 Introduzione alle Reti Neurali . . . 20
  - 3.1.3 Reti Neurali Convolutive . . . 21
- 3.2 Rete Neurale del PulsEcg . . . 22
  - 3.2.1 Dataset e Preprocessing . . . 22
  - 3.2.2 Struttura della rete . . . 23

4 Specifiche e Design 27
- 4.1 Applicazione Android ECG Watch . . . 28
- 4.2 Fasi preliminari all'implementazione . . . 30

4.2.1 Definizione dei Requirements 32
4.2.2 Design proposto per l'applicazione 38

5 Ambiente iOS e Librerie 45
5.1 Swift vs Objective-C 45
5.2 Elementi base di un programma Swift 46
5.2.1 UIViewController 47
5.2.2 Segue 48
5.2.3 Interface Builder 49
5.3 NotificationCenter 49
5.4 Libreria CoreBluetooth 51
5.4.1 Bluetooth Low Energy 51
5.4.2 Il servizio ECG 53
5.4.3 Le funzioni di CoreBluetooth 53
5.5 Libreria CoreLocation 54
5.6 Libreria Charts 56
5.7 API per Import-Export 58
5.7.1 Activity View Controller e PDFKit 60
5.7.2 Document Picker View Controller 62
5.7.3 MessageUI 63
5.8 Libreria TensorFlowLite 64

6 Front-End 68
6.1 User Experience 68
6.1.1 Profilo Utente 69
6.1.2 Design dell'interfaccia 70
6.1.3 Logo e icone 72
6.2 Navigazione 73
6.3 Schermata principale e impostazioni 77
6.3.1 HomePage VC 77
6.3.2 GeneralOptions VC 82
6.4 Connessione al bracciale e acquisizione 83
6.4.1 ConnectBluetoothDevice VC 84
6.4.2 AquireECG VC 87
6.5 Archivio 92
6.6 Visualizzazione della misurazione 94
6.6.1 GeneralShowECG VC 95
6.6.2 ShowECG VC 100

7 Back-End 106
7.1 Classe BleManager 106
7.2 Elaborazione dei segnali 112
7.2.1 Elaborazione ECG 113
7.2.2 Elaborazione PPG 119
7.3 Calcolo di HeartBeat e SpO2 120
7.3.1 HB e Fibrillazione Atriale 121
7.3.2 SpO2 e segnalazione delle anomalie 125
7.4 Derivazione della pressione 128
7.4.1 Data Preprocessing 128
7.4.2 Elaborazione dell'output e ipertensione 130

8 Conclusione 135

Bibliografia 137

Elenco delle tabelle

1.1 Corrispondenza tra tracciato elettrocardiografico e attività cardiaca 3

4.1 Stakeholders coinvolti nel sistema 32
4.2 Interfacce necessarie tra attori e sistema 33
4.3 Requirements funzionali dell'app PulsEcg 35
4.4 Requirements non funzionali dell'app PulsEcg 36
4.5 ID delle classi per la lettura della matrice di tracciabilità 4.6. Gli ID sono scelti per semplice abbreviazione del nome della classe. Le classi seguite dall'acronimo VC sono quelle dotate di interfaccia grafica. 42
4.6 Matrice di Tracciabilità dei FR. Le classi coinvolte nel soddisfacimento del FR riportato sulla riga della tabella sono contrassegnate con una X. Gli ID dei FR sono consultabili in Tab. 4.3, quelli delle classi in Tab. 4.5. 43

7.1 Coefficienti dei cinque blocchi biquadratici costituenti il filtro Notch IIR 116

Elenco delle figure

1.1 Tipica forma d'onda dell'ECG misurata dalla II derivazione[3] . . . 3
1.2 Triangolo di Einthoven . . . 4
1.3 Carta millimetrata da ECG . . . 5
1.4 ECG con fibrillazione atriale vs ECG normale [4] . . . 5
1.5 Principio di fotopletismografia (PPG) [7]: (a) riflettente; (b) trasmissiva; (c) esempio di segnale PPG. . . . 6
1.6 Sfigmomanometro . . . 8

2.1 Fronte del bracciale ECG Watch . . . 13
2.2 Retro del bracciale ECG Watch . . . 13
2.3 Circuito PulsEcg . . . 15
2.4 Circuite Block Diagram del PulsEcg . . . 16

3.1 Rapporto tra AI, ML e DL . . . 19
3.2 Rete feedforward . . . 20
3.3 Rappresentazione dei calcoli che avvengono in ogni nodo nascosto . 21
3.4 Percentuali di soggetti con RMSE inferiore a 10 per ciascun segnale in ingresso e per ciascuna rete, nel caso di inizializzazione random e test di generalizzazione della rete migliore . . . 24
3.5 Grafico ResNet visualizzato con Tensorboard: grafico ResNet + LSTMl (a), blocco resnet (b). Ogni ResNet è composta da 4 blocchi e quindi a seconda dell'architettura (ResNet o ResNet + LSTM) può essere seguito da strati LSTM . . . 25

4.1 Schermate principali dell'applicazione ECG Watch . . . 28
4.2 Schermate di visualizzazione delle misurazioni dell'applicazione ECG Watch . . . 29
4.3 Processo di realizzazione di un sistema composto da una parte hardware ed una software . . . 31
4.4 Use Cases dell'app iOS PulsEcg pt1 . . . 37
4.5 Use Cases dell'app iOS PulsEcg pt2 . . . 38

4.6 Fase di design del processo di sviluppo di un sistema 39
4.7 Struttura del package dell'app PulsEcg 40
4.8 Class Diagram dell'app PulsEcg 41

5.1 Esempio della maggiore leggibilità di Swift rispetto a Objective-C. . 46
5.2 Stati del ciclo di vita di un VC 47
5.3 Rappresentazione di un Segue tra due VC 48
5.4 Pattern Observer 50
5.5 Architettura GATT. 52
5.6 Screenshot della schermata con il dettaglio del grafico ECG 59
5.7 Screenshot della schermata con il dettaglio del grafico PPG 59
5.8 Aspetto dell'Activity VC d'esportazione. 60
5.9 PDF generato dall'app PulsEcg, comprendente la posizione geografica 61
5.10 Aspetto del VC di composizione email generato dall'applicazione PulsEcg con condivisione della posizione. Alla mail è allegato un pdf oppure un json. 63

6.1 Corrispondenza colore-emozione[34] 71
6.2 Palette di colori scelti per la GUI dell'app PulsEcg 71
6.3 Homepage dell'applicazione PulsEcg 73
6.4 Logo e piattaforma di progettazione delle icone dell'app PulsEcg . . 74
6.5 Anteprima dello storyboard dell'applicazione, raffigurante i View Controllers e la navigazione tra di essi 75
6.6 Schema di navigazione tra View Controllers 76
6.7 UI dei View Controller HomePage e GeneralOptions 78
6.8 Struttura delle views e segue connessi all'HomePage VC 79
6.9 Struttura delle views e segue connessi al GeneralOptions VC 82
6.10 UI dei View Controller AquireECG e ConnectBluetoothDevice . . . 84
6.11 Struttura delle views e segue connessi al ConnectBluetoothDevice VC 85
6.12 Definizione dei membri che compongono la classe Misurazione . . . 92
6.13 UI del View Controller Archivio. Nello screenshot sono presenti file appartenenti ad un'unica sezione mese-anno 93
6.14 Ricerca tra i file dell'Archivio VC 95
6.15 Struttura delle views e segue connessi al GeneralShowECG VC . . . 96
6.16 GeneralShowECG VC 97
6.17 Struttura delle views e segue connessi al ShowECG VC 101
6.18 Dettaglio sull'ECG dello ShowECG VC 102
6.19 Dettaglio sul PPG dello ShowECG VC 102
6.20 Dettaglio sulla pressione dello ShowECG VC 103

7.1 Vecchio screenshot del segnale ECG, acquisito quando l'app non era ancora munita della logica necessaria all'elaborazione dei segnali. . 113

7.2 Filtro Notch usato da PulsEcg per rimuovere il rumore dal segnale ECG 115

Glossario

AA

Attività Atriale

ABP

Pressione Arteriosa

AI

Intelligenza Artificiale

BLE

Bluetooth Low Energy

CNN

Rete Neurale Convolutiva

DBP

Pressione Sanguigna Diastolica

DL

Deep Learning

DRL

Driven Right Leg Circuit

ECG

Elettrocardiogramma

FA

Fibrillazione Atriale

FIR

Finite Impulse Response

FR

Requisito Funzionale

GATT

General Attribute Profile

GUI

Graphic User Interface

IDE

Integrated Development Environment

IIR

Infinite Impulse Response

ML

Machine Learning

NFR

Requisito Non Funzionale

PCB

Process Control Block

PPG

Fotopletismografia

RMSE

Root Mean Square Error

SBP

Pressione Sanguigna Sistolica

SpO2

Saturazione emoglobinica arteriosa

SV

Stack View

UDT

User Defined Type

UX

User Experience

VC

View Controller

Capitolo 1

Introduzione

I disturbi cardiaci, al giorno d'oggi, coprono una grossa percentuale dei problemi di salute nel mondo. Perciò viene loro dedicato un grande sforzo sotto il profilo dello studio, della prevenzione e del trattamento. Per rilevare le anomalie cardiache, i medici tengono sotto controllo alcuni parametri vitali dei pazienti, tra cui il segnale ECG, la percentuale di SpO2 nel sangue ed il valore della pressione arteriosa.

PulsEcg è un progetto nato dalla necessità di offrire supporto alla sanità, rendendo queste procedure medicali di monitoraggio più rapide, precise e semplici. La presente tesi tratta dello sviluppo di una applicazione per iOS che si interfacci con il componente hardware del sistema.

Al fine di comprendere meglio le motivazioni che hanno mosso la progettazione del PulsEcg e le tecnologie impiegate per la sua realizzazione, è necessaria un'introduzione. Nei prossimi paragrafi verrà quindi fornita una panoramica su come interpretare i parametri vitali citati, su quali siano gli strumenti attualmente utilizzati per la loro acquisizione e sul perchè sia così importante semplificare queste procedure attraverso l'utilizzo di dispositivi wearable.

1.1 Background Medico

In questa sezione verranno rapidamente offerte al lettore tutte le conoscenze mediche necessarie alla compresione dei successivi capitoli.

1.1.1 Elettrocardiogramma

Il principale strumento usato per il monitoraggio della salute cardiaca è l'elettrocardiogramma (ECG), introdotto da Willem Einthoven [1].

Il cuore umano è composto nella sua quasi totalità dal miocardio, un tessuto muscolare complesso la cui contrazione permette di pompare sangue nel corpo.

Nello specifico, la contrazione delle fibre muscolari è dovuta all'attività bioelettrica spontanea di alcune cellule cardiache. Questa attività è ciclica e può essere rilevata come una variazione di potenziale elettrico, ovvero un segnale con frequenze che vanno da 0,01 Hz a 150 Hz e con ampiezza di 1 mV [2]. Tale segnale si diffonde in tutto il corpo, tanto da poter essere acquisito in diversi punti della sua superficie con degli elettrodi. Le informazioni ottenute dagli elettrodi vengono inviate ad uno strumento chiamato **elettrocardiografo**, per tracciare l'**onda complessa caratteristica PQRST** (Fig. 1.1), scomponibile nelle onde riportate in Tab. 1.1. Ciascuna delle sotto-onde ha corrispondenza diretta con un certo movimento del cuore. Qualunque loro deformazione rispetto all'andamento atteso può indicare una disfunzione/anomalia cardiaca.

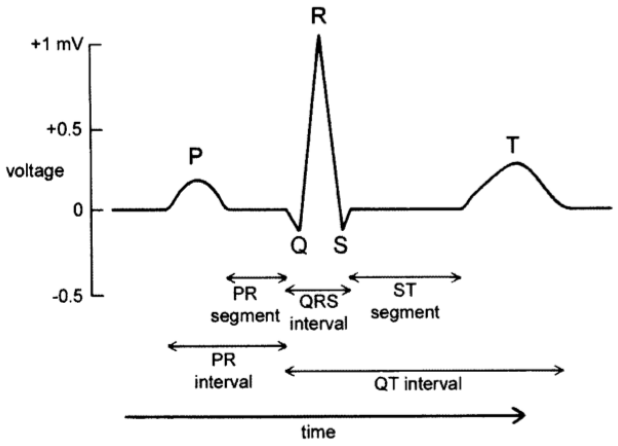


Figura 1.1: Tipica forma d'onda dell'ECG misurata dalla II derivazione[3]

Onda/Segmento	**Descrizione**	**Durata[s]**	**Ampiezza[mV]**
P	Depolarizzazione atriale	0.05-0.12	0.2-0.4
PR	Tempo di propagazione dell'onda di depolarizzazione dal nodo seno atriale	0.16-0.2	
ST	Tempo di ripolarizzazione ventricolare	0.27 – 0.33	
QT	Tempo di depolarizzazione e ripolarizzazione ventricolare	0.35 – 0.42	
QRS	Tempo di depolarizzazione ventricolare	0.08-0.12	1.0-2.0
[HTML]EFEFEF			

Tabella 1.1: Corrispondenza tra tracciato elettrocardiografico e attività cardiaca

L'attività elettrica generata dal cuore può essere vista come un vettore su un piano tridimensionale, osservabile per proiezione da più angolazioni. Ogni punto di osservazione, detto **derivazione**, fornisce una diversa versione dello stesso tracciato. Distinguiamo tre possibili configurazioni, per un totale di dodici derivazioni: bipolari, unipolari e precordiali. La più semplice sfrutta le derivazioni bipolari,

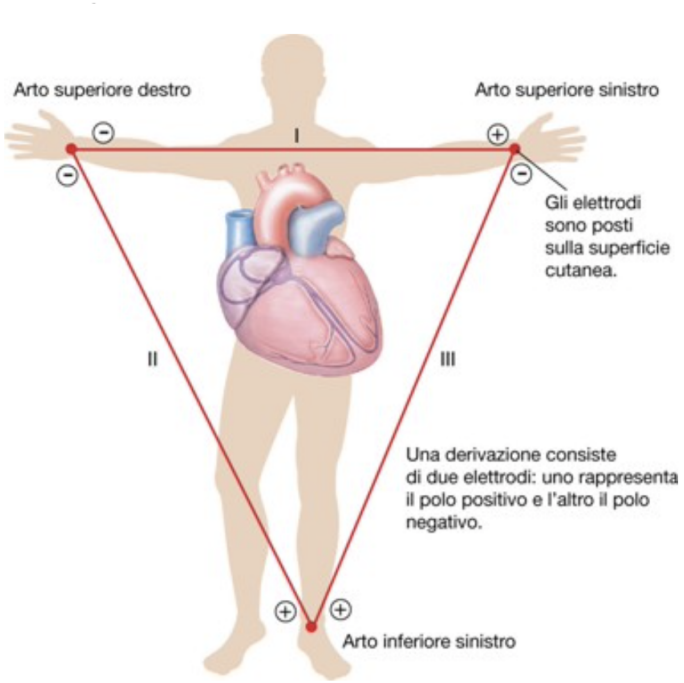


Figura 1.2: Triangolo di Einthoven

ponendo degli elettrodi di superficie ai vertici del **triangolo di Einthoven** (Fig. 1.2), un triangolo equilatero invertito incentrato sul cuore. Gli elettrodi registrano l'attività cardiaca proiettata su un lato del triangolo, perciò ogni derivazione può essere calcolata come somma/differenza dei potenziali acquisiti su polso destro (PD), polso sinistro (PS) e caviglie (C) [4].

$$Lead_I = PS - PD \tag{1.1a}$$
$$Lead_2 = C - PD \tag{1.1b}$$
$$Lead_3 = C - PS \tag{1.1c}$$

In base alla derivazione d'acquisizione, il segnale elettrocardiografico evidenzia aspetti diversi dell'attività cardiaca. Un ECG rilevato da una derivazione bipolare permette di rilevare superficialmente alcune anomalie, in presenza delle quali sarebbe necessaria una analisi più approfondita, sfruttando altre derivazioni.

Il segnale elettrocardiografico è tracciato su una particolare carta millimetrata (Fig. 1.3) costituita da quadratini, di lato 1mm, raggruppati in insiemi da 5x5mm. Ogni quadratino di 1 mm corrisponde sulle ascisse a 0.04 secondi, perciò un secondo sarà rappresentato da 5 quadrati grandi. Sulle ordinate, un quadratino indica 0.1 mV.

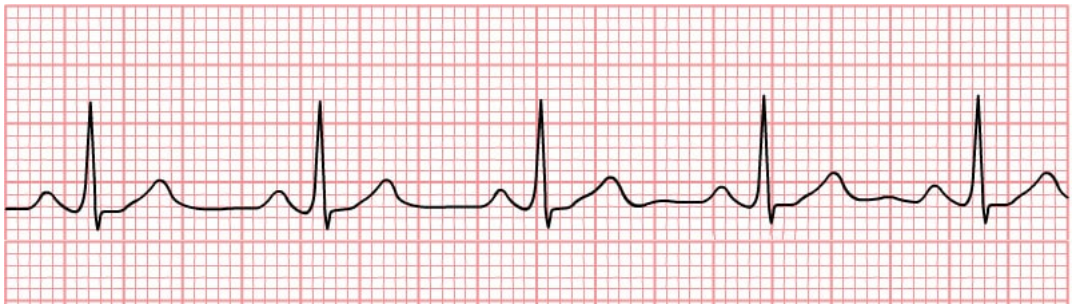


Figura 1.3: Carta millimetrata da ECG

Fibrillazione atriale

Una disfunzione cardiaca che può essere rilevata mediante un'attenta osservazione dell'andamento del segnale elettrocardiografico è la **fibrillazione atriale**. La fibrillazione atriale (FA) è l'aritmia più comune negli adulti. Consiste in degli impulsi disorganizzati generati intorno alle vene polmonari che portano alla deformazione dell'onda P di depolarizzazione atriale. La zona dell'elettrocardiogramma che precede il complesso QRS è, in questo caso, caratterizzata da numerose rapide onde di attività atriale (AA). [5].

Figura 1.4: ECG con fibrillazione atriale vs ECG normale [4]

La Fibrillazione Atriale rappresenta una delle patologie cardiache più pericolose e difficili da rilevare. Questa condizione può rimanere silente, cioè senza mostrare alcun sintomo, per anni e non essere rilevata nemmeno da strumenti professionali. In effetti, all'inizio si hanno solo alcune percosse anormali, che però diventano più frequenti nel tempo. Occasionalmente ci possono essere sintomi come palpitazioni cardiache, svenimenti, vertigini, mancanza di respiro o dolore toracico. Ovviamente, un cuore che batte in modo così irregolare mette il paziente a rischio di insufficienza cardiaca, demenza e ictus [6]. Queste considerazioni sottolineano l'importanza di sviluppare un sistema che sia in grado di rilevare la fibrillazione atriale in maniera efficiente.

1.1.2 Pulsossimetria e Fotopletismografia

Il livello di ossigeno nel sangue, rilevato attraverso la misurazione della sua saturazione (SpO2), è un indicatore ampiamente utilizzato per la diagnosi clinica del benessere del paziente, in quanto l'ossigeno ha un ruolo fondamentale per la corretta funzionalità degli organi e dei tessuti vitali. Se il corpo umano non è in grado di scambiare e fornire ossigeno in modo efficiente, le singole cellule, il cuore ed il cervello possono riportare gravi danni a lungo termine. Pertanto, misurare accuratamente la SpO2 ad alta frequenza è fondamentale per fornire segnali di allarme in presenza di anomalie o potenziali problemi di salute.

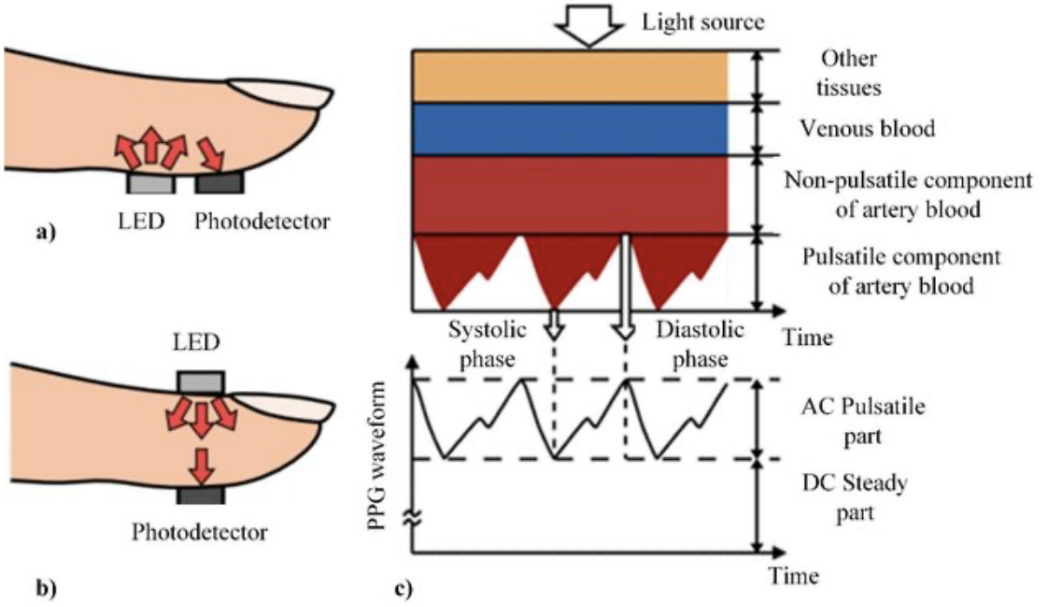


Figura 1.5: Principio di fotopletismografia (PPG) [7]: (a) riflettente; (b) trasmissiva; (c) esempio di segnale PPG.

La **pulsossimetria** è una misura comune della SpO2 per ambienti ospedalieri e domiciliari. Un tipico sistema pulsossimetrico genera dei fasci di luce a lunghezze d'onda specifiche per poter attraversare in profondità la punta di un dito, il lobo dell'orecchio o altre zone alle estremità del corpo del paziente. Dopo aver attraversato strati di pelle, arterie e cellule del sangue, le luci colpiscono dei fotoelettrodi dedicati. In base all'intensità di tali luci è possibile quindi stimare il livello di SpO2 periferica [8].

L'SpO2 deriva dal rapporto tra la quantità di emoglobina ossigenata e la quantità totale di emoglobina presente nel sangue. A causa della differenza nello spettro di assorbimento dell'emoglobina ossigenata e deossigenata, un pulsossimetro è dotato

di una coppia di diodi emettitori di luce rossa (con lunghezze d'onda intorno a 660 nm) e infrarossa (940 nm), e da un fotorilevatore [9].

Se fotorilevatore e led sono posti sulla stessa faccia del saturimetro, si tratta di pulsossimetria riflettente (Fig. 1.5.a). Parliamo invece di pulsossimetria trasmissiva se si trovano ai lati opposti del dispositivo (Fig. 1.5.b). Nel primo caso il segnale riflesso può essere monitorato tramite la **fotopletismografia (PPG)**, che rappresenta la variazione di assorbimento della luce. La trasmissione della luce attraverso i tessuti infatti diminuisce durante la sistole a causa dell'aumento del volume del sangue arterioso e aumenta durante la diastole [10] (Fig. 1.5.c). L'onda pletismografica si modifica in presenza di anomalie cardiache, perciò permette, insieme all'ECG, di valutare più nel dettaglio la salute del cuore.

Le componenti DC e AC dei due led hanno diverse ampiezze, perciò se ne calcola il rapporto (Eq. 1.2). L'SpO2 è quindi ricavata con un'approssimazione lineare derivata da una best-fit straight-line dei dati SpO2 rispetto a R (Eq. 1.3) [11].

$$R_{Red} = \frac{AC_{Red}}{DC_{Red}} \qquad R_{IR} = \frac{AC_{IR}}{DC_{IR}} \tag{1.2a}$$

$$R_{Tot} = \frac{R_{Red}}{R_{IR}} \tag{1.2b}$$

$$SpO2 = 104 - R_{Tot} * 17 \tag{1.3}$$

1.1.3 Pressione Arteriosa e Sfigmomanometro

La **pressione sanguigna** è la forza esercitata dal sangue contro le pareti dei vasi sanguigni quando viene pompato dal cuore. La pressione dipende della resistenza arteriosa, dal diametro della luce arteriosa e dal volume sanguigno di espulsione, chiamato anche gittata cardiaca. I valori ottenuti con una misurazione della pressione subiscono tuttavia l'influenza di altri fattori, come ad esempio l'età, fattori genetici, sesso, eccesso di peso, obesità, stile di vita sedentario, ingestione di alcol, tabacco e dieta [12]. Quando la pressione sanguigna è troppo bassa, ci troviamo di fronte ad una condizione di **ipotensione**. Se invece la pressione è costantemente alta, si tratta di **ipertensione**, la quale può essere particolarmente problematica perchè porta ad un forte aumento delle malattie cardiovascolari.

Lo strumento tradizionale per la misurazione della pressione sanguigna è lo **sfigmomanometro**. I valori sono misurati in millimetri di mercurio (mmHg). La cifra più alta rilevata è detta **pressione sistolica (SBP)**, e indica la pressione raggiunta in fase di contrazione del miocardio. Il valore più basso rappresenta invece la **pressione diastolica (DBP)**, verificata quando quando il miocardio si rilassa [12].

Figura 1.6: Sfigmomanometro

Lo sfigmomanometro sfrutta l'occlusione dell'arteria brachiale, realizzata per pressione da parte di un bracciale gonfiabile. Il bracciale viene gonfiato attraverso una pompa sino a quando la pressione esterna non supera la SBP, così da interrompere il flusso sanguigno. Quindi, la pressione all'interno del bracciale viene ridotta tramite una valvola fino al ripristino del flusso. Durante queste fasi, un sistema di misurazione connesso al bracciale mostra i valori di pressione del sangue.

Sebbene sia tuttora il più diffuso, questo metodo presenta alcuni **grossi svantaggi** che ne limitano l'uso in determinati contesti clinici o domestici:

1. non è adatto ai neonati;
2. il paziente può essere disturbato dal gonfiaggio della cuffia e ciò può causare un improvviso aumento della pressione arteriosa;
3. non è utilizzabile se la pelle del paziente è danneggiata;
4. deve essere utilizzato con attenzione, perchè potrebbe lesionare i nervi;
5. non è possibile misurare una PA continua perché è necessaria una pausa di almeno 1-2 minuti prima di poter ripetere il procedimento. Una misurazione continua della pressione sanguigna è particolarmente importante in una moltitudine di contesti clinici [13].

1.2 Dispositivi Wearable

La progettazione e lo sviluppo di **dispositivi indossabili** (wearable), per il monitoraggio dei parametri vitali, sta negli anni guadagnando sempre più interesse da parte della comunità scientifica, in quanto permette di controllare lo stato di salute

di chi soffre di patologie a rischio anche fuori dall'ospedale. Dispositivi di questo tipo sono in grado di acquisire, analizzare ed eventualmente esportare dati sulla salute di chi li indossa, durante attività quotidiane e sportive. La segnalazione di eventuali anomalie aiuta l'utente a capire quando è il momento di portare avanti un'indagine più accurata sul proprio stato di salute, mentre l'esportazione e condivisione dei dati può velocizzare lo scambio di informazioni con il proprio medico curante.

Per rispondere a questa domanda, negli ultimi anni sono stati prodotti numerosi prototipi di sistemi e prodotti commerciali, che mirano a fornire informazioni di feedback in tempo reale sui valori di pressione sanguigna, temperatura corporea e cutanea, saturazione di ossigeno, frequenza respiratoria , elettrocardiogramma, ecc. Le misurazioni ottenute vengono solitamente comunicate tramite un collegamento wireless o cablato ad un nodo centrale, ad esempio un PDA (Personal Digital Assistant) o una scheda a microcontrollore, che può a sua volta visualizzare le relative informazioni su un'interfaccia utente, o trasmettere i segni vitali a un centro medico [14].

1.2.1 Requisiti

I sistemi wearable di monitoraggio dei parametri vitali devono soddisfare determinati criteri medici rigorosi e contemporaneamente tener conto di vincoli ergonomici e hardware. La progettazione di un sistema di monitoraggio deve assolvere diversi requisiti di vestibilità, perciò il peso e la dimensione devono essere contenuti, così da non ostacolare i movimenti dell'utilizzatore. Inoltre, è necessario considerare le possibili preoccupazioni degli utenti relative alle radiazioni, all'estetica, alla privacy dei dati medici personali raccolti e al consumo di energia. Per finire, anche il prezzo del sistema deve essere contenuto, così da garantirne l'accesso ad un ampio pubblico [14].

1.2.2 Obiettivi del sistema PulsEcg

Per quanto riguarda l'acquisizione del segnale ECG e il valore della saturazione dell'ossigeno nel sangue, il mercato offre attualmente diverse alternative di dispositivi piccoli e semplici da usare, eventualmente wearable. Nell'ambito della misurazione della pressione arteriosa invece, lo sfigmomanometro continua ad essere il dispositivo maggiormente usato. Nel capitolo 1.1.3 abbiamo però evidenziato i numerosi limiti di questo strumento, sia per quanto riguarda l'impossibilità di acquisire valori con continuità, sia per la scomodità e complessità d'utilizzo. Questo è il motivo per cui è interessante esplorare nuove possibilità che semplifichino e velocizzino questo procedimento. Il sistema PulsEcg propone una soluzione basata

sull'intelligenza artificiale.

Il sistema PulsEcg che verrà analizzato nei prossimi capitoli è un sistema composto da un dispositivo hardware di piccole dimensioni, avente le sembianze di un braccialetto, e da un'applicazione per cellulare che sia in grado di interfacciarsi con esso, sia in ambiente Android che iOS.

Mediante degli elettrodi ed un sensore, il bracciale PulsEcg permette di acquisire i segnali elettrocardiografico e fotopletismografico, e di inviarli all'applicazione mediante Bluetooth.

L'applicazione PulsEcg è in grado di elaborare i segnali, di visualizzarli, di analizzarli (allo scopo di ricavare HB, FA e SpO2), di archiviarli ed eventualmente di permetterne la condivisione. Inoltre, è capace di ricavare i valori di pressione sistolica e diastolica in maniera non invasiva, rapida ed estremamente semplice, grazie ad un algoritmo di intelligenza artificiale che riceve in input i segnali precedentemente preprocessati. La misurazione della pressione diventa così una procedura estremamente rapida, comoda, adatta a tutte le età ed accessibile ad un basso costo. Lo scopo di questa tesi è la progettazione e lo sviluppo dell'applicazione PulsEcg in ambiente iOS.

Capitolo 2

Hardware

L'applicazione iOS oggetto di questa tesi è stata sviluppata allo scopo di interfacciarsi con il **PulsEcg**, un dispositivo hardware di cui circuito e firmware sono stati progettati ed implementati da un precedente tesista. Il PulsEcg è dotato di elettrodi e sensori in grado di misurare i **segnali PPG ed ECG** e di inviarli poi, mediante Bluetooth Low Energy, ad un applicativo che possa elaborarli e visualizzarli.

Questo dispositivo rappresenta la versione aggiornata del braccialetto ECG Watch. Il sistema **ECG Watch**, risalente al 2015, comprende sia una semplice applicazione Android che un primo dispositivo in grado di acquisire il solo segnale ECG e di inviarlo all'applicazione sfruttando il protocollo Bluetooth 2.0. Il sistema ECG Watch è stato quindi il punto di partenza, sia per il bracciale PulsEcg che per le applicazioni con cui si interfaccia.

2.1 ECG Watch

L'ECG Watch è un sistema di acquisizione wireless basato su microcontrollore, che consente di registrare un segnale ECG semplicemente toccando un elettrodo con il dito. I dati acquisiti vengono inviati all'interfaccia utente, disponibile sia in ambiente Android che Windows. Il dispositivo è stato progettato prestando particolare attenzione a garantire delle dimensioni compatte ed un consumo energetico molto contenuto. Esso è integrato in un **braccialetto**, protetto da una custodia realizzata mediante stampa 3D. Le applicazioni mobili e desktop consentono di avviare acquisizioni, archiviarle, inviarle al server medico ed eseguire algoritmi di identificazione [15].

L'intero sistema è composto da quattro parti:

1. **Braccialetto ECG Watch**, ovvero il cuore hardware del sistema, costituito da un dispositivo alimentato a batteria al litio basato sul microcontrollore

MSP430G2955 che campiona i segnali analogici e gestisce la comunicazione Bluetooth 2;

2. **ECG Watch App** (di cui si parlerà meglio nel paragrafo 4.1) è il software Android sviluppato che, grazie ad un design semplice e accattivante, permette di avviare l'acquisizione, memorizzare i dati nello smartphone e inviarli al medico;
3. **Desktop UI** è il software Windows che, oltre a quanto già possibile nell'applicazione mobile, consente anche un accurato filtraggio e de-noising del segnale e di eseguire l'algoritmo di identificazione;
4. **Identification Algorithm** è il sistema di classificazione che consente l'identificazione biometrica dell'utente.

Figura 2.1: Fronte del bracciale ECG Watch

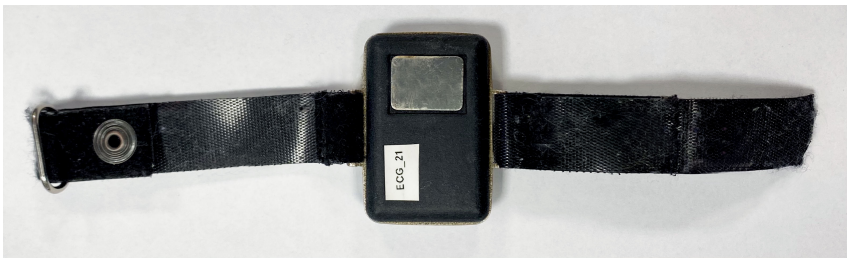


Figura 2.2: Retro del bracciale ECG Watch

Il componente del sistema preso in esame per l'aggiornamento a PulsEcg è stato il braccialetto. All'interno del dispositivo è possibile identificare diverse sezioni:

1. **Sezione di Alimentazione:** è composta principalmente da una batteria ai polimeri di litio e da un circuito micro-USB di ricarica. La tensione proveniente dalla batteria viene stabilizzata e alimenta separatamente sia la sezione analogica che quella digitale, in modo da evitare la propagazione dei disturbi.

2. **Sezione Analogica:** è costituita da diversi stadi. Il primo stadio è un filtro passa-alto utilizzato per annullare l'effetto batteria degli elettrodi e il potenziale DC del corpo umano. Nei casi in cui, per un'avaria del circuito, viene applicata tutta la tensione agli elettrodi, questa parte garantisce che la corrente che attraversa l'utente sia inferiore al livello di corrente soglia di sicurezza. Il secondo step consiste in un'amplificazione differenziale per ridurre il rumore causato dall'interferenza principale (50Hz). Seguono poi un filtro passa banda di circa [10-170 Hz] e uno stadio di amplificazione 10. Il sistema di feedback è un Driven Right Leg Circuit o "DRL". È un circuito elettrico che viene spesso aggiunto agli amplificatori di segnali biologici, per ridurre le interferenze di modo comune. Questo perché l'ECG misura segnali elettrici molto piccoli che potrebbero essere oscurati da interferenze elettromagnetiche generate dal corpo che funge da antenna.

3. **Sezione Digitale:** è un sistema basato su microcontrollore Texas Instruments MSP430F2955 adibito a campionare il segnale ECG di uscita della sezione analogica. Il segnale viene memorizzato in memoria flash e inviato tramite bluetooth, quando richiesto. Il circuito è stato progettato in modo da porre i componenti su entrambe le sue facce, allo scopo di contenere le dimensioni, e con tre piani di massa separati, per ridurre il rumore.

4. **Sezione Bluetooth:** il modulo Bluetooth è collegato al microcontrollore utilizzando l'interfaccia UART. Il microcontrollore è in grado di eventualmente interrompere la connessione con tale modulo per risparmiare energia [15].

2.2 PulsEcg

Il PulsEcg garantisce miglioramenti significativi delle prestazioni dell'ECG Watch in termini di:

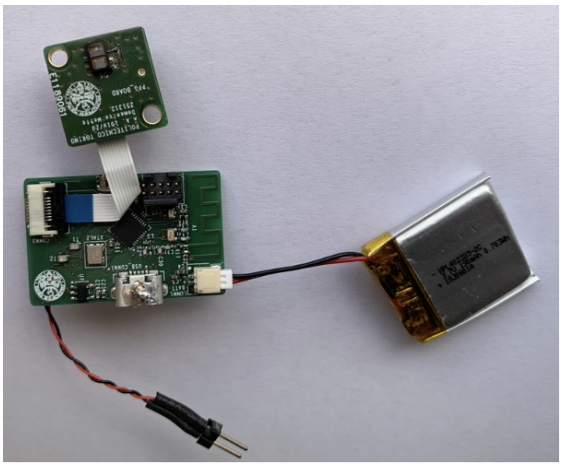
1. rumore del segnale ECG. Infatti lo spettro di potenza del segnale acquisito dall'ECG Watch presenta disturbi evidenti causati dagli alimentatori che circondano il dispositivo;

2. nuove funzionalità, ovvero il sensore per l'acquisizione del segnale pletismografico;

3. durata della batteria e qualità della comunicaizone, scegliendo un protocollo Bluetooth Low Energy al posto del Bluetooth 2[16].

Il circuito del PulsEcg, schematizzato in Fig. 2.4, è composto da due PCB (Process Control Block), uno dedicato all'ECG e all'alimentazione, uno al PPG.

(a) Circuito PulsEcg con elettrodi



(b) Circuito PulsEcg privo di elettrodi

Figura 2.3: Circuito PulsEcg

All'interno dell'ECG PBC è gestita l'alimentazione, i filtri (Passa Alto, Passa Basso e Twin-T Notch) ed il microcontrollore. Il microcontrollore controlla gli altri blocchi, raccoglie i dati dell'ECG, monitora la batteria e ottiene i valori del PPG attraverso una comunicazione I2C. L'interazione tra questo PCB e il mondo esterno è realizzata mediante diversi connettori: il connettore degli elettrodi, il connettore USB, il connettore piatto e il connettore della batteria.

Il PPG PBC è composto da un connettore piatto e da un sensore PPG (MAXM86161). Il flat cable permette di collegare il PCB PPG al PCB ECG, mentre il sensore è in grado di acquisire il segnale PPG da cui ricavare il valore di SpO2 [16].

Per poter utilizzare le funzionalità di comunicazione Bluetooth, il progetto firmware non è stato creato da zero a causa dell'elevata complessità dello stack BLE, ma è stato avviato da un progetto denominato ProjectZero, al quale sono state aggiunte la gestione del Sensor Controller e parte della comunicazione con il sensore MAXM86161. Per permettere alle applicazioni di comunicare con il dispositivo, è stato generato un nuovo servizio GATT ad hoc (questa parte verrà spiegata nel dettaglio in Cap. 5.4.1).

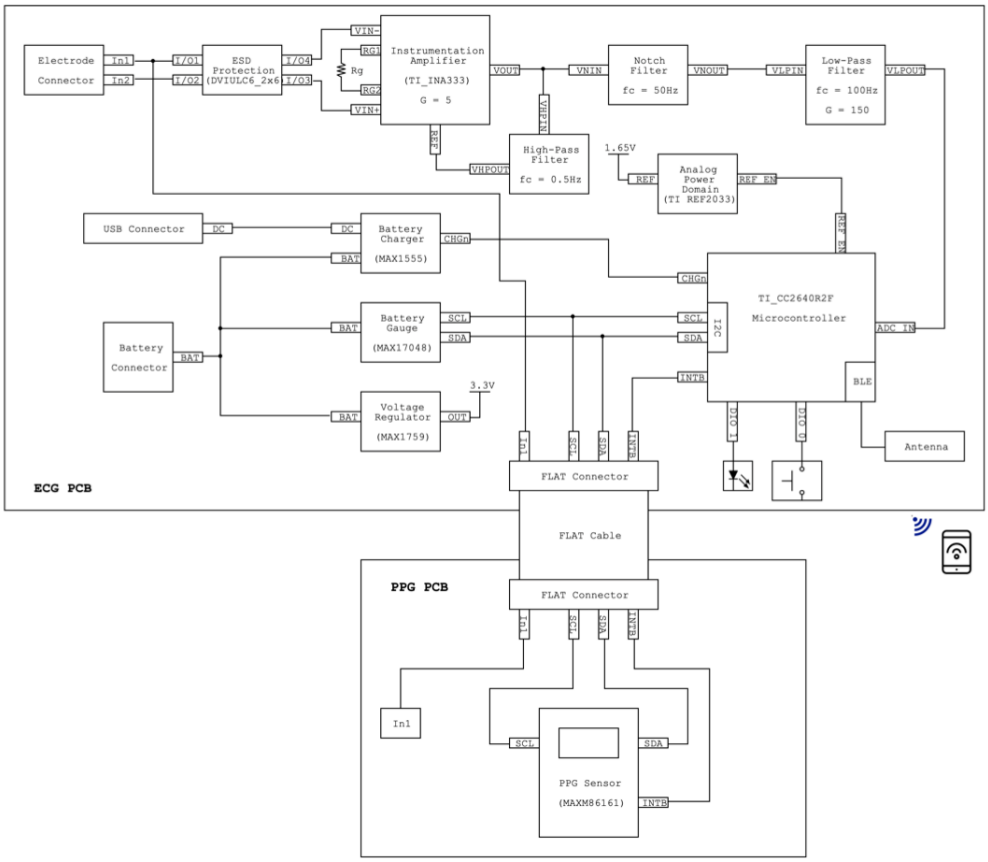


Figura 2.4: Circuite Block Diagram del PulsEcg

Capitolo 3

Rete Neurale

I segnali ECG e PPG raccolti dal dispositivo PulsEcg e successivamente filtrati dall'applicazione sono utilizzati per ricavare i valori di pressione arteriosa sistolica e diastolica in maniera non invasiva, **senza l'impiego di una cuffia**. In particolare, nell'applicazione PulsEcg è stato integrato un **algoritmo di intelligenza artificiale basato su rete neurale** progettato in un precedente studio.

Il seguente capitolo è organizzato in due sezioni. La prima parte vuole offrire un'introduzione al mondo del machine learning necessaria per la comprensione della seconda sezione, in cui viene illustrata la struttura della rete neurale che è stata inserita nell'applicazione. Il meccanismo di integrazione vera e propria in ambiente iOS verrà affrontato nel capitoli 5.8 e 7.4.

3.1 Fondamenti di Deep Learning

L'algoritmo utilizzato per la derivazione della pressione fa parte di un insieme di metodologie che possono essere raccolte nel campo del Deep Learning. Per poter comprendere di cosa si tratti però, è necessario affrontare l'argomento con un approccio dall'alto verso il basso. Prima verranno introdotti i concetti di Intelligenza Artificiale e Machine Learning, che sono più generali rispetto al Deep Learning. Poi, si passerà ad un'introduzione alle reti neurali. L'ultimo step sarà la descrizione delle reti convolutive, tipiche del Deep Learning.

3.1.1 Intelligenza Artificiale e Machine Learning

Il campo dell'Intelligenza Artificiale (AI) rappresenta l'insieme di studi accomunati dall'intento di dotare i computer di un'intelligenza simile a quella umana. Il Machine Learning è il ramo dell'intelligenza artificiale incentrato sull'apprendimento automatico delle macchine. Esso è diventato un elemento fondamentale della nostra

vita quotidiana. Chiediamo abitualmente agli smartphone dotati di ML di suggerirci i ristoranti in cui mangiare o di guidarci in un posto che non conosciamo. Le tecniche basate sul ML sono diventate essenziali in moltissimi campi della scienza e dell'ingegneria.

Un algoritmo di machine learning, studiando dei dati di training, è in grado di apprendere quale sia il ragionamento necessario per poter prendere delle decisioni in merito ad un problema, ragionamento che sarà adattabile di volta in volta in base ai nuovi dati di input. I protagonisti del machine learning sono i dati, il modello e la funzione di perdita. Queste tre componenti sono combinate per implementare il principio scientifico basato su tentativi ed errori sotto forma di algoritmo informatico. Questo principio si basa sull'adattare dinamicamente un'ipotesi in base ai dati che descrivono un fenomeno. L'ipotesi permette di fare previsioni sugli eventi futuri. I metodi ML scelgono l'ipotesi partendo da un ampio insieme di possibilità, sfruttando la discrepanza tra previsioni e dati osservati. Tale distanza è quantificata utilizzando una funzione di perdita (loss). I dati sono rappresentati come dei punti nello spazio, dotati di caratteristiche e appartenenti ad una certa classe, identificata da una label. Lo spazio dei modelli (o delle ipotesi) è un insieme di sistemi in grado di mappare un dato in una determinata classe in base alle sue caratteristiche. La funzione di loss permette di valutare la bontà di un modello, ovvero l'affidabilità delle sue predizioni [17].

I metodi del Machine Learning vengono suddivisi in due categorie: **Shallow Learning** e **Deep Learning**. Il primo sfrutta le Support Vector Machines (SVM) o le reti neurali ad un solo livello. Il Deep Learning è invece quella sottosezione del Machine Learning che si appoggia su reti neurali a più livelli nascosti [18]. Intelligenza Artificiale, Machine Learning e Deep Learning sono legati tra loro come mostrato in Fig. 3.1.

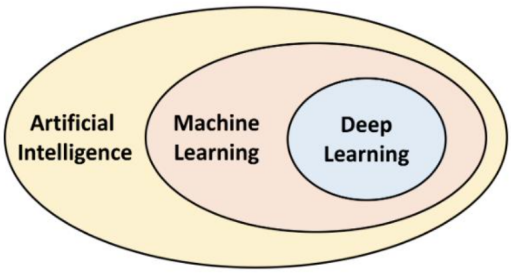


Figura 3.1: Rapporto tra AI, ML e DL

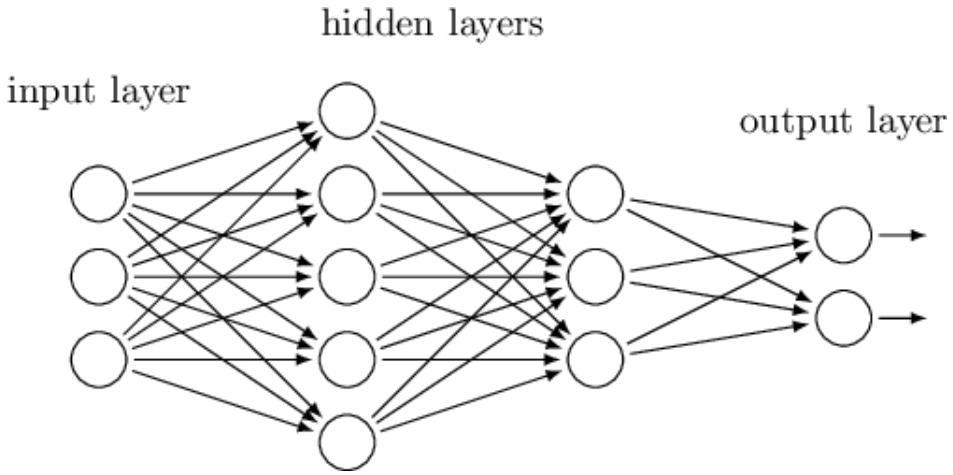


Figura 3.2: Rete feedforward

3.1.2 Introduzione alle Reti Neurali

Le **reti neurali** sono ispirate alla struttura del cervello umano, costituito da unità di elaborazione, i neuroni, e da canali di connessione, le sinapsi. Analogalmente quindi, una rete è composta da nodi intelligenti connessi l'uno con l'altro attraverso una fitta rete. La struttura base di una rete neurale è detta feedforward. Nella figura 3.2 vengono evidenziati i protagonisti della sua architettura:

- **nodi d'ingresso** che si limitano ad accogliere i dati di input. Tali dati devono essere stati preprocessati e normalizzati.
- uno o più stadi di **nodi nascosti** intelligenti. Ciascuno di questi nodi è connesso a tutti i nodi dello stadio precedente. Ogni nodo dell'ultimo stadio nascosto è connesso a tutti i nodi di output.
- **nodi di output**, in cui avviene l'assegnazione ad una classe (predizione). I nodi di uscita sono tanti quante lo sono le etichette di classe. L'uscita su ogni nodo rappresenta la probabilità che l'input appartenga a quella determinata classe.

All'interno di ogni nodo degli stadi nascosti sono memorizzate diverse informazioni: un **vettore di pesi**, un **offset** e una **funzione di attivazione**. Il vettore di pesi e l'offset avranno valori diversi da nodo a nodo, mentre la funzione è costante. Ogni nodo riceve in input un vettore di ingressi (perché riceve un elemento da ogni nodo di input), calcola il prodotto scalare tra esso e il vettore dei pesi, somma l'offset e applica la funzione d'attivazione. La funzione di attivazione ha il compito di normalizzare il risultato. Questi passaggi sono riassunti in Fig. 3.3.

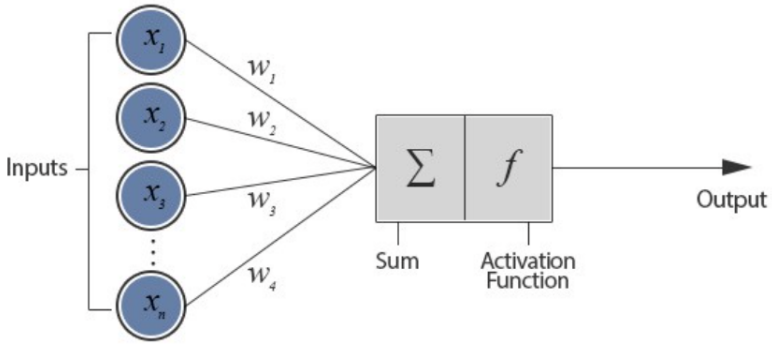


Figura 3.3: Rappresentazione dei calcoli che avvengono in ogni nodo nascosto

Quando il modello viene inizializzato, i vettori dei pesi e gli offset vengono generati randomicamente. A questo punto, la **fase di training** ha inizio. Partendo da un dataset preetichettato, uno ad uno i record vengono dati in pasto alla rete. Una volta raggiunto il layer d'output, viene calcolato l'errore con la **funzione d'errore**, ovvero la differenza tra la predizione attesa e quella ottenuta. L'errore viene quindi **propagato all'indietro** per modificare i valori dei parametri di ogni nodo, nel tentativo di correggere la rete. Questo meccanismo va perpetuato sino al verificarsi di una condizione di terminazione tra:

1. l'errore è minore di una certa soglia;
2. i pesi non variano più propagando l'errore;
3. sono state effettuate tante iterazioni (dette epoche) quanto fissato a priori.

In base al numero di neuroni impiegati e al modo in cui sono interconnessi, si possono definire diversi tipi di reti neurali. Sebbene le reti neurali siano in grado di determinare dei parametri (pesi e offsets) con l'apprendimento, altri parametri devono essere decisi a priori, mediante un **processo di tuning**. Esempi sono la dimensione dei batch (sottoinsiemi del dataset che vengono dati in input alla rete) e la funzione di loss.

Le reti neurali garantiscono un'accuratezza elevata, una gran robustezza in caso d'eccezioni ed una classificazione efficiente e rapida. I difett di questo classificatore sono la lentezza della fase di training e la difficoltà di configurazione e d'interpretabilità.

3.1.3 Reti Neurali Convolutive

Il concetto su cui si basano le **reti neurali convolutive** è che ogni neurone possa avere un **campo recettivo** diverso dagli altri. Ciascuno dei neuroni nei primi livelli

di una rete si concentra su una diversa feature, e poi combina il proprio output con quello degli altri nodi, in modo da poter passare all'analisi di caratteristiche più complesse. Queste reti sono solitamente utilizzate per il riconoscimento delle immagini, ma in alcuni casi anche per altri scopi.

Una rete neurale convolutiva può essere composta da decine di livelli. I dati sono gestiti in un formato detto **tensore** (matrice a N dimensioni). L'input viene scansionato con dei **filtri scorrevoli**, in modo da estrapolarne solo alcune caratteristiche molto specifiche. Man mano che si scende in profondità, le features vengono combinate per fare un'analisi ad un livello di astrazione sempre più alto [19].

Possiamo distinguere due fasi:

1. **Livello convolutivo :** dei filtri scorrevoli traslano sull'input e ne estraggono delle features. La funzione di attivazione modifica il risultato della convoluzione;

2. **Pooling :** riduce la dimensione spaziale del problema. Con ogni fase di convoluzione il problema viene esteso, e nella fase di pooling viene ricompresso in maniera diversa da come era inizialmente.

Lo strato convoluzionale e lo strato di pooling insieme formano l'n-esimo livello di una rete neurale convoluzionale. A seconda della complessità dei dati, il numero dei livelli può crescere per analizzare dettagli di ancor più basso livello, ma al costo di una maggiore potenza di calcolo.

I fully connected layer vengono aggiunti per l'addestramento di combinazioni non lineari delle caratteristiche d'alto livello ottenute dai livelli convoluzionali. Nella parte finale la rete diventa invece una normale feed forward con in uscita una softmax che normalizzi i risultati.

3.2 Rete Neurale del PulsEcg

Questo capitolo si focalizza sulla rete neurale convolutiva che è stata progettata per la derivazione delle pressioni misurate con il bracciale PulsEcg. La prima sezione tratta del dataset di training e del preprocessing dei suoi dati. La seconda descrive la struttura della rete.

3.2.1 Dataset e Preprocessing

La correlazione tra PPG, ECG e pressione arteriosa è stata studiata a partire dal **dataset MIMIC**. Il database è costituito da acquisizioni dei segnali fisiologici di un centinaio di pazienti in terapia intensiva, perciò rappresenta tutte quelle fisiopatologie che possono provocare improvvise variazioni della pressione sanguigna.

I segnali ECG, PPG e ABP sono campionati a 125 Hz. Inizialmente si è tenuto conto solo di PPG e ABP. Il preprocessing è avvenuto attraverso i seguenti step:

1. I valori nulli sono stati sostituiti con la copia del campione più vicino;
2. I record con linee piatte nell'ABP o nel PPG sono stati eliminati; record con più del 5% di picchi piatti in ABP e PPF sono stati eliminati;
3. Le misurazioni con PPG e ABP anomali sono state eliminate;
4. I segnali PPG sono stati filtrati con un filtro Butterworth di 4 ordine;
5. I valori outlier di PPG e ABP sono stati rimossi con un filtro Hampel;
6. Rimozione dei pazienti con meno di 3 ore di registrazione;
7. I segnali PPG sono stati standardizzati;
8. I segnali ABP sono stati normalizzati.

In seguito è stato creato un secondo set di dati, comprendente anche la V derivazione dell'ECG (la più frequente nel database MIMIC). La pipeline di preelaborazione è stata quindi estesa perchè comprendesse anche questo segnale. L'ECG è stato filtrato con un filtro Chebyshev di tipo 1 passabanda di ottavo ordine, con frequenza di taglio di 2 e 59 Hz. Solo 40 record del dataset MIMIC sono sopravvissuti al preprocessing, considerando che appena una cinquantina di pazienti possedevano tutti e tre i segnali necessari [20].

3.2.2 Struttura della rete

La ricerca della rete neurale più adatta al problema è avvenuto mediante studio di diverse possibili strutture. Le reti neurali considerate sono state analizzate ad ogni iterazione di un processo di tuning, ovvero le prestazioni sono state valutate per ogni configurazione possibile di iperparametri e confrontate con le precedenti. Anche i dati di input delle reti sono stati testati in più combinazioni, allo scopo di comprenderne la correlazione con i valori di pressione. Talvolta è stato usato il solo segnale PPG o ECG, altrimenti entrambi contemporaneamente.

La prima rete analizzata è stata la **NNOE**. La rete è stata configurata con 50 neuroni nell'hidden layer e 3 regressori. In secondo luogo, ci si è concentrati sulla **LSTM** e sulla **LSTM bidirezionale**. L'hidden layer è stato impostato a 300 unità. Per tutte e tre le reti descritte la procedura di test è stata la stessa: allenare la rete su ciascun soggetto del dataset; selezionare il modello migliore; testare tale modello su tutti gli altri soggetti. In Tabella 3.4 si riportano le percentuali

di soggetti all'interno del database che, nella derivazione di SBP e DBP, hanno ottenuto valori di Root Mean Square Error (RMSE) inferiori a 10. La tabella elenca sia le percentuali ottenute con inizializzazione randomica dei pesi, sia quelle ottenute con la rete selezionata grazie al miglior soggetto.

	Inizializzazione random			Test generalizzazione rete migliore		
	NNOE	LSTM	BLSTM	NNOE	LSTM	BLSTM
PPG	65%	74%	69%	13%	69%	74%
ECG	39%	76%	67%	11%	26%	65%
ECG + PPG	43%	82%	71%	19%	47%	69%

Figura 3.4: Percentuali di soggetti con RMSE inferiore a 10 per ciascun segnale in ingresso e per ciascuna rete, nel caso di inizializzazione random e test di generalizzazione della rete migliore

La NNOE è stata la rete che ha fornito prestazioni peggiori, soprattutto nel test di generalizzazione. Le reti LSTM e BLSTM hanno prestazioni simili sui test "random", ma la BLSTM è migliore nella generalizzazione. Dal punto di vista della validazione, usando il metodo del **Leave One Out**, nessuna delle reti ha ottenuto risultati che possono essere considerati ottimali. Questo potrebbe essere dovuto a problemi di overfitting o alla scelta di valori non ottimali degli iperparametri [21].

La rete definitiva, frutto di questi esperimenti e di alcuni studi successivi, è costituita da una **ResNet** (una particolare CNN arricchita con delle connessioni scorciatoia) opportunamente modificata e seguita da **tre strati LSTM** (ciascuno composto da 128 neuroni), di cui il primo **bidirezionale**. La ResNet modificata è costituita da 4 blocchi ResNet, come evidenziato in Fig. 3.5. Gli strati convoluzionali hanno kernel pari a 3 e step pari a 2, mentre il numero di filtri sale in ogni blocco partendo da 64 fino a 512. Ogni operazione di convoluzione è seguita dalla normalizzazione del batch.

La memoria bidirezionale a lungo termine (BLSTM) è stata scelta perchè è in grado di estrapolare caratteristiche sia in avanti che all'indietro. È un approccio di cui si comprende l'utilità riflettendo su quanto sia utilizzato anche dagli esseri umani ogni giorno: suoni, parole e persino intere frasi che all'inizio non significano nulla finiscono spesso per trovare un senso alla luce del contesto futuro.

Questa rete ha ottenuto un Mean Absolute Error (MAE) di soli 4.118 mmHg per la pressione sistolica e di 2.228 mmHg per la pressione diastolica.

La rete neurale utilizzata per la predizione della pressione nel progetto PulsEcg è stata realizzata sfruttando Tensorflow e Google Colaboratory, ed è quindi scritta in Python. [20].

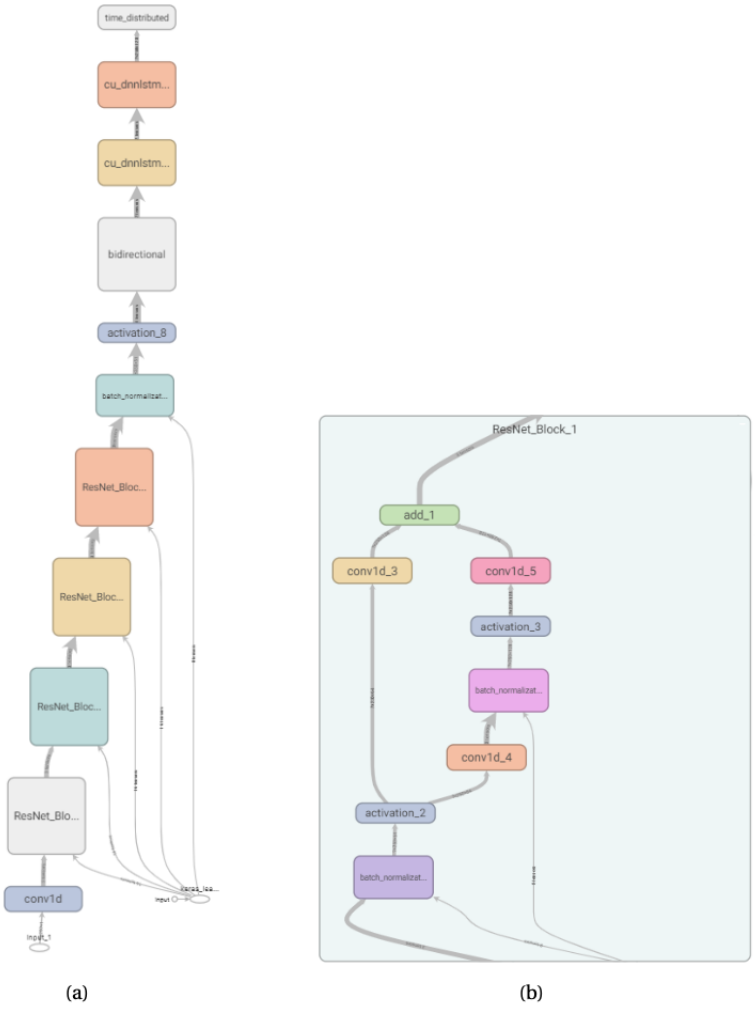


Figura 3.5: Grafico ResNet visualizzato con Tensorboard: grafico ResNet + LSTMl (a), blocco resnet (b). Ogni ResNet è composta da 4 blocchi e quindi a seconda dell'architettura (ResNet o ResNet + LSTM) può essere seguito da strati LSTM

Capitolo 4

Specifiche e Design

Questo capitolo si incentra sulla fase fondamentale dello sviluppo dell'applicazione iOS: la sua progettazione. La scrittura del codice è stata infatti preceduta da un approfondito studio del problema e delle possibili strategie risolutive, tenendo conto dello stato dell'arte.

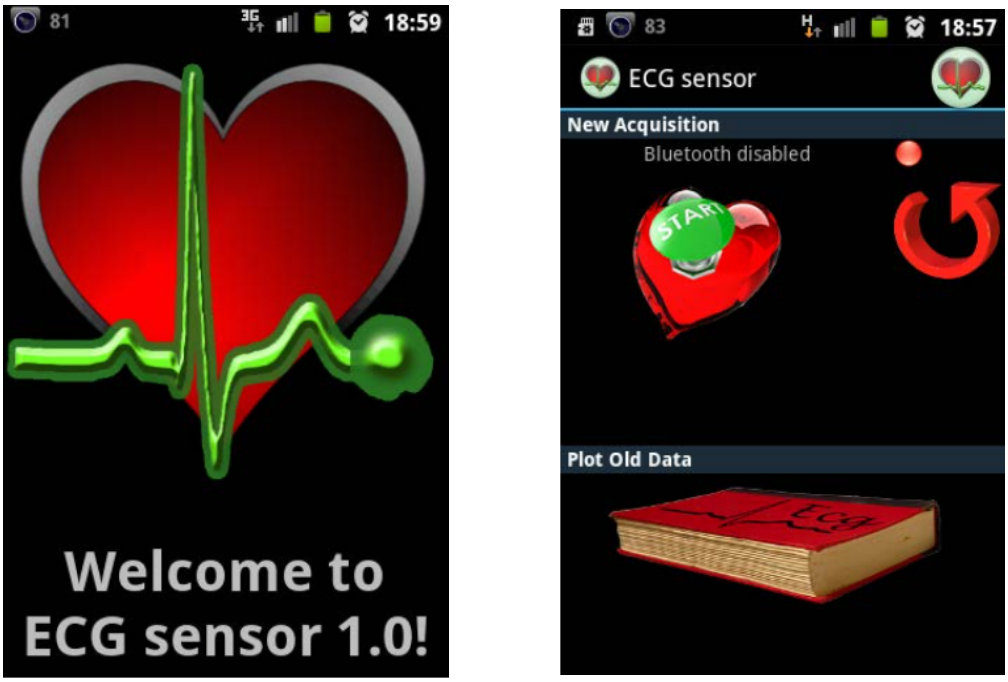
Il focus è stato posto prima di tutto sull'applicazione dell'ECG Watch. Sebbene le funzionalità di quest'app siano fortemente limitate rispetto a quelle richieste dalla PulsEcg, essa rappresenta comunque la soluzione di un problema simile. Poter scorrere tra le schermate di un'applicazione già implementata rende evidenti gli aspetti che potrebbero essere migliorati, quelli da rivisitare completamente e quelli da mantenere intatti.

Successivamente, ci si è concentrati sulla definizione formale delle specifiche dell'applicazione PulsEcg. Ciò significa visualizzare in maniera chiara quali funzionalità sono richieste, in quale contesto devono essere fornite e da quale tipo di utente verranno sfruttate. Questo studio deve essere rigoroso e deve produrre una serie di documenti che schematizzino gli obiettivi del progetto (perlopiù attraverso tabelle e grafici). Senza una visione complessiva e ben definita delle finalità dello sviluppo infatti, la fase di design sarebbe fortemente soggetta ad errori e/o ripensamenti per cui bisognerebbe tornare indietro alle specifiche, rallentando il processo.

Per finire si è passati al design. In questa fase le funzionalità elencate al punto precedente sono state scomposte in sottotask, in modo da poterne assegnare la logica a diversi moduli dell'applicazione. Questo step, così come i precedenti, è del tutto indipendente dall'ambiente di sviluppo e dal sistema operativo d'esecuzione. Nonostante ciò, la fase di design permette di entrare nel dettaglio di come strutturare concettualmente il codice che verrà implementato.

4.1 Applicazione Android ECG Watch

L'**applicazione ECG Watch** è una semplice interfaccia per il sistema operativo Android in grado di comunicare con il bracciale ECG Watch grazie al protocollo Bluetooth 2.0. L'ECG Watch, come è stato spiegato nel capitolo 2.1, è un bracciale dotato di elettrodi per l'acquisizione del segnale elettrocardiografico e di un blocco per la comunicazione bluetooth. L'applicazione è essenziale, priva di algoritmi di filtraggio del segnale, calcolo dei picchi e rilevazione di anomalie. Essa permette esclusivamente di acquisire l'ECG, visualizzarlo e condividerlo in formato txt.



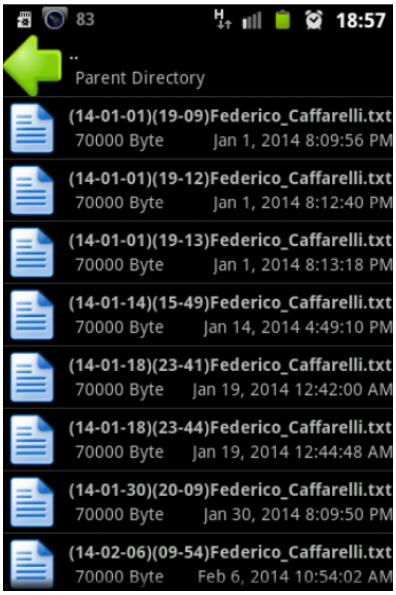
(a) SplashScreen **(b)** HomePage

Figura 4.1: Schermate principali dell'applicazione ECG Watch

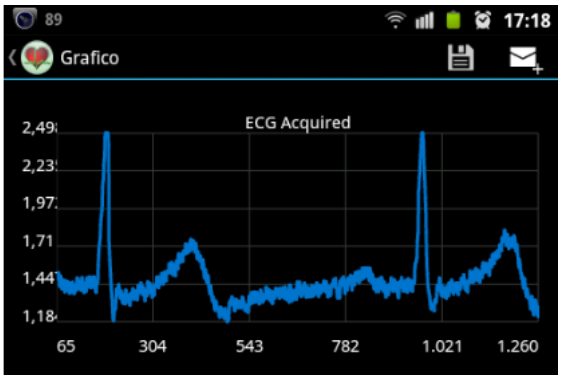
Una volta avviata, l'applicazione mostra uno splashscreen introduttivo e richiede l'abilitazione del bluetooth. Viene quindi caricata la view della schermata principale, che funge da menu. Questa pagina è essenziale. Permette tre azioni, attraverso dei pulsanti: un bottone per avviare l'acquisizione; una freccia per mostrare il grafico dell'ultima acquisizione; un libro per aprire l'intero archivio delle misurazioni ECG. Le quattro schermate (Activity) dell'applicazione sono:

1. **SplashScreen :** è l'activity che si avvia subito dopo il lancio dell'applicazione. Dura pochi secondi e mostra un cuore pulsante animato con un suono in sottofondo che simula il battito cardiaco.

2. **MainActivity :** è la classe più importante. Incapsula la logica di funzionamento, è responsabile di generare altri thread e di istanziare le altre classi. A livello funzionale si tratta della schermata che permette all'utente di muoversi tra i vari servizi.
3. **BTManager :** è una classe che, una volta avviata, funziona in background senza alcuna interfaccia visibile e gestisce la connessione con il dispositivo Bluetooth.
4. **FileManager :** è l'activity che implementa il file manager. Questa classe permette all'utente di scegliere una delle registrazioni salvate per visualizzarla, inviarla o cancellarla.
5. **ShowGraphActivity :** è la classe che si occupa della visualizzazione della forma d'onda appena acquisita o memorizzata in un file. [15].



(a) Archivio delle misurazioni



(b) Grafico ECG

Figura 4.2: Schermate di visualizzazione delle misurazioni dell'applicazione ECG Watch

Così come l'ECG Watch ha rappresentato il punto di partenza per la progettazione di un nuovo dispositivo che permettesse di acquisire anche il PPG e di ridurre le interferenze sull'ECG, questa applicazione è stata utile per poter prendere spunto nello sviluppo di una nuova versione per il PulsEcg.

Con la sua osservazione sono stati identificati prima di tutto i suoi **migliori aspetti progettuali**, da replicare nella nuova applicazione, come ad esempio la

composizione generale delle schermate FileManager e ShowGraphActivity, o il principio di navigazione a partire da una HomePage in cui sono riassunte tutte le funzionalità.

Allo stesso tempo sono evidenti **diversi limiti**, a partire dall'estetica non uniforme ed eccessivamente contrastata, in grado di rendere confusionaria anche una pagina estremamente semplice.

La schermata di visualizzazione del grafico può essere notevolmente migliorata. L'esportazione della misurazione dovrebbe essere possibile in diversi formati, in modo che sia disponibile un'alternativa anche per chi non possiede l'applicazione, come il pdf. Il segnale inoltre deve essere filtrato per una migliore visualizzazione ed elaborato per ricavare altre utili informazioni, quali ad esempio la fibrillazione atriale. Un altro limite di questa schermata consiste nella rappresentazione del grafico: la griglia su cui è tracciato non rispecchia la carta millimetrata utilizzata dai medici per leggere l'ECG, perciò non è di facile interpretazione.

L'acquisizione avviene mediante un bracciale, che l'utente può quindi indossare sia sul braccio sinistro che sul braccio destro. L'applicazione dovrebbe offrire la possibilità di impostare il braccio d'acquisizione tra una misurazione e l'altra, così che l'applicazione possa invertire i campioni acquisiti in base alla derivazione da cui sono stati rilevati, prima di visualizzarli sullo schermo.

Il file Manager deve essere organizzato meglio, nell'ottica di ospitare decine e decine di misurazioni, senza che l'esperienza di navigazione tra file peggiori. Le possibili soluzioni sono la suddivisione dei file in sezioni temporali e l'inserimento di un'opzione di ricerca.

Ovviamente a queste problematiche si aggiungono le sfide dovute all'integrazione di nuove funzionalità nel bracciale. L'applicazione PulsEcg dovrà possedere un'estensione dell'interfaccia dedicata alla visualizzazione della pressione, del PPG e dei dati da esso ricavati. A questo si somma tutta la parte elaborazione dei segnali ed integrazione della rete neurale.

Sviluppare l'applicazione perchè sia ospitata da un altro sistema operativo rispetto ad Android, ovvero iOS, significa inoltre dover selezionare nuovi approcci e soluzioni che siano più idonei a questo ambiente. Nei prossimi paragrafi verranno analizzati i requisiti del problema e poi verrà proposto un design per l'applicazione iOS.

4.2 Fasi preliminari all'implementazione

Il corretto processo di progettazione ed implementazione di un sistema, così come definito nel campo dell'ingegneria del software, dovrebbe essere strutturato attraverso delle precise fasi in successione:

1. Definizione delle **specifiche** di sistema;
2. **Progettazione e design** del sistema;
3. **Implementazione** effettiva;
4. **Integrazione** delle varie componenti del sistema;
5. **Testing** del sistema.

In molti casi, così come in quello del PulsEcg, il sistema comprende sia una componente hardware (il bracciale) che una software (l'applicazione). Dopo una definizione dei requirements e del design generale del sistema complessivo, tutte le fasi devono quindi essere ripetute separatamente per le componenti hardware e per quelle software (come mostrato in Fig. 4.3). Le due sezioni sono poi integrate tra loro e testate nel complesso. La tesi presente si focalizza sulla sola applicazione iOS, e quindi lo studio dei requisiti e del design che verrà presentato nei prossimi paragrafi riguarda il solo lato software della applicazione iOS, noti a priori i requirements generali di sistema.

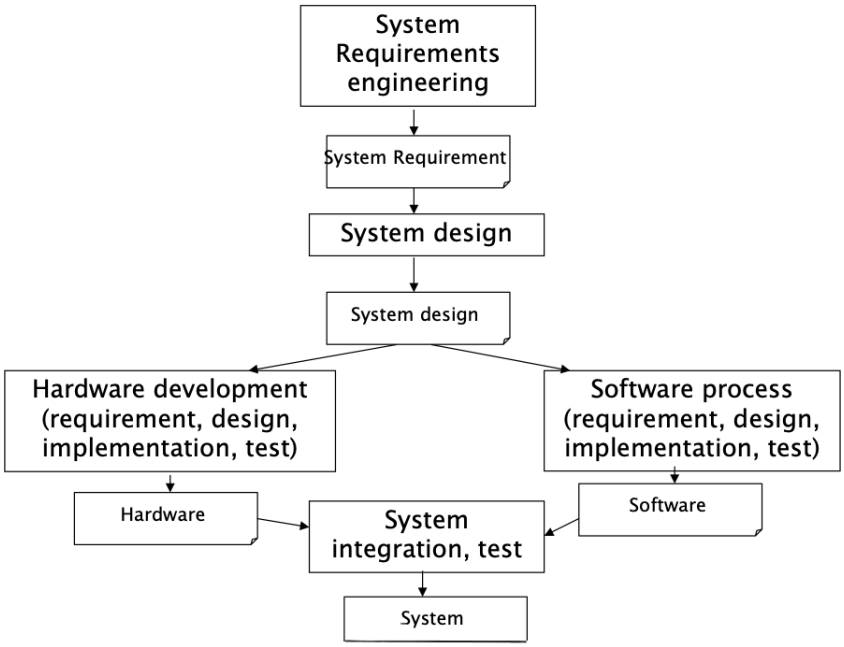


Figura 4.3: Processo di realizzazione di un sistema composto da una parte hardware ed una software

4.2.1 Definizione dei Requirements

Questa sezione riguarda la definizione accurata delle proprietà di cui il prodotto dovrebbe essere dotato, prima di iniziarne effettivamente lo sviluppo. Senza questa fase preliminare, le caratteristiche del sistema non sarebbero chiare, e quindi si ricadrebbe facilmente in errori ed incompresioni durante l'implementazione software. I requirements producono un documento completo e coerente, in cui tutte le caratteristiche del prodotto sono descritte nel dettaglio senza conflitti o contraddizioni.

Stakeholders e Interfacce

Stakeholder	**Descrizione**
Paziente	è colui che interagisce direttamente con il sistema, indossando il PulsEcg Watch e usando l'app per acquisire i propri segnali ed archiviarli
Medico	è il medico curante del paziente. Egli non utilizza il bracciale e può scegliere se utilizzare o meno l'applicazione. Infatti può interfacciarsi direttamente, usandola per visualizzare e archiviare le acquisizioni dei pazienti. Altrimenti può interfacciarvisi indirettamente prendendo visione dei pdf da essa generati.
Servizio Email o di messaggistica	è necessario per poter garantire comunicazione tra il paziente ed il medico curante.
Servizio GPS	permette di ricavare la posizione geografica del paziente.
PulsEcg Watch	è usato per ottenere i campioni acquisiti durante le acquisizioni. Fa parte del sistema PulsEcg, ma in questo caso stiamo valutando i requirements della sola applicazione iOS.
PulsEcg Android App	ha interesse ad aprire i file generati dall'app iOS. Anche in questo caso l'app Android è considerata esterna all'applicazione iOS ma è ovviamente interna al sistema PulsEcg.

Tabella 4.1: Stakeholders coinvolti nel sistema

Gli stakeholders sono le persone o gli altri sistemi che si interfacciano direttamente con il sistema e che lo possono influenzare, oltre che esserne influenzati. Questa fase è essenziale per visualizzare chiaramente le parti che saranno in gioco e iniziare

ad ipotizzare i loro bisogni. Gli stakeholders del sistema PulsEcg sono elencati in Tab. 4.1

Una volta individuati gli stakeholders, è necessario elencare il modo in cui questi si interfacciano con il sistema. Le interfacce vanno analizzate sia logicamente che fisicamente (Tab. 4.2).

Attore	**Interfaccia Fisica**	**Interfaccia Logica**
Paziente	Dispositivo dotato di iOS (iPhone o iPad) e connessione ad internet	Graphic User Interface dell'applicazione
Medico	Dispositivo dotato di iOS (iPhone o iPad) con connessione ad internet o qualsiasi dispositivo in grado di visualizzare un pdf e di ricevere mail o un semplice foglio	GUI dell'applicazione o lettore di PDF
Mail Service	Connessione Internet e protocolli	APIs
GPS Service	Connessione Internet	APIs
PulsEcg Watch	Modulo bluetooth	APIs
PulsEcg Android App	Dispositivo dotato di iOS (iPhone o iPad) e connessione ad internet	File System del dispositivo

Tabella 4.2: Interfacce necessarie tra attori e sistema

Storie e Persone

Quella che verrà presentata ora è una tecnica di software engineering per definire in modo informale cosa dovrebbe fare il sistema e soprattutto quali bisogni dell'utente soddisfi. Inquadrare l'utente a cui è destinato il prodotto è parte fondamentale della definizione dei requirements, perchè da questo dipenderanno gran parte delle scelte implementative, specialmente quelle riguardanti il front end. Una **Persona** è un sottoinsieme degli attori coinvolti nel sistema, nel nostro caso il solo paziente. Per ogni persona coinvolta è necessario inquadrare lo scenario (la **Storia**) tipico della sua vita in cui l'applicazione lo potrebbe aiutare. Il metodo delle Storie e delle Persone pone luce sugli obiettivi e sulle motivazioni del prodotto.

Evidenzia il cambiamento positivo che l'applicazione porterebbe nelle vite dei clienti, raccontandolo attraverso brevi fotogrammi informali delle loro vite.

1. **SeP-1** - Immaginiamo una persona con problemi di cuore piuttosto gravi, probabilmente anche anziana. Un utente di questo tipo necessita di fare visite continue di routine per controllare che i parametri siano sempre entro certi limiti. Questa persona può essere immaginata anche nel contesto della pandemia da SARS-CoV-2. Grazie al PulsEcg, sarà possibile ridurre al minimo le visite di routine perchè il bracciale stesso segnala quando sia il momento di indagare meglio sul proprio stato di salute. Oltre alla comodità del monitoraggio a domicilio, considerando la pandemia, il PulsEcg abbassa di gran lunga il rischio di contagio per medici e pazienti. L'applicazione, con la sua semplicità ed essenzialità, sarà facilmente utilizzabile anche da un anziano con poca esperienza d'uso nel mondo smart.

2. **SeP-2** - Un'altra situazione in cui l'uso del sistema PulsEcg potrebbe essere determinante è il caso di una persona che soffre di ipertensione arteriosa che non riesce da sola ad usare lo sfigmomanometro, o che presenta una pelle degli arti danneggiata. L'applicazione permetterebbe di misurare la pressione senza alcuna fatica e problematicità.

Requisiti funzionali e non funzionali

In questa sezione verranno elencati i requirements funzionali e non funzionali dell'applicazione PulsEcg. I requirements funzionali sono i servizi che l'applicazione deve offrire, in termini di singole funzionalità e comportamenti. I requirements non funzionali descrivono invece i vincoli sulle funzionalità, che possono riguardare il dominio, le performance, l'affidabilità e l'esperienza dell'utente. Tra i Non Functional Requirements (NFR) rientrano anche i vincoli dettati dalla legge, ad esempio il rispetto della privacy e di un certo grado di sicurezza. I Functional Requirements (FR) sono mostrati in Tab. 4.3, i NFR in Tab. 4.4. Ad ogni requisito, funzionale e non, sono stati associati degli ID brevi, così da poterne fare riferimento nelle prossime fasi di progettazione. Gli ID sono costituiti dal tipo di requisito (funzionale o non funzionale), da una lettera che ne indichi il tipo e da un contatore. Il tipo, nel caso dei NFR, è indicato nella colonna alla destra dell'ID. Per quanto riguarda i FR invece, il tipo è inteso come task di appartenenza. "B" ad esempio indica il bluetooth, perchè l'associazione, dissociazione e connessione sono attività riconducibili a questo aspetto, che potrebbero quindi far parte di una stessa schermata. "A" indica invece l'acquisizione di una misurazione da bracciale. "E" si riferisce all'elaborazione dei segnali acquisiti, quindi il filtraggio, il preprocessing e l'identificazione delle anomalie. "S" sta per show, ovvero l'ipotetica schermata di visualizzazione della misurazione. A questo task è associata anche la condivisione e

l'esportazione. Infine "O" indica le operazioni di configurazione, ed "F" l'uso del filesystem di archiviazione.

ID	**Descrizione**
FR_B1	Associazione ad un bracciale PulsEcg
FR_B2	Connessione automatica al bracciale associato
FR_B3	Dissociazione di un bracciale
FR_A1	Avvio di un'acquisizione
FR_A2	Salvataggio automatico di una misurazione in formato json
FR_E1	Filtraggio del segnale ECG
FR_E2	Filtraggio del segnale PPG
FR_E3	Calcolo dell'HB
FR_E4	Calcolo dell'SpO2
FR_E5	Calcolo delle pressioni
FR_S1	Visualizzazione ECG con griglia e picchi
FR_S2	Visualizzazione PPG
FR_S3	Visualizzazione valori di pressione, segnalazione di anomalie
FR_S4	Visualizzazione di un riassunto della misurazione
FR_S5	Esportazione PDF per mail
FR_S6	Esportazione file PDF
FR_S7	Esportazione JSON per email
FR_S8	Esportazione file JSON
FR_S9	Eliminazione file misurazione
FR_S10	Visualizzazione HB e segnalazione FA
FR_S11	Visualizzazione SpO2 e segnalazione anomalie
FR_F1	Importazione di file di misurazione
FR_F2	Esplorazione dei file salvati con possibilità di ricerca automatica
FR_S12	Condivisione della posizione, abitabile o meno
FR_O1	Selezione del braccio di acquisizione
FR_O2	Configurazione app con dettagli paziente e medico

Tabella 4.3: Requirements funzionali dell'app PulsEcg

Casi d'uso

I casi d'uso rappresentano le situazioni di interazione tra attori e sistema. In altre parole, essi descrivono il comportamento del sistema in varie condizioni, mentre risponde e soddisfa le richieste dell'utente, o eventualmente di altri attori descritti

ID	Tipo	Descrizione	Per
NFR_D1	Dominio	Le unità di misura del segnale ECG sono i secondi ed i millivolt	FR_S1
NFR_D2	Dominio	Il segnale PPG è espresso solo in secondi, ma non ha u.m. sull'asse y	FR_S2
NFR_D3	Dominio	I valori di pressione sono espressi in mmHg	FR_S3
NFR_D4	Dominio	L'HB è espresso in numero di battiti per minuto	FR_E3
NFR_D5	Dominio	La SpO2 è espressa sottoforma di percentuale nel sangue	FR_E4
NFR_U	Usabilità	L'applicazione deve essere minimale e intuitiva così che qualsiasi utente, soprattutto tra gli anziani, che rappresentano la più alta percentuale di pubblico interessato, possa imparare ad usarla in pochissimo tempo.	tutto
NFR_P	Privacy	I dati sanitari raccolti dalla applicazione devono essere salvati localmente nel dispositivo. Nessun database deve essere adoperato.	FR_A2
NFR_E	Efficienza	Gli algoritmi di elaborazione dei segnali e di intelligenza artificiale non devono aver bisogno di più di 1 secondo totale per misurazione, così da non generare scontento nell'utente.	FR_E1, FR_E2, FR_E3, FR_E4, FR_E5
NFR_M	Manutenibilità	Deve essere possibile aggiungere nuove caratteristiche bluetooth con poco sforzo, così che la logica per la trasmissione di nuovi dati dovuti all'aggiornamento dell' hardware sia già predisposta. Lo stesso vale per la durata di acquisizione, che deve essere gestita dal software mediante costanti.	tutto
NFR_I	Interoperabilità	I file JSON prodotti dall'applicazione devono essere compatibili anche con l'applicazione equivalente in ambiente Android.	FR_A2, FR_S7, FR_S8

Tabella 4.4: Requirements non funzionali dell'app PulsEcg

nella prima sezione dei requirements. I casi d'uso coinvolgono gli attori e contestualizzano le procedure descritte nei requirements funzionali, permettendo di visualizzarle come dei veri e propri algoritmi operativi. I casi d'uso dell'applicazione PulsEcg sono riassunti nello **Use Case Diagram**, suddiviso per semplicità in due grafici in Fig. 4.4 e 4.5 realizzati con lo strumento di modellazione Astah. Come si evince da questi grafici, i functional requirements sono stati connessi tra loro con delle frecce che ne rappresentano le relazioni. Sulla sinistra l'utente, è attore principale del sistema, ovvero colui che interagisce attivamente con l'applicazione, indirizzandone le operazioni. Sulla destra troviamo invece gli attori che beneficiano del prodotto dell'applicazione (come il medico), oppure che vengono interpellati/-comandati dal sistema, come il servizio Email e GPS e il dispositivo hardware. Le frecce continue indicano la comunicazione diretta con un attore, ovvero corrispondono ad un input-output mediante una delle interfacce precedentemente definite. Le frecce tratteggiate indicano inclusione, ovvero connettono degli FR che non devono essere direttamente richiesti dall'utente a dei task dovuti ad un input-output. L'Use Case Diagram permette di visualizzare per la prima volta quella che potrà essere la navigazione dell'utente attraverso l'applicazione.

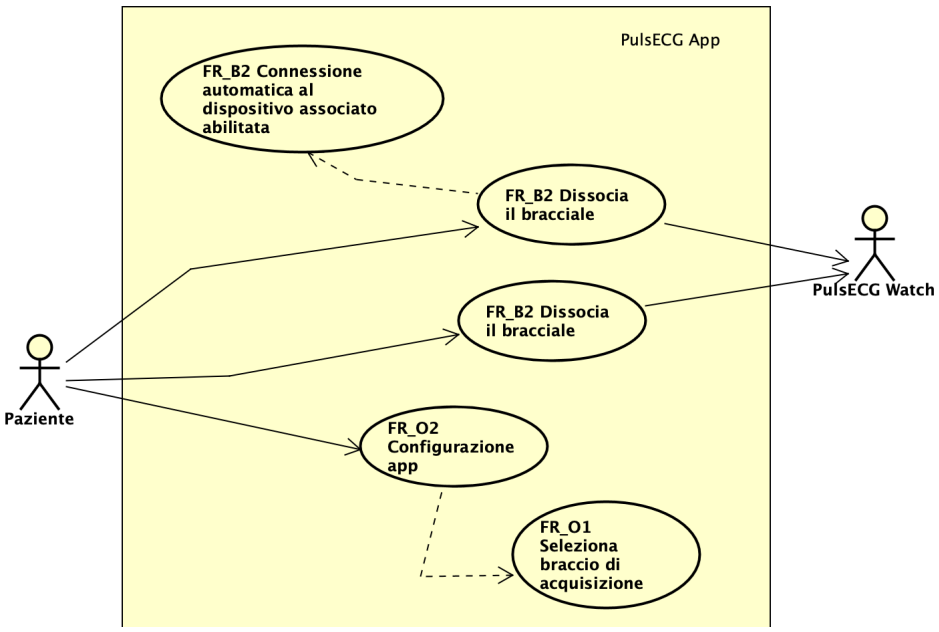


Figura 4.4: Use Cases dell'app iOS PulsEcg pt1

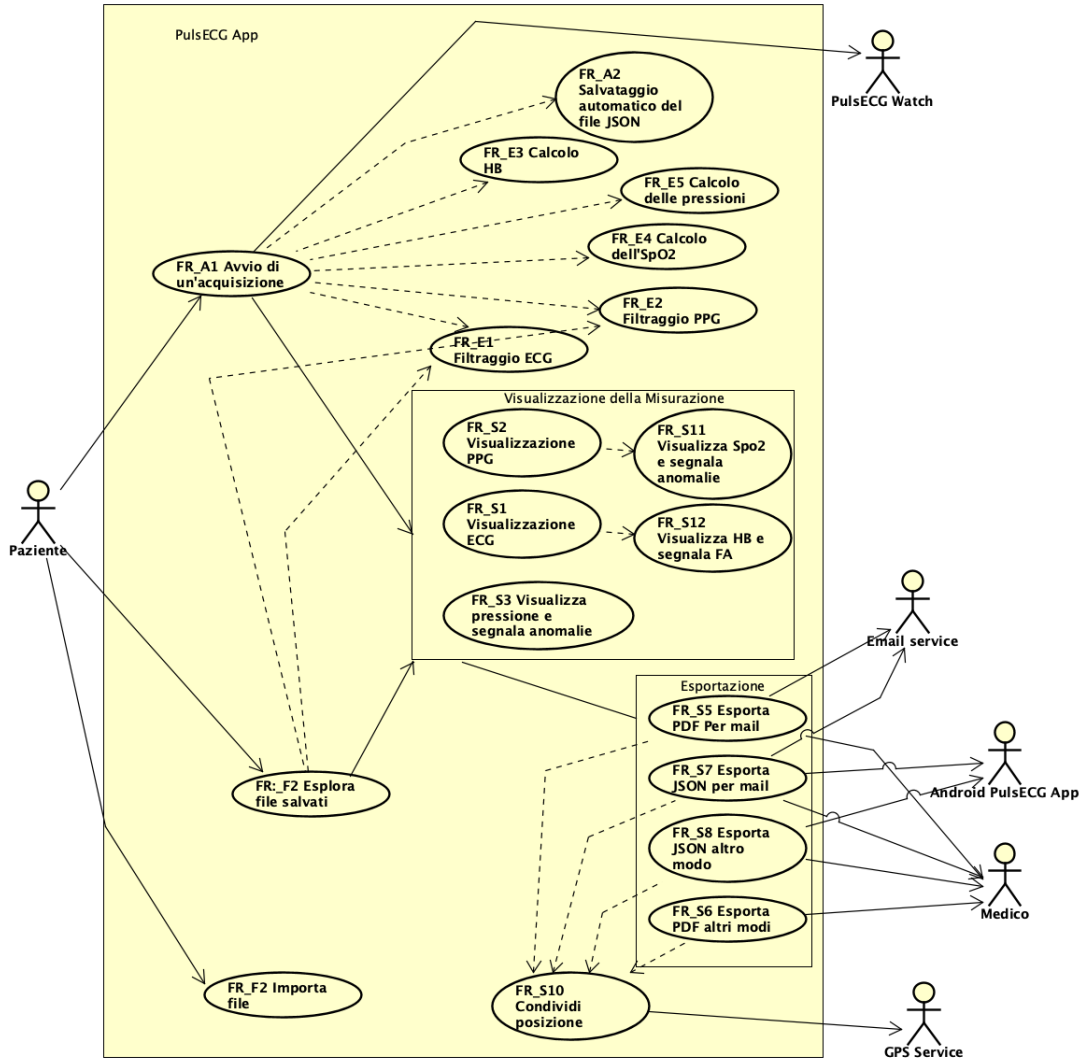


Figura 4.5: Use Cases dell'app iOS PulsEcg pt2

4.2.2 Design proposto per l'applicazione

Dopo la fase di definizione dei requisiti, si è passati al design dell'architettura. In questa fase è stato deciso quali sarebbero stati i componenti principali dell'applicazione e un loro quadro di controllo e comunicazione. Fare design permette di rendere esplicite le scelte progettuali dello step precedente, documentandole e rivalutandole. La progettazione software, in particolare, consiste nel definire i suoi principali componenti, come ad esempio le classi, e il modo in cui queste interagiranno tra loro. In questa fase sono definite le proprietà più importanti di

ogni classe e il loro coordinamento mediante chiamate a funzione.

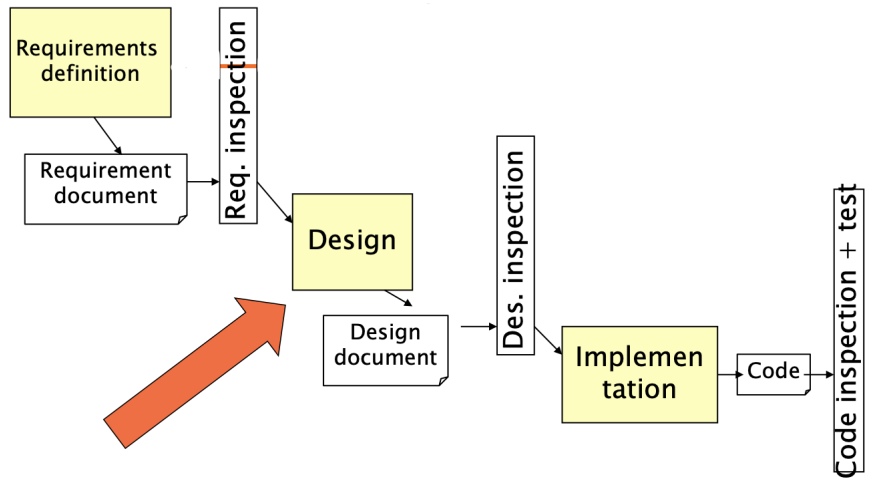


Figura 4.6: Fase di design del processo di sviluppo di un sistema

Package e Class Diagram

Il progetto XCode verrà organizzato in modo da contenere nella cartella principale il file Storyboard (file di interfaccia grafica e navigazione) e tre sottocartelle di cui una dedicata a tutti i ViewControllers, ovvero le classi dotate di interfaccia grafica (le schermate dell'applicazione), una dedicata alle classi di supporto o background prive di interfaccia (una per il bluetooth, una per la processazione dei segnali ecc) ed una per le TableViewCell, degli oggetti associati ad una vista che rappresentano una cella di una tabella. Quest'ultima cartella è necessaria perchè molte schermate dell'app saranno basate su viste tabellari di elementi, del tutto simili ai recycler view del mondo di android, per cui è necessario definire degli item. Il package è schematizzato in Fig. 4.7.

Le classi costituenti questo package si relazionano secondo quanto riportato nel Class Diagram in Fig. 4.8. Questa immagine evidenzia i principali attributi e metodi di ogni classe.

Matrice di tracciabilità

La matrice di tracciabilità infine viene utilizzata per assicurarsi che il design proposto nei punti precedenti garantisca una copertura totale dei requirements funzionali del sistema. La matrice di tracciabilità dell'applicazione PulsEcg è riportata in Tab. 4.6. Ad ogni riga della matrice è associato uno dei functional requirements precedentemente definiti, identificato per mezzo del suo ID (vedi Tab. 4.3). Sulle

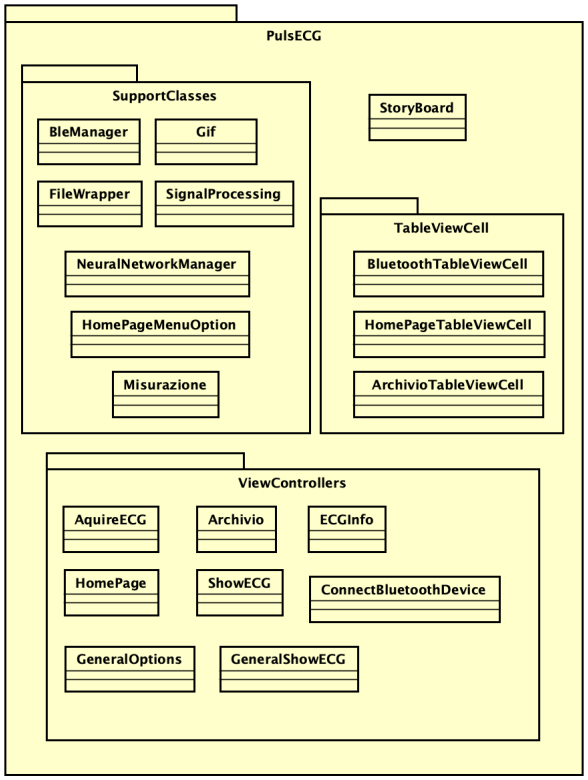


Figura 4.7: Struttura del package dell'app PulsEcg

colonne sono invece riportate le classi progettate attraverso il Class Diagram (Fig. 4.8). Per questioni di spazi, ad ogni classe è stato associato un ID (corrispondenze id-classe consultabili in Tab. 4.5), utilizzato come titolo della colonna di riferimento. Ogni funzionalità sarà implementata richiamando le funzioni di una o più classi. Se una classe è coinvolta nel soddisfacimento di un certo task, la matrice di tracciabilità fa corrispondere all'intersezione tra le due una X. Tutte le classi accompagnate dall'acronimo VC (che sta per View Controller) possiedono una interfaccia grafica con cui l'utente può interagire, dando il via a delle operazioni. Per ogni funzionalità quindi, la VC indicata con X è la classe visualizzata sullo schermo in quel momento, nonchè la classe che chiama le funzioni delle altre classi contrassegnate, che invece agiscono silenziosamente. Ad esempio, il FR FR_B1 (task di associazione di un bracciale all'applicazione) coinvolge ConnectBluetoothDevice VC, ovvero la classe adibita alla gestione bluetooth e munita di interfaccia grafica con cui l'utente può interagire per avviare delle operazioni. L'azione dell'utente su questo VC per associare un dispositivo scatena le chiamate a funzioni della classe BleManager, la quale agisce in background e avvisa il VC delle modifiche, sfruttando i metodi di notifica della classe NotificationManager. Se per una funzionalità la matrice di

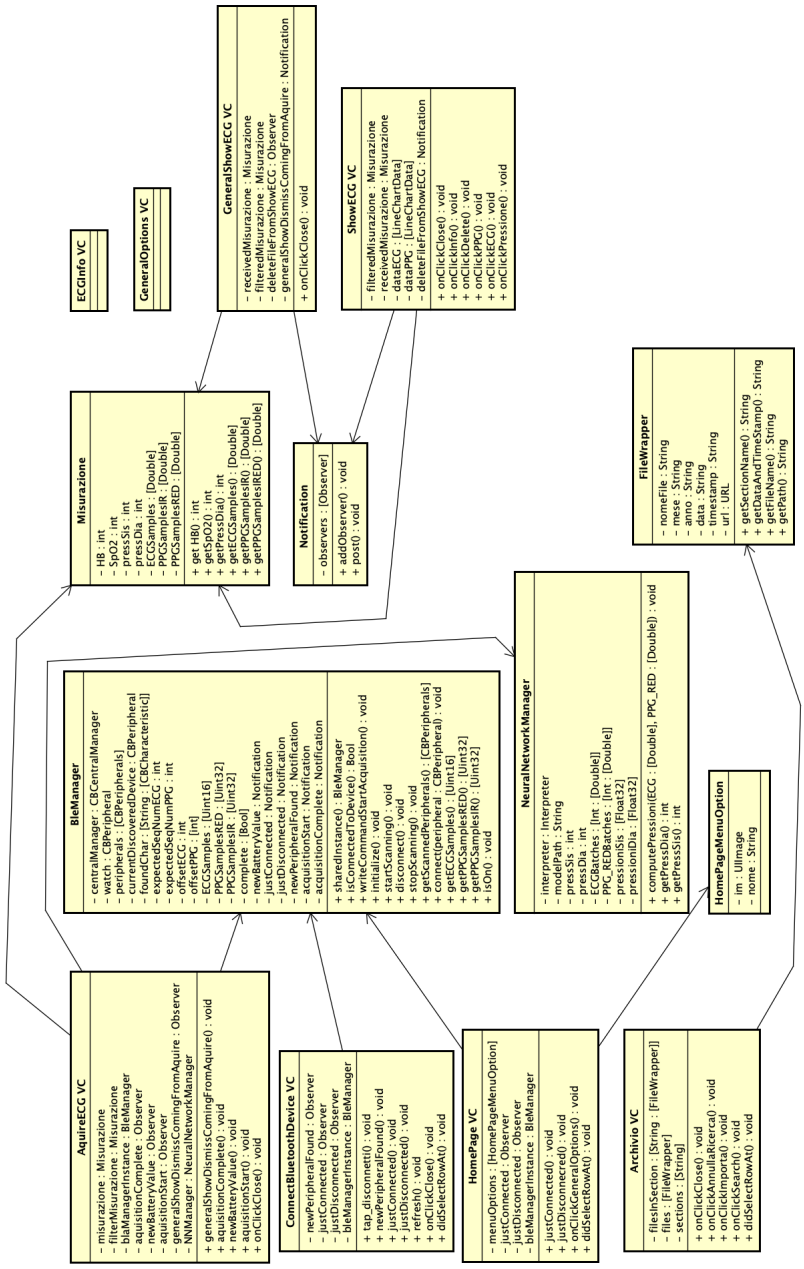


Figura 4.8: Class Diagram dell'app PulsEcg

tracciabilità contrassegna con una X due VC, significa che essa viene soddisfatta in due diversi punti dell'applicazione (dove per punti si intendono logicamente le schermate visualizzate dall'utente). Ad esempio le funzionalità di visualizzazione di alcuni dati della misurazione (come HB e pressione) saranno ripetute sia in

ShowECG VC che in GeneralShowECG.

ID	**Class Name**
HP	HomePage VC
Aq	AquireECG VC
Ar	Archivio VC
I	InfoECG VC
S	ShowECG VC
GS	GeneralShowECG VC
CB	ConnectBluetoothDevice VC
BM	BleManager
NNM	NeuralNetworkManager
FW	FileWrapper
N	Notification
M	Misurazione
GO	GeneralOptions VC

Tabella 4.5: ID delle classi per la lettura della matrice di tracciabilità 4.6. Gli ID sono scelti per semplice abbreviazione del nome della classe. Le classi seguite dall'acronimo VC sono quelle dotate di interfaccia grafica.

FR ID	**HP**	**Aq**	**Ar**	**I**	**S**	**GS**	**CB**	**BM**	**NNM**	**FW**	**N**	**M**	**GO**
FR_B1							X	X			X		
FR_B2	X							X			X		
FR_B3							X	X			X		
FR_A1		X					X				X	X	
FR_A2		X										X	
FR_E1		X			X	X						X	
FR_E2		X			X							X	
FR_E3		X										X	
FR_E4		X										X	
FR_E5		X							X			X	
FR_S1					X	X						X	
FR_S2					X							X	
FR_S3					X	X						X	
FR_S4						X						X	
FR_S5					X							X	
FR_S6					X							X	
FR_S7					X							X	
FR_S8					X							X	
FR_S9			X		X						X		
FR_S10					X	X						X	
FR_S11					X	X						X	
FR_F1				X						X			
FR_F2				X						X			
FR_S12					X								X
FR_O1													X
FR_O2													X

Tabella 4.6: Matrice di Tracciabilità dei FR. Le classi coinvolte nel soddisfacimento del FR riportato sulla riga della tabella sono contrassegnate con una X. Gli ID dei FR sono consultabili in Tab. 4.3, quelli delle classi in Tab. 4.5.

Capitolo 5

Ambiente iOS e Librerie

Una volta progettata l'applicazione a livello astratto e indipendente dall'ambiente, il passo successivo è stato quello di selezionare ed eventualmente personalizzare gli strumenti offerti da iOS necessari per sviluppare le funzionalità richieste.

5.1 Swift vs Objective-C

Il primo quesito che è stato posto riguarda la scelta del **linguaggio di programmazione** da utilizzare per lo sviluppo dell'applicazione. Nel mondo IOs sono infatti disponibili due possibilità: Objective-C (il linguaggio Apple tradizionale) e Swift.

Swift è il nuovo linguaggio di programmazione creato da Apple, lanciato nel 2014 per offrire un'alternativa a Objective-C, in quanto quest'ultimo non ha avuto evoluzione negli anni e ed è molto diverso dai tipici linguaggi di programmazione con sintassi C ++. Swift si ispira invece a C ++, C#, Java e Python, perciò la sua sintassi è semplice, non utilizza puntatori, include miglioramenti nelle strutture dati, aiuta gli sviluppatori a fare meno errori, incorpora nuove funzionalità e un nuovo paradigma di programmazione. Si tratta di un **linguaggio di programmazione imperativo e orientato agli oggetti** come Objective-C, ma in più incorpora anche la **programmazione funzionale** (permettendo l'uso delle chiusure) [22].

Swift è quindi un linguaggio più semplice, più potente, più sicuro (in quanto fortemente tipizzato e dotato di meccanismi di autogestione degli overflow e della memoria) e facilmente leggibile da chi conosce la sintassi C/Java (Fig. 5.1). Quest'ultimo aspetto è rilevante nell'ottica di eventuali futuri aggiornamenti dell'applicazione ad opera di altri programmatori, in quanto incontrerebbero sicuramente meno difficoltà nel comprendere il codice preesistente, anche fossero alle prime armi con Swift.

Le precedenti motivazioni, unite alla tendenza di Apple a spronare verso l'abbandono di Objective-C, hanno portato alla scelta di **Swift come linguaggio di sviluppo dell'applicazione PulsEcg**.

Objective-C

```
const int count = 10;
double price = 23.55;

NSString *firstMessage = @"Swift is awesome. ";
NSString *secondMessage = @"What do you think?";
NSString *message = [NSString stringWithFormat:@"%@%@", firstMessage, secondMessag
e];

NSLog(@"%@", message);
```

Swift

```
let count = 10
var price = 23.55

let firstMessage = "Swift is awesome. "
let secondMessage = "What do you think?"
var message = firstMessage + secondMessage

print(message)
```

Figura 5.1: Esempio della maggiore leggibilità di Swift rispetto a Objective-C.

5.2 Elementi base di un programma Swift

In questa sezione verranno presentati i principali componenti di una applicazione iOS realizzata in Swift, in modo da agevolare la comprensione di quanto verrà affrontato nel resto di questo capitolo.

Nello specifico, la prima parte introduce il lettore ai View Controllers, entità del linguaggio Swift aventi corrispondenza diretta con una pagina dell'applicazione visualizzata sullo schermo. Ogni View Controller è gestore della sua vista, ma anche gestore delle funzionalità nascoste dietro di essa.

Successivamente, si parlerà brevemente di come muoversi attraverso View Controllers, realizzando la navigazione nell'applicazione. L'elemento Swift utilizzato a questo scopo è chiamato Segue.

Infine, verrà data una breve descrizione dello strumento Storyboard, l'Interface Builder offerto in programmazione Swift per l'implementazione dell'interfaccia grafica, per poter definire la navigazione all'interno di un'app e per connettere gli elementi grafici alla logica dei View Controllers.

5.2.1 UIViewController

Gli elementi fondamentali di una applicazione iOS sono chiamati UI View Controllers. Un **UIViewController (VC)** è un'entità a cui è associata un'interfaccia grafica che occupa tutto lo schermo, organizzata in subviews. Ogni UIViewController rappresenta un certo task dell'applicazione quindi, oltre ad una view, possiede anche la logica necessaria per il suo funzionamento. Quando l'utente si muove tra le schermate dell'app, si sta di fatto spostando tra un UIViewController e l'altro. In Android si tratta del corrispettivo di una Activity.

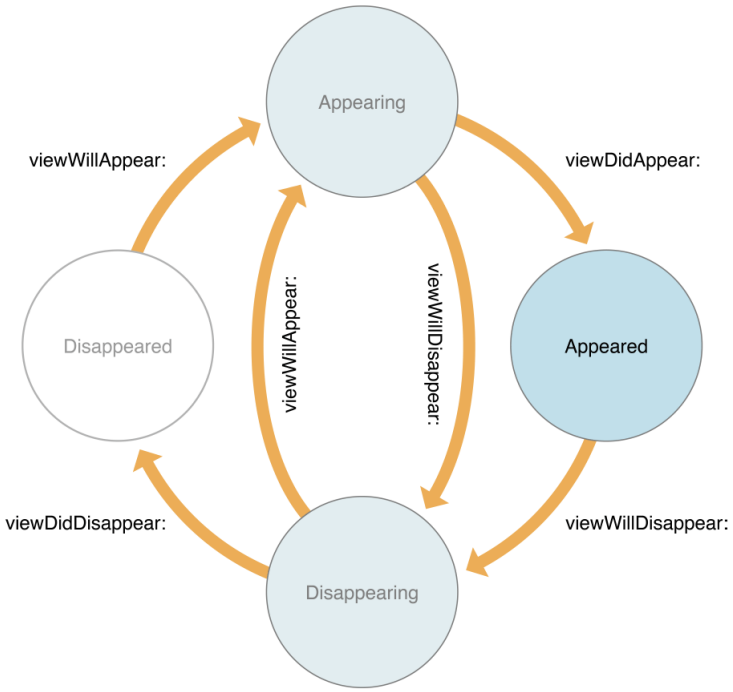


Figura 5.2: Stati del ciclo di vita di un VC

Un oggetto della classe UIViewController possiede una serie di metodi che gestiscono la sua gerarchia di visualizzazione. iOS chiama automaticamente queste funzioni al momento giusto, ogni volta che il VC transita da uno stato all'altro del suo **ciclo di vita**. Il programmatore può fare l'override di questi metodi per popolare il VC con il codice necessario all'esecuzione del task di cui è responsabile

[23]. Gli stati del VC e le callback chiamate ad ogni transizione sono schematizzate in Fig. 5.2.

1. **viewDidLoad()** - Chiamato quando la view del VC viene creata e caricata da uno storyboard (file di layout e navigazione di un'applicazione iOS). iOS chiama viewDidLoad () solo una volta, quando viene creata la vista del VC. I metodi successivi invece, sono chiamati più volte dopo la creazione del VC, in relazione alla sua visibilità sullo schermo. Questo metodo funge quindi da entry point della logica del VC.

2. **viewWillAppear()** - Chiamato appena prima che la view venga aggiunta alla gerarchia di visualizzazione dell'app. La sua funzione parallela è la viewWillDisappear().

3. **viewDidAppear()** - Chiamato subito dopo che la view del VC è stata aggiunta alla gerarchia di visualizzazione dell'app. La sua funzione parallela è la viewDidDisappear().

5.2.2 Segue

Per navigare tra un VC e l'altro, Swift mette a disposizione i Segue. Un **Segue** definisce una transizione tra due VC, riferendosi ad un collegamento definito nel file storyboard dell'app. Il punto di partenza di un segue può essere un pulsante, la riga di una tabella o il riconoscimento di una certa gesture. Il punto finale di un segue è il VC che si desidera visualizzare [23]. La view del VC destinazione sarà posta in cima alla gerarchia di visualizzazione dell'applicazione, coprendo quella del VC sorgente. Durante un segue il VC sorgente può passare dei dati al VC destinazione, grazie al metodo *prepare* di cui è possibile fare override.

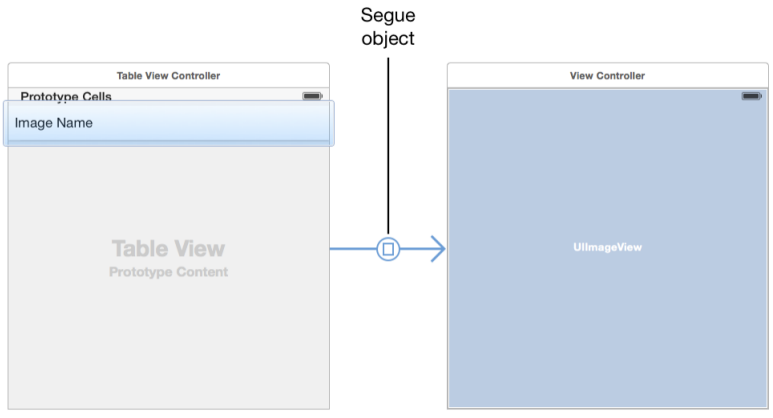


Figura 5.3: Rappresentazione di un Segue tra due VC

5.2.3 Interface Builder

L'Integrated Development Environment (IDE) usato per lo sviluppo delle applicazioni iOS è **XCode**. Esso offre un meccanismo per semplificare lo sviluppo di interfacce grafiche e la definizione della navigazione tra di esse. Tutto ciò viene poi salvato in un file chiamato **StoryBoard**. I tool di questo tipo sono chiamati Interface Builder. Apple ha deciso di introdurlo in quanto gli utenti desiderano uno strumento affidabile che combini sia la prototipazione rapida, che interfacce target completamente operative, fornendo allo stesso tempo funzionalità elevate. E' desiderabile inoltre che lo strumento sia facile da usare e da imparare, e che sia rapido nell'esecuzione [24].

Usando l'Interface Builder, ogni VC viene disegnato separatamente nello StoryBoard. Successivamente i VC vengono connessi tra loro definendo nuovi segue, di cui è possibile specificare il tipo. Gli oggetti grafici usati all'interno di un VC (che estendono la classe UIView), per far sì che siano modificabili programmaticamente o che scatenino degli eventi al tocco, devono essere connessi al codice. Ciò è possibile mediante:

1. **Actions** - fornite solo dagli oggetti che prevedono già un interazione, come i bottoni. Sono funzioni che vengono lanciate al tocco di oggetti interattivi.
2. **Outlets** - forniti per tutti gli oggetti grafici. Si tratta di riferimenti nel codice alle viste. Accedendo ai riferimenti, è possibile modificarne l'aspetto ed eventualmente rimuoverle dalla gerarchia di visualizzazione.

I VC che compongono l'applicazione PulsEcg ed i Segue che ne descrivono il rapporto, ovvero la navigazione, verranno trattati nel dettaglio dal paragrafo 6.2 in poi.

5.3 NotificationCenter

Non tutte le classi che compongono l'app PulsEcg estendono un UIViewController. Alcuni task complessi infatti, come ad esempio la gestione del bluetooth, sono gestiti da classi prive di view ed agiscono in background, aggiornando di tanto in tanto il modello dei dati. Visto che il modello deve essere riflesso nella vista, è necessario un **meccanismo asincrono di notifica** che permetta di aggiornare la view del VC correntemente visualizzato, o addirittura di transitare su un altro VC, appena avvengono delle modifiche nei dati.

Per realizzare questo comportamento nell'app PulsEcg, si è utilizzato un **pattern Observer**. Il pattern Observer consente ad un'entità di tenere traccia delle attività

operate da una o più altre entità. Questo pattern definisce un'interfaccia che gli oggetti osservati possono usare per notificare all'osservatore il verificarsi di un evento e fornisce un meccanismo per aggiungere o rimuovere oggetti da osservare. L'osservatore percepisce una notifica quando lo stato dell'oggetto osservato cambia. Di solito c'è un accoppiamento minimo tra l'osservatore e le entità osservabili. Ogni osservatore implementa e aggiorna l'interfaccia, mentre l'oggetto osservato non sa nulla dell'esistenza dell'osservatore, ne tantomeno della vista [25].

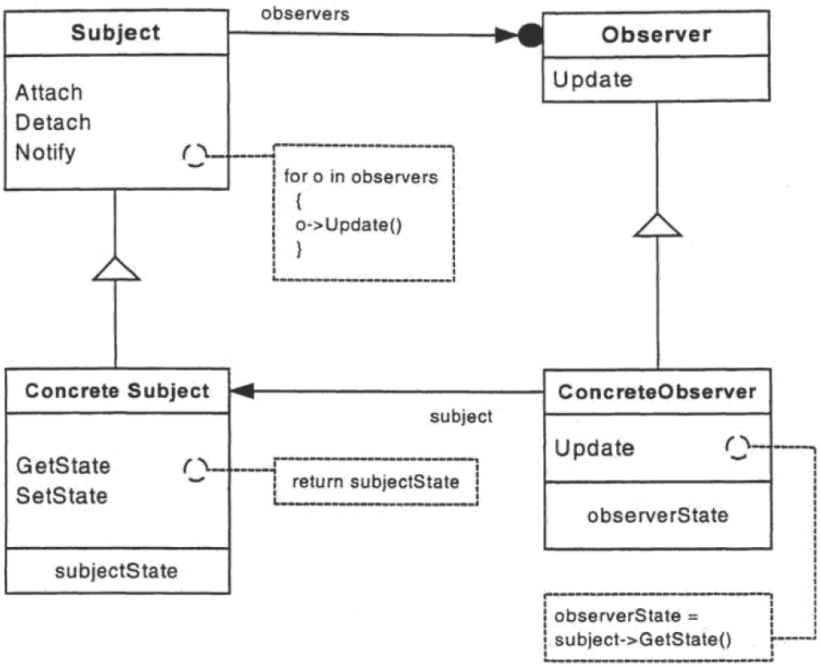


Figura 5.4: Pattern Observer

Lo strumento Swift utilizzato nell'applicazione per implementare questo pattern è il **NotificationCenter**. Le classi prive di vista vengono estese, in modo da implementare un centro di notifica. In ogni centro sono registrate un certo numero di notifiche, identificate da una stringa. I ViewController interessati si registrano quindi ad uno più centri di notifiche, utilizzando i metodi *addObserver*. Quando un oggetto si aggiunge come osservatore, specifica quali notifiche desidera ricevere mediante l'identificativo. Un oggetto può quindi chiamare questo metodo più volte per registrarsi come osservatore per diverse tipi di notifiche di uno o più centri [23]. Per ogni iscrizione ad una notifica, il VC deve anche **associare un handler della notifica**, ovvero una funzione che verrà scatenata quando viene intercettata la notifica osservata. Sono gli handler a modificare effettivamente la view. La classe che estende il NotificationCenter può mandare le notifiche mediante una azione di

post non appena ha terminato di modificare i dati da visualizzare.

5.4 Libreria CoreBluetooth

La comunicazione tra PulsEcg e l'applicazione è realizzata mediante Bluetooth Low Energy. In ambiente iOS, tale comunicazione può essere gestita utilizzando il framework CoreBluetooth.

5.4.1 Bluetooth Low Energy

Il **Bluetooth Low Energy (BLE)** è stato introdotto nel 2011 come caratteristica distintiva del Bluetooth 4.0. Il BLE è studiato per le applicazioni che richiedono un trasferimento episodico o periodico di piccole quantità di dati, pertanto è particolarmente adatto a sensori, attuatori e altri piccoli dispositivi per cui il consumo energetico deve essere estremamente basso. Il Ble garantisce buone prestazioni con un numero elevato di nodi di comunicazione, un consumo energetico molto basso, una robustezza pari al Bluetooth classico, brevi tempi di attivazione e connessione ed un buon supporto per smartphone e tablet [26].

Nella comunicazione BLE, un dispositivo ed un cellulare scambiano dati utilizzando il protocollo **General Attribute Profile (GATT)**, che si appoggia al meccanismo di trasporto ATT.

L'**Attribute Protocol (ATT)** fornisce un meccanismo per scoprire gli attributi di un dispositivo remoto e per leggere o scrivere su questi attributi. ATT segue un modello client server. Il server espone una serie di attributi al client, ed esso può può scoprirli, leggerli e scriverci sopra. Un attributo è qualcosa che rappresenta un dato che il dispositivo desidera condividere con altri. Il protocollo ATT è progettato per inviare o estrarre i dati provenienti da o diretti verso un dispositivo remoto, offrendo anche un servizio di notifiche in modo che il dispositivo remoto possa essere avvisato quando i dati cambiano. Un dispositivo può implementare il solo client, il solo server o entrambi i ruoli contemporaneamente. Tuttavia, solo un server può essere attivo su uno stesso dispositivo [27].

Il GATT si basa su servizi, caratteristiche e descrittori. Un dispositivo può avere più servizi, ciascuno dei quali offre carattertistiche contenenti un valore (attributo) che può essere letto, scritto o sottoscritto per le notifiche. Ad ogni servizio e ad ogni caratteristica è associato un UUID (Universally Unique Identifier) che lo identifica. Lo schema di comunicazione opera come segue:

1. il dispositivo (periferica) trasmette un messaggio in cui annuncia la propria disponibilità;

2. il dispositivo centrale (il cellulare) esegue la scansione delle periferiche disponibili;

3. una volta riconosciuta in fase di scanning la giusta periferica, il dispositivo centrale interrompe la scansione e avvia una procedura di connessione con essa;

4. il dispositivo centrale esplora il dispositivo periferico, analizzandone tutti i servizi, le caratteristiche e i descrittori;

5. il dispositivo centrale scambia informazioni con il dispositivo periferico utilizzando le caratteristiche del servizio scelto.

Ad ogni caratteristica sono associate delle proprietà che definiscono le azioni possibili su di essa: lettura, scrittura e notifica. Le richieste di lettura (da dispositivo) e scrittura (su dispositivo) trasmettono un singolo valore. Per ottenere più dati, suddivisi in pacchetti, o per ricevere aggiornamenti periodici, vengono utilizzate le notifiche. Il dispositivo centrale si pone in ascolto su una caratteristica specifica e il dispositivo periferico invia i dati in modo asincrono. Prima dell'ascolto però, l'applicazione richiede una scrittura su un descrittore dedicato [28].

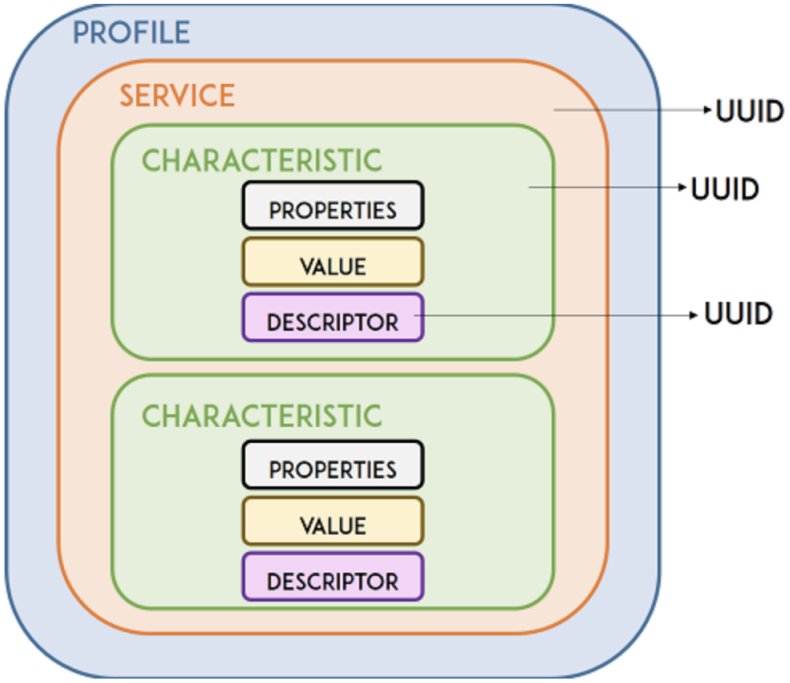


Figura 5.5: Architettura GATT.

5.4.2 Il servizio ECG

Questo capitolo, ora che si è presa confidenza con il meccanismo di comunicazione tipico del BLE, si focalizza sulla descrizione del nuovo servizio GATT ad hoc che è stato generato e integrato nell'hardware, allo scopo di trasmettere i segnali acquisiti all'applicazione (come anticipato nel capitolo 2.2). Il servizio ECG (UUID: 0xBABE) possiede 4 caratteristiche:

1. **Comando di start aquisition in modalità write :** l'applicazione scrive su questa caratteristica per informare il dispositivo in merito all'inizio dell'acquisizione.

2. **Dati ECG in modalità Notify :** utilizzata per trasferire da dispositivo ad applicazione i pacchetti contenenti i campioni dell'ECG. Il dispositivo inizia a scrivere i dati sulla caratteristica solo a seguito della scrittura sulla caratteristica di start.

3. **Dati PPG in modalità Notify :** utilizzata per trasferire da dispositivo ad applicazione i pacchetti contenenti i campioni del PPG. Il dispositivo inizia a scrivere i dati sulla caratteristica solo a seguito della scrittura sulla caratteristica di start.

4. **Dati Batteria in modalità Notify :** utilizzata per trasferire da dispositivo ad applicazione un pacchetto contenente i dati sul livello della batteria. Il dispositivo scrive il dato sulla caratteristica solo a seguito della scrittura sulla caratteristica di start.

5.4.3 Le funzioni di CoreBluetooth

In ambiente iOS, la comunicazione con dispositivi BLE è possibile grazie alle API offerte da **CoreBluetooth**. Le classi di nostro interesse fornite dal framework CoreBluetooth sono: CBCentral, CBPeripheral, CBService, CBCharacteristic e CBUUID. Il CBCentral è il dispositivo che fa scan, ovvero l'applicazione. La gestione delle varie fasi della connessione è realizzata principalamente mediante diverse funzioni:

1. *scanForPeripherals* - è un metodo chiamabile dal programmatore su di un CBCentralManager. Permette di avviare lo scanning di periferiche, specificando eventualmente il servizio desiderato. Non si può fare l'override di questa funzione.

2. *didDiscover* - viene chiamato automaticamente una volta attivato lo scanning, nel momento in cui viene individuata una periferica. Se ne può scrivere l'override.

3. *connect* - è chiamabile dal programmatore, eventualmente nel corpo della didDiscover. Richiede la connessione con una certa periferica.

4. *didConnect* - viene chiamato automaticamente nel momento in cui la connessione con una certa periferica è avvenuta. Questo metodo è sovrascrivibile. Da questo metodo parte l'esplorazione dei servizi presenti nella periferica, grazie al metodo discoverServices, che causerà l'esecuzione del didDiscoverServices, nel quale è possibile chiamare discoverCharacteristics che porterà all'esecuzione di didDiscoverCharacteristicsFor. Se tutti i servizi e le caratteristiche di interesse sono presenti, allora l'applicazione resta connessa al dispositivo, altrimenti è necessario un meccanismo di disconnessione.

5. *didUpdateValueFor* - è chiamata automaticamente ogni volta che una caratteristica impostata in modalità Notify viene aggiornata con un nuovo valore. Al suo interno è necessario scrivere la logica di identificazione della caratteristica sorgente, nel caso ci fossero più notifiche da UUID diversi, e il codice per l'elaborazione del dato.

L'implementazione specifica di queste funzioni nell'applicazione PulsEcg verrà affrontata nel capitolo 7.1.

5.5 Libreria CoreLocation

Una delle funzionalità richieste nei requirements dell'applicazione PulsEcg è il **reperimento della localizzazione geografica del paziente**. Il paziente può infatti abilitarne la condivisione, così da allegare la posizione geografica ai pdf e alle email generate automaticamente a partire da una acquisizione. Questa funzionalità è già disponibile nella versione dell'app sviluppata in questa tesi, ma il suo potenziale potrebbe essere ancor più evidente con i prossimi aggiornamenti del dispositivo, nel caso venisse integrato un accellerometro per identificare ad esempio una caduta improvvisa. Ciò potrebbe generare un allarme nella applicazione con immediata condivisione della posizione dell'utente.

La libreria (o API) che consente di ottenere informazioni sulla posizione di un dispositivo iOS è il framework **Core Location**. Core Location può utilizzare tutti i metodi di geolocalizzazione che il dispositivo mette a disposizione come A-GPS, Wi-Fi, triangolazione di torri cellulari, ecc. Core Location offre tre servizi:

1. *Standard Location Service*: così chiamato perché disponibile in tutti i dispositivi iOS e in tutte le versioni iOS. Questo è il servizio più adatto quando non serve un costante aggiornamento della posizione, ma si desidera un'alta precisione.

2. *Significant Change Location Service*: segnala la posizione corrente e notifica all'utente le modifiche a quella posizione con un basso consumo energetico. Questo servizio è il più adatto nel caso in cui servissero dei continui aggiornamenti della posizione;

3. *Region Monitoring Service* : un servizio per la definizione di regioni e di notifica del superamento dei confini [29].

Nel caso dell'app PulsEcg, il servizio più adatto è l'SLS, perchè la posizione deve essere rilevata una sola volta, nell'istante in cui si decide di allegarla ad un file da esportare dall'applicazione, ma l'indirizzo generato deve essere il più preciso possibile.

Per usare il servizio di localizzazione standard, è stato necessario configurare un *CLLocationManager* all'interno del VC ShowECG (vedi capitolo 6.6) e chiamare il suo metodo *startUpdatingLocation*. Ogni volta che viene trovata una nuova posizione, il delegato associato al Location Manager la gestisce, attraverso la funzione *didUpdateLocations*. Il VC ShowECG estende la classe CLLocationManagerDelegate, così che sia lui stesso ad essere delegato dal location manager. L'unica funzione del location manager che è stata implementata nell'estensione è appunto la didUpdateLocations. Tale funzione (come si può vedere nel codice 5.1) preleva l'ultimo elemento del vettore che contiene tutte le locazioni che sono state rilevate dal GPS dal momento in cui la startUpdatingLocation è stata chiamata. Ovviamente le posizioni sono rilevate da CoreLocation solo nel caso in cui l'utente abbia **autorizzato la sua condivisione**. La gestione delle autorizzazioni avviene alla prima attivazione. Una volta concessa, l'autorizzazione all'uso della posizione da parte dell'applicazione non può essere revocata dall'interno della stessa (ma solo dalle impostazioni del dispositivo) ma l'utente può decidere grazie alle impostazioni dell'applicazione quando abilitarne la condivisione esternamente all'app (vedi capitolo 6.4). Una volta prelevata la posizione dal vettore, questa deve essere convertita in un formato che sia comprensibile all'utente e al medico, in quanto dovrà essere in seguito allegata alle email ed ai pdf esportati. A questo scopo viene inizializzato un oggetto CLPlacemark, ricavato dalle coordinate della posizione geografica appena ottenuta. CLPlacemark è una classe user-friendly che indica un luogo in termini di indirizzo. I dettagli del placemark sono poi stati inseriti all'interno di una struttura convertibile su richiesta in stringa. La posizione, ora espressa sottoforma di indirizzo, è salvata in una variabile globale del VC, così che, nel momento in cui l'utente cliccherà su un tasto di esportazione, questa sarà pronta all'uso.

Listing 5.1: Funzione di aggiornamento della posizione del Location Delegate

```
1    func locationManager(_ manager: CLLocationManager,
     didUpdateLocations locations: [CLLocation]) {
2        if let p = locations.first{
```

```
3             let geoCoder = CLGeocoder()
4             geoCoder.reverseGeocodeLocation(locations.first!) {
    placemarks, error in
5
6                 guard let placemark = placemarks?.first else {
7                     let errorString = error?.localizedDescription ??
    "Unexpected Error"
8                     print("Unable to reverse geocode the given
    location. Error: \(errorString)")
9                     return
10                }
11
12                let reversedGeoLocation = ReversedGeoLocation(with:
    placemark)
13                self.currentAddress = reversedGeoLocation.
    formattedAddress
14
15            }
16        }
17    }
```

5.6 Libreria Charts

Nello sviluppo dell'app PulsEcg grande attenzione è stata data alla visualizzazione dei segnali ECG e PPG. L'obiettivo è ottenere dei grafici facilmente leggibili, zoomabili a piacimento ed esteticamente piacevoli.

Il primo step è stato selezionare una libreria per grafici potente, gratuita e personalizzabile. La scelta è ricaduta sulla **libreria Charts** (sviluppata da Daniel Cohen Gindi), un'ampia libreria scritta in Swift per traduzione diretta della libreria Java MPAndroidChart (di Philipp Jahoda) per Android. Questo parallelismo tra sistemi operativi è conveniente. Infatti l'applicazione Android PulsEcg traccia i grafici con la libreria MPAndroidChart, perciò qualunque futura modifica nella visualizzazione dei segnali potrà rapidamente essere ottennuta sia su IOs che su Android. Sia per PPG che per ECG sono stati usati dei LineChart.

Il grafico PPG rappresenta insieme sia il segnale Infrarosso che il Rosso. Il grafico ECG è composto dal set di campioni del segnale, e da ulteriori set composti da soli due punti, tanti quanti sono i picchi del segnale.

Il segnale PPG non richiedeva un grafico particolare. Il segnale ECG invece, come illustrato in sezione 1.1.1, è tipicamente tracciato su una carta millimetrata speciale. Per agevolare la lettura del segnale da parte del medico quindi, si è ritenuto di far si che sia nel PDF che in app il grafico presentasse una griglia analoga. Considerando che l'asse x ed y rappresentano rispettivamente i secondi ed i Volt, la griglia è così composta:

1. una linea fine parallela all'asse y deve essere tracciata ogni 40 millisecondi, ed una spessa ogni 200 ms;

2. una linea fine parallela all'asse x deve essere tracciata ogni 0.1 mV, ed una spessa ogni 0.5 mV.

La libreria Charts non permette di ottenere una griglia simile, se non modificandone il codice sorgente. In particolare, nella versione originale, pur potendo definire il numero di linee da visualizzare per ogni asse, questo è vincolato da un tetto massimo pari a 25. Questo è dovuto al fatto che ad ogni linea corrisponde necessariamente una label che ne indichi il valore nell'asse, quindi più di 25 linee genererebbero una sovrapposizione delle labels. In più tutte le linee hanno di default lo stesso colore e lo stesso spessore.

Il codice della libreria è stato quindi modificato, eliminando prima di tutto il vincolo delle 25 righe, così che venisse disegnata una linea ogni 40ms per l'asse x ed una ogni 0.1 mV per l'asse y. Un'altra novità è legata allo spessore: il codice della funzione renderGridLines è stata modificata (vedi Cod. 5.2 e i suoi commenti per la modifica all'asse x) nel senso che ogni 5 righe viene tracciata una linea con spessore più elevato. Le labels sono stampate ogni secondo e ogni 0,1 mV grazie alle modifiche attuate alla funzione drawLabels (vedi Cod. 5.3 e i suoi commenti per la modifica all'asse x). Le modifiche sono state parallele per le classe di render dell'asse x ed asse y.

Listing 5.2: Funzione di stampa della griglia relativa all'asse x.

```
open override func renderGridLines(context: CGContext)
    {
        //...codice intatto...

        for i in stride(from: 0, to: entries.count, by: 1)
        {   //se la riga è multiplo di 5, la stampo spessa
            if i % 5 == 0 {
                context.setStrokeColor(xAxis.gridColor.cgColor)
                context.setLineWidth(0.5)
            }
            //altrimenti la stampo fine
            else {
                context.setStrokeColor(xAxis.gridColor.cgColor)
                context.setLineWidth(xAxis.gridLineWidth)
            }
            position.x = CGFloat(entries[i])
            position.y = position.x
            position = position.applying(valueToPixelMatrix)

            drawGridLine(context: context, x: position.x, y: position
    .y)
        }
```

```
22      }
```

Listing 5.3: Funzione di stampa delle labels sull'asse x

```
1  @objc open func drawLabels(context: CGContext, pos: CGFloat, anchor:
       CGPoint)
2      {
3          //...codice intatto...
4
5          for i in stride(from: 0, to: entries.count, by: 1)
6          {
7              //...codice intatto...
8
9              //ciò che segue, ovvero la stampa, avviene solo al
       compimento di ogni secondo, ovvero ogni 5 righe grandi. Si disegna
        una riga grande ogni 5 righe piccole, quindi sono 25 iterazioni
10             if i % 25 != 0 {
11                 continue
12             }
13             if viewPortHandler.isInBoundsX(position.x)
14             {
15             //il valore in millisecondi dell'asse x viene convertito
       in secondi. La stringa è manipolata in modo da non stampare zeri
       nel caso di secondo netto
16                 var label = String(Double(xAxis.entries[i]) * 2 /
       1000)
17                 let labelns = label as NSString
18                 let regex = "^0*$"
19                 let components = label.components(separatedBy: ".")
20                 if components[1].range(of: regex, options: .
       regularExpression, range: nil, locale: nil) != nil {
21                     label = label.components(separatedBy: ".")[0]
22                 }
23                 label = label + " s"
24
25                 //...codice intatto...
26             }
27         }
28     }
```

Le figure 5.6 e 5.7 mostrano i grafici che si sono ottenuti nell'app PulsEcg.

5.7 API per Import-Export

Uno dei requisiti più importanti dell'applicazione PulsEcg è la possibilità di esportare le acquisizioni, sottoforma di file per l'user (PDF) o di file di gestione interna

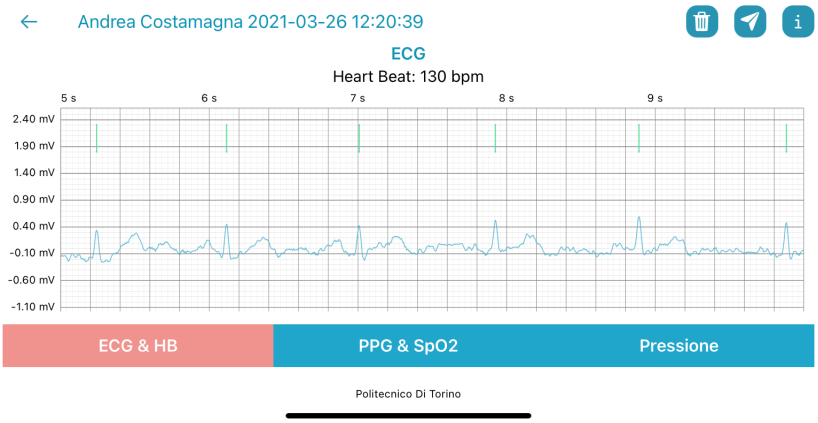


Figura 5.6: Screenshot della schermata con il dettaglio del grafico ECG

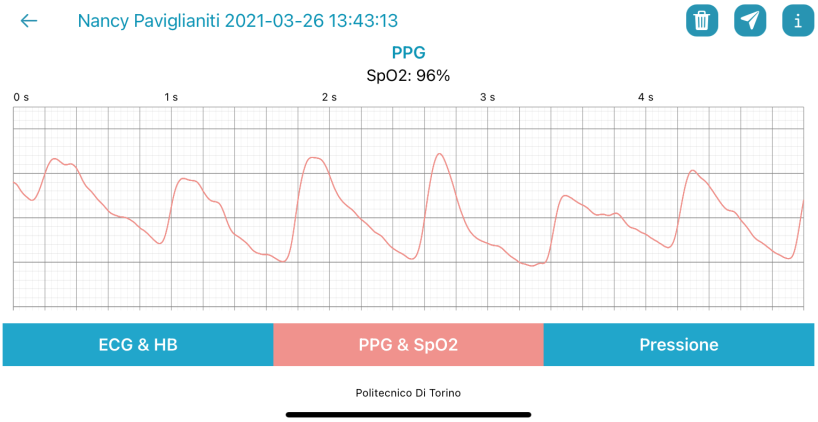


Figura 5.7: Screenshot della schermata con il dettaglio del grafico PPG

(JSON). L'esportazione deve essere possibile su più canali: salvataggio su dispositivo, condivisione tramite mail o applicazioni di messaggistica, apertura attraverso applicazioni di editing.

Attraverso l'applicazione deve inoltre essere possibile importare i file JSON di passate acquisizioni, provenienti potenzialmente da altri dispositivi, anche Android.

Per soddisfare tutte queste richieste, gli strumenti iOS che sono stati selezionati sono il VC Activity per l'esportazione tramite altre app, il VC DocumentPicker per l'importazione dei file JSON e il framework MessageUI per la generazione automatica delle email.

5.7.1 Activity View Controller e PDFKit

iOS fornisce al programmatore il **View Controller Activity** allo scopo di offrire vari servizi standard, come la copia di elementi sulla pasteboard, la pubblicazione di contenuti su siti di social media, l'invio di elementi tramite e-mail o SMS e altro ancora. L'applicazione è responsabile della configurazione, presentazione e chiusura di questo VC [23].

Questo view controller è completamente gestito dal SO. Nel caso dell'esportazione, l'unica preoccupazione è stata dover istanziare il VC, passare il file da esportare come parametro (definendone il tipo) e presentare la sua view sullo schermo. Queste azioni sono realizzate dal VC ShowECG, le cui funzionalità sono descritte nel capitolo 6.6. Il VC appare sottoforma di pop-up, con lo stile tipico del sistema iOS.

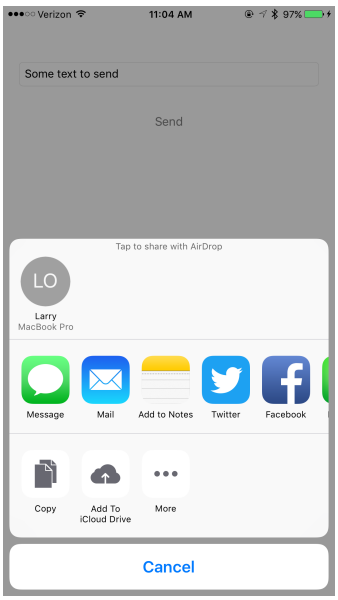


Figura 5.8: Aspetto dell'Activity VC d'esportazione.

Il JSON è generato per mappatura di un oggetto rappresentante la misurazione PulsEcg in JSONObject e per codifica di quest'ultimo in stringa. La stringa è poi scritta su un file, il quale è passato come parametro all'Activity VC per l'esportazione.

Più problematica è stata invece la generazione del PDF. La libreria offerta da iOS per creazione e modifica dei file PDF è chiamata **PDFKit**.

La libreria PDFKit permette di vedere un PDF come un oggetto Rectangle (definito da un punto d'origine, un'altezza ed una larghezza) che può ospitare altri oggetti Rectangle. Gli oggetti in questione, nel caso del PulsEcg, saranno il logo dell'applicazione, alcune informazioni in formato testuale (quali il nome

del paziente, i secondi di acquisizione, l'indirizzo ecc), i grafici ECG e PPG, delle immagini che riassumano i valori di pressione, HB e SpO2. Inoltre, la struttura del PDF deve essere dinamica, in modo che possa crescere in pagine e contenuto in base al numero di secondi d'acquisizione. Il pdf risultante è riportato in Fig. 5.9.

Figura 5.9: PDF generato dall'app PulsEcg, comprendente la posizione geografica

Il codice di generazione del pdf è molto lungo e complesso. Lo possiamo suddividere in zone:

1. **generazione delle immagini da allegare**. Sia i grafici che i valori di HB, SpO2 e pressione sono disegnati nel pdf a partire da immagini. Per quanto riguarda i valori della misurazione, le immagini sono ricavate come segue: viene salvato un riferimento alla schermata correntemente visualizzata; viene mostrata la schermata con il dettaglio delle pressioni e ne viene acquisito uno

screenshot; i valori delle pressioni della view vengono modificati in valori di HB e SpO2, il colore della view viene modificato, e se ne cattura lo screenshot; la view viene ripristinata con l'aspetto originale; viene ripristinata la schermata visualizzata all'inizio. Per i grafici si sono dovute sperimentare diverse strategie. Inizialmente si è provato con la stessa tecnica, acquisendo uno screenshot dei dettagli, ma la quadrettatura dell'ECG rischiava di risultare deformata in base alle dimensioni dello schermo del dispositivo ospitante. Un secondo tentativo è stato quello di disegnare il grafico direttamente sul PDF. La quadrettatura era perfetta, ma la funzione di draw del grafico non permetteva di scegliere la posizione del grafico nel pdf. La soluzione finale è stata quindi quella di disegnare i grafici su di una view non compresa nella gerarchia del VC (quindi mai visualizzata sullo schermo), poi convertita in immagine. Il numero di immagini da generare dipende dai secondi di acquisizione: ogni immagine rappresenta 10 secondi.

2. **scrittura del logo e delle informazioni sulla misurazione**. Una volta inizializzato il contesto del pdf, dichiarando le dimensioni del suo CGRect, in questa fase si dichiarano diversi rettangolini contenenti tutte le informazioni sulla misurazione. Ogni CGRect specifica la posizione dell'oggetto all'interno del pdf. Le informazioni in merito all'acquisizione sono reperite dall'oggetto Misurazione, contenente tutti i dati sull'utente e sui suoi parametri vitali.

3. **scrittura delle immagini**. Con lo stesso meccanismo basato su CGRect usato al punto precedente, vengono stampate le due immagini di Hb, SpO2 e pressioni, una dopo l'altra. Le immagini dei grafici sono invece stampate all'interno di una sezione di codice dinamica, dipendente dal numero di immagini da stampare. Nel caso in cui la durata di acquisizione sia di soli 10s, ci saranno solo due grafici, uno per il ppg ed uno per l'ecg, stampati sulla prima pagina assieme agli altri dati. Se la durata è di 20s, l'ecg sarà nella prima pagina, il ppg nella seconda. Se la durata è maggiore, allora la prima pagina conterrà i dettagli, la seconda sarà dedicata all'ecg e la terza al ppg.

5.7.2 Document Picker View Controller

L'importazione dei file JSON è gestita all'interno del VC Archivio, descritto nel capitolo le cui funzionalità sono descritte nel capitolo **??**. Il Document Picker VC è offerto da iOS per permettere all'utente di selezionare documenti da importare, esportare, aprire o spostare. Oltre alle funzionalità di esportazione, esso supporta due diverse modalità di importazione:

1. Importazione di un documento esterno per copia. L'originale resta invariato.

2. Importazione con modifica del documento esterno [23].

Nell'applicazione PulsEcg si è usata la prima soluzione. L'applicazione, per ragioni di sicurezza, accede durante la sua esecuzione solo ai file contenuti nella sua zona protetta, detta sandbox. Con l'import, è possibile selezionare un file al di fuori della sandbox, aprirlo in modo da poterlo copiare, e poi scrivere tale contenuto in un nuovo file interno alla sandbox.

Il document picker view controller deve essere instanziato e presentato sullo schermo dal programmatore, specificando l'estensione dei file che saranno importabili (in questo caso i JSON). Il VC appare sottoforma di pop-up, e permette di navigare all'interno della memoria del telefono, sfruttando l'applicazione di sistema FileManager. Una volta selezionato un file, si ritorna all'applicazione, con il richiamo del delegato. La classe Archivio è stata estesa in modo da essere essa stessa delegata del DocumentPicker che instanzia. La funzione del delegato è stata implementata per copiare il file selezionato in locale e per aggiornare la view dell'archivio.

5.7.3 MessageUI

Figura 5.10: Aspetto del VC di composizione email generato dall'applicazione PulsEcg con condivisione della posizione. Alla mail è allegato un pdf oppure un json.

Per velocizzare ulteriormente la condivisione di una misurazione PulsEcg, l'applicazione deve fornire un meccanismo di generazione automatica delle email, allegandovi il json o il pdf. Il **framework MessageUI** fornisce dei VC specializzati per la presentazione di interfacce con cui comporre messaggi di testo, e-mail e SMS. Queste interfacce permettono di consegnare dei messaggi senza richiedere all'utente di abbandonare l'app corrente. Prima di visualizzare il VC, è possibile configurarlo in modo che sia già precompilato in alcuni campi. Una volta presentato,

l'utente ha la possibilità di personalizzare i contenuti prima di inviare o annullare il messaggio. L'azione dell'utente viene poi accolta e gestita dal delegate. [23].

All'interno della classe ShowECG VC è stata scritta la funzione inviaEmail che ha il compito di configurare e presentare sullo schermo il MFMailComposeViewController. Prima di tutto la funzione controlla se il servizio mail è disponibile e internet è acceso. Poi reperisce tutte le informazioni utili alla compilazione dell'e-mail: i dati salvati dall'utente nella configurazione dell'applicazione come nome, cognome e mail del medico; i dati reperiti da file quali la data e l'orario; la posizione precalcolata come spiegato nel capitolo 5.5. La posizione verrà ovviamente allegata solo se il paziente ne ha abilitato la condivisione (questo dato è reperito tra quelli di configurazione).

Infine, all'email deve essere allegato il file di misurazione. Se è stato scelto di allegare un pdf, allora verrà chiamata la funzione per generarlo (trattata al capitolo 5.7.1), altrimenti si esporterà il file json contenuto nel file system. Più avanti sarà spiegato come in alcune situazioni il file non si trovi nel filesystem. In tal caso il file è generato in questo momento, salvato in cache ed esportato. In questo modo il file verrà eliminato dalla cache alla chiusura dell'applicazione, senza essere mai scritto in memoria non volatile.

5.8 Libreria TensorFlowLite

La rete neurale per ricavare la pressione arteriosa a partire dai segnali ECG e PPG è stata creata, allenata e validata su Google Colaboratory in Python, sfruttando TensorFlow. La rete deve però essere utilizzabile per predire i valori di pressione ad ogni acquisizione da dispositivo, quindi deve essere salvata localmente sul dispositivo. Il modello allenato della rete è stato quindi esportato ed integrato nel codice sorgente dell'applicazione.

La **libreria TensorFlowLite**, realizzata parallelamente sia in Swift che in Android, permette di importare modelli di rete neurale pre-allenati realizzati mediante TensorFlow all'interno di una applicazione mobile. In particolare, un modello può essere esportato in codice Python in diversi formati: file checkpoint, SavedModel e FrozenGraph. Ciascuno di questi può essere poi convertito in un file con estensione tflite, ovvero compatibile con la libreria TensorFlowLite. Nel nostro caso, il modello è stato prima salvato in formato Saved Model e poi convertito in TFLite (Cod. 5.4). Tale file deve essere incorporato tra gli asset dell'applicazione mobile, e può poi essere aperto e riconvertito in modello in linguaggio nativo d'ambiente ospitante grazie ad un interprete.

Listing 5.4: Codice Python d'esportazione del file TensorFlowLite

```
1  #salvataggio del modello in formato SavedModel
2  path = "/content/drive/My Drive/Colab Notebooks/saved_model"
```

```
3     os.mkdir(path)
4     tf.saved_model.save(model, path)
5     #conversione del SavedModel in TFLite file
6     converter = tf.compat.v1.lite.TFLiteConverter.from_saved_model('/
      content/drive/My Drive/Colab Notebooks/saved_model/')
7     converter.target_spec.supported_ops = [
8                                             tf.lite.OpsSet.
      TFLITE_BUILTINS,
9                                             tf.lite.OpsSet.
      SELECT_TF_OPS
10                                             ]
11    converter.allow_custom_ops=True
12    tflite_model = converter.convert()
13    #salvataggio del file
14    path = "/content/drive/My Drive/Colab Notebooks/
      ModelTensorflowLite"
15    os.mkdir(path)
16    with open('/content/drive//Colab Notebooks/
      ModelTensorflowLite/model.tflite', 'wb') as f:
17       f.write(tflite_model)
```

L'interprete Swift della libreria TensorFlowLite, ricevuto il file tflite come argomento, traduce il modello automaticamente e alloca la memoria per i tensori di input ed output. A questo punto, esso è pronto a ricevere un tensore di input per cui restituire un tensore di output. Il programmatore deve occuparsi della pre-processazione dei dati, normalizzandoli e suddividendoli in arrays di dimensione pari alla capacità di un batch, così come definito alla creazione della rete. Le funzioni offerte dalla libreria TensorFlowLite sono la *copy* per caricare il tensore di input, la *invoke* per calcolare l'output attraverso il modello e la *output* per ottenere il risultato calcolato a seguito dell'invocazione.

L'interprete e le variabili necessarie alla rete per i suoi calcoli sono stati impacchettati in una classe Singleton (per cui è instanziabile un unico oggetto, eventualmente referenziato in più punti) chiamata NeuralNetworkManager, così che l'inferenza del modello, essendo un'operazione costosa, sia realizzata una volta sola (Cod. 5.5). La classe offre i metodi pubblici per:

1. ottenere l'istanza del NetworkManager, munita dell'interprete che ha caricato il modello;

2. lanciare il calcolo della pressione sistolica e diastolica per una coppia di segnali PPG ed ECG. Dentro questa funzione (Cod. 7.18) si nasconde il preprocessing, il richiamo della rete per ogni batch e il post processing del risultato (calibrazione e calcolo media);

3. restituire i valori finali ottenuti.

Listing 5.5: Costruttore dell'unica istanza della classe NeuralNatworkManager

```
1     init(){
2     //importazione del modello da file, inizializzazione dell'
      interprete e allocazione della memoria per i tensori
3         modelPath = Bundle.main.path(forResource: "model", ofType: "
      tflite")
4         do{
5         interpreter = try Interpreter(modelPath: modelPath!)
6         try interpreter!.allocateTensors()
7         }catch{
8             interpreter=nil
9             print("Errore nell'importazione del modello")
10        }
11    //Inizializzazione delle variabili necessarie per i calcoli
12        pressSis = nil
13        pressDia = nil
14        pressioniSis = []
15        pressioniDia = []
16        ECGBatches = [:]
17        PPG_REDBatches = [:]
18        for i in 0 ..< NeuralNetworkConstants.nBatches {
19            ECGBatches[i] = []
20            PPG_REDBatches[i] = []
21        }
22    }
```

Capitolo 6

Front-End

Se il precedente capitolo voleva fornire una panoramica sugli strumenti necessari alla stesura del codice dell'applicazione, soffermandosi anche su come ne sia stata implementata la personalizzazione, in questo capitolo affronteremo il vero e proprio processo di realizzazione dell'app PulsEcg.

La fase di **implementazione**, essendo molto ampia, è stata suddivisa in due capitoli: uno sul back-end ed uno sul front-end. I termini inglesi front-end e back-end denotano, rispettivamente, la parte visibile all'utente di un programma e con cui egli può interagire (tipicamente un'interfaccia utente) e la parte che permette l'effettivo funzionamento di queste interazioni. Il front-end, nella sua accezione più generale, è responsabile dell'acquisizione dei dati di ingresso e della loro elaborazione con modalità conformi a specifiche predefinite e invarianti, tali da rendere i dati stessi utilizzabili dal back-end [30].

In questo capitolo il focus è posto sul **front-end**. Poichè il front-end riguarda in gran parte l'interfaccia grafica, una grande attenzione sarà posta sulla User Experience. Le scelte prese a riguardo derivano soprattutto dall'analisi dei requirements non funzionali di usabilità e dallo studio delle Storie e Persone, ovvero del target di pubblico che userà l'applicazione. Dopo una panoramica su questi aspetti, il capitolo si articola in più parti: la prima descrive la navigazione dell'utente tra view controllers, le altre analizzano nel dettaglio ogni VC dell'applicazione.

6.1 User Experience

Con il rapido sviluppo della scienza e della tecnologia, e in particolare con l'ascesa del settore Internet e dei prodotti software, le persone si sono abituate sempre di più ad utilizzare servizi di questo tipo, tra cui gli applicativi mobili. I prodotti per terminali mobili come smartphone e tablet sono quindi diventati sempre più importanti nella vita di tutti. Ridurre il costo di apprendimento degli utenti quando

utilizzano dispositivi diversi e consentire loro di avere un'esperienza operativa senza interruzioni nel comportamento cross-screen, sono questioni importanti che i progettisti devono affrontare nella progettazione interattiva multischermo. La tecnologia non rappresenta più l'unico l'elemento di competizione dei prodotti. L'altro parametro fondamentale è quello dell'esperienza dell'utente. I dispositivi mobili sono limitati dalle dimensioni dello schermo, il che comporta molte sfide per la progettazione della User Experience [31].

La **User Experience (UX)** è comunemente descritta come la qualità olistica dell'interazione tra un utente e un sistema interattivo. I ricercatori e professionisti di UX concordano sui tre pilastri che influenzano l'UX: l'utente, il sistema e il contesto. Hassenzahl e Tractin-sky definiscono l'UX come conseguenza: dello stato interno ed esterno di un utente; delle caratteristiche del sistema progettato (ad esempio, complessità, scopo, usabilità, funzionalità, ecc.); del contesto (o dell'ambiente) all'interno del quale avviene l'interazione (es. assetto organizzativo / sociale, significatività dell'attività, volontarietà d'uso, ecc.) [32].

L'obiettivo dello studio dell'UX è quello di migliorare l'interazione tra esseri umani e sistemi informatici, garantendo che i sistemi offrano usabilità e soddisfazione. riuscire a realizzare interfacce che assistano gli utenti nell'esecuzione delle attività in modo ottimale è una sfida importante per tutti i progettisti. Come detto precedentemente, quando gli utenti interagiscono con i sistemi informatici molti fattori possono influenzare le prestazioni complessive. L'utente condiziona l'UX con tre fattori: interni, esterni e di stress. I fattori interni si riferiscono a condizioni fisiche e mentali che sono intrinseche o introdotte dagli esseri umani come operatori, come l'età, la capacità visiva e la motivazione. I fattori esterni si riferiscono ad elementi di contorno, quali l'ambiente circostante. I fattori di stress sono principalmente quelle pressioni psicologiche o fisiologiche che influenzano il processo decisionale e l'azione dell'operatore direttamente o indirettamente, come carico di lavoro intenso, ritmo di lavoro elevato, dolore, esaurimento, stress a lungo termine, ecc. Lo studio dei fattori influenti fornisce implicazioni utili per gli ingegneri informatici al fine di ottimizzare il design dell'interazione [33].

6.1.1 Profilo Utente

Lo studio delle Storie e Persone avvenuto nel capitolo 4.2.1 ci permette di delineare il profilo del tipico utente dell'applicazione PulsEcg. Un **Profilo Utente** rappresenta il target di un certo gruppo di utenti sulla base di informazioni collettive sui dati mentali, fisici e demografici dei suoi membri. Lo scopo finale dei profili utente è di aiutare i progettisti a riconoscere o conoscere l'utente presentando loro una descrizione degli attributi che lo descrivono: il sesso, l'età, il livello di istruzione, l'atteggiamento, le esigenze tecniche e il livello di abilità [33].

Trattandosi di una applicazione rivolta sia agli uomini che alle donne, gli aspetti della profilazione che son stati principalmente tenuti in conto sono stati:

1. **L'esperienza degli utenti** : l'abilità o la conoscenza che essi hanno acquisito mediante la pratica d'uso di un certo tipo di sistema. Può essere vista come una conseguenza della capacità degli utenti di adattarsi agli ambienti fisici e sociali. Gli utenti sono classificati in diverse categorie lungo la scala dell'esperienza: utenti inesperti, che conoscono il compito ma hanno poca o nessuna conoscenza del sistema; utenti intermittenti consapevoli, che conoscono il compito ma a causa dell'uso poco frequente, possono avere difficoltà a ricordare la conoscenza sintattica di come realizzare i loro obiettivi; utenti abituali esperti, che hanno una profonda conoscenza delle attività e degli obiettivi correlati e delle azioni necessarie per raggiungere gli obiettivi [33].

2. **L'età** : è una questione importante da tenere a mente. È stato riscontrato che l'invecchiamento provoca un declino, ad esempio, della fisiologia e neurofisiologia dell'occhio, dei fattori fisici, sensoriali e cognitivi, e comporta un rallentamento in quasi tutti i compiti che stressano prestazioni rapide. È generalmente accettato che tali declini accelerino dopo i quarantacinque anni. È stato inoltre riscontrato che le persone anziane hanno maggiori problemi nell'uso dei telefoni cellulari. Nell'indagare sulle caratteristiche di un telefono cellulare adatto all'invecchiamento, Kurniawan ha scoperto che gli anziani hanno paura delle conseguenze dell'utilizzo di una tecnologia sconosciuta [33].

6.1.2 Design dell'interfaccia

L'applicazione PulsEcg è indirizzata principalmente a **persone anziane**, in quanto queste sono più facilmente soggette a problemi cardiaci. All'anzianità di solito coincide una **bassa familiarità con il mondo smart**. Gli anziani infatti hanno vissuto gran parte della loro vita senza cellulari e computer, perciò hanno più difficoltà a sentirsi a loro agio nel loro utilizzo, anche per la paura di cui sopra. Per questo motivo, i principali concetti che hanno guidato la creazione della UI sono stati: fiducia, minimalismo e automatismo.

Vista la diffidenza degli anziani verso le nuove tecnologie, la GUI è stata progettata nel tentativo di trasmettere un senso di **fiducia**. Ciò è stato possibile grazie alla teoria dei colori, e del loro legame con l'emotività. Come mostrato in figura 6.1, il colore associato alla fiducia è il blu. La fiducia può essere rafforzata con l'uso dell'arancione, che trasmette amichevolezza. Il verde inoltre esprime tranquillità, e può essere anche un riferimento implicito alla salute [34] (l'applicazione ha scopi medici, quindi il colore ne comunica anche la funzione). Un'interfaccia utente solitamente è caratterizzata da un colore primario, da alcune sfumature più scure e

Figura 6.1: Corrispondenza colore-emozione[34]

più chiare di tale colore, e da uno/due colori d'accento, contrastanti con i precedenti, per concentrare l'attenzione su di un punto dello schermo. Compatibilmente con la teoria dell'armonicità tra colori (strumento Adobe Color [35]), si è cercato di rispettare i principi emozione-colore descritti, scegliendo un blu petrolio come colore primario (e diverse sue sfumature secondarie). L'arancione corallo e il verde acqua sono stati invece usati per i punti di accento più o meno rilevante. Oltre a questi colori, sono stati scelti anche il nero per gran parte del testo (il più leggibile) e il bianco per lo sfondo. La palette è raffigurata in Fig. 6.2.

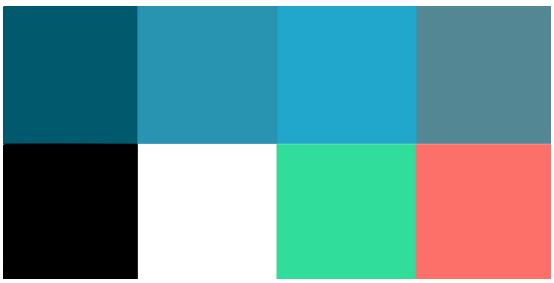


Figura 6.2: Palette di colori scelti per la GUI dell'app PulsEcg

Il secondo principio della Graphic User Interface (GUI) dell'applicazione PulsECG è il **minimalismo**. Questo principio deve essere applicato sia all'estetica che alle funzionalità, eliminando del tutto la ridondanza. Minimalismo nell'estetica significa ridurre al minimo il numero di informazioni mostrate sullo schermo. I pochi elementi devono essere ben evidenziati, perciò si è data importanza allo spazio. Gli spazi vuoti hanno tanta rilevanza quanto quelli pieni. Questi permettono di evitare che il cervello si sforzi inutilmente nel cercare di suddividere l'area che gli occhi stanno osservano in sezioni funzionali, separandole a priori con ampi spazi. Ciò permette all'utente anziano, con probabili problemi di vista, di concentrarsi su pochi e ben definiti elementi. Dal punto di vista funzionale, un'interfaccia minimalista offre poche pagine, poche azioni per pagina (non più di due o tre bottoni) e una navigazione rapida tra le schermate. L'homepage dell'applicazione (Fig. 6.3) permette di evidenziare tutti questi aspetti: ampi spazi che separano le azioni; pochi bottoni; poche informazioni mostrate (solo quelle essenziali).

L'**automatismo** di certe azioni, il forzare certi passaggi o certi formati di visualizzazione, può liberare l'utente da alcuni compiti di cui occuparsi. Ad esempio, nella navigazione, è stato scelto di forzare il ritorno all'homepage dopo alcune azioni, in modo da non disorientare l'utente come fosse in un labirinto. Un'altra scelta è stata quella di forzare l'orientamento delle schermate. Laddove un utente esperto può trovare giovamento nel poter ruotare la schermata a piacimento, al contrario un inesperto potrebbe essere infastidito dalla rotazione continua ad ogni variazione dell'angolazione del dispositivo. Nell'applicazione PulsEcg tutte le schermate sono verticali, escluso il dettaglio della misurazione, in cui è forzato il landscape per visualizzare meglio i grafici. Altre comodità di questo tipo sono il salvataggio automatico, la condivisione della posizione automatica, l'autogenerazione della email, la connessione automatica al dispositivo associato ecc.

Infine, si è tenuto conto dell'**adattabilità dell'interfaccia ai vari schermi**. Infatti, per anni iOS è stato esente da questo problema, perchè i modelli di iPhone e iPad erano davvero pochi. Allo scopo di soddisfare più utenti però, Apple ha introdotto numerose varianti nelle dimensioni dello schermo: modelli plus, modelli max, modelli mini ecc. Il layout di ogni schermata è stato quindi costruito in maniera adattabile, costruendo dei vincoli basati sui contorni dello schermo, e permettendo alle view di crescere liberamente al suo interno, a seconda dalle dimensioni.

6.1.3 Logo e icone

Nell'ottica di rimanere in linea con il principio di semplicità e usabilità, le immagini e le icone dell'applicazione sono minimaliste e rimandano immediatamente alla

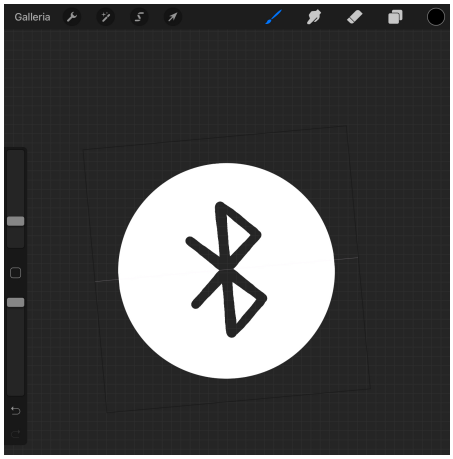
Figura 6.3: Homepage dell'applicazione PulsEcg

funzione del bottone che decorano. Tutte le immagini sono bianche ed in 2D. Il logo (Fig. 6.4a) è stato reperito su internet, come risorsa vettoriale gratuita ad uso libero [36]. Le icone del menu e l'animazione dell'ECG invece sono originali, disegnate a mano con il software per iPad Procreate (Fig. 6.4b) e rifinite con Adobe Photoshop (software per pc). Le icone più piccole sono invece quelle offerte da iOS. Si è preferito mantenere le icone di sistema sia per motivi di uniformità estetica con il resto delle applicazioni, sia perchè l'utente è abituato a riconoscerle.

6.2 Navigazione

La navigazione dell'applicazione, così come la GUI di ogni schermata, è stata implementata con l'Interface Builder di XCode ed è contenuta nel file **StoryBoard**

(a) Logo dell'applicazione PulsEcg [36]



(b) Interfaccia di Procreate, software usato per la progettazione delle icone e delle animazioni su Procreate

Figura 6.4: Logo e piattaforma di progettazione delle icone dell'app PulsEcg

(Fig. 6.5). Le entità coinvolte nella navigazione sono i View Controller, ovvero le entità che rappresentano una schermata, e a cui è quindi associata un'interfaccia grafica che copre tutto lo schermo.

Come accennato nei punti precedenti, il numero di View Controller e la navigazione tra di essi sono stati studiati per essere estremamente semplici e rapidi. La figura 6.6 ne riassume il meccanismo. L'Homepage VC offre 4 bottoni. Il click di ogni bottone genera una transizione ad un nuovo VC, in particolare al:

1. **ConnectBluetoothDevice**. Si tratta del VC usato dall'utente per associare un nuovo dispositivo bluetooth. Questo VC è una foglia di navigazione, ovvero può solo essere dismesso. A quel punto verrà nuovamente mostrata la schermata d'origine (l'homepage oppure GeneralOptions VC). Tutti i segue che portano a questo VC sono di tipo present modally senza passaggio di parametri. La modalità present modally prevede che la nuova view copra completamente quella del VC precedente (quindi è necessario creare un bottone di back che dismetta il VC attualmente visualizzato).

2. **GeneralOptions**, ovvero il VC per la configurazione delle preferenze utente. Il segue tra homepage e info è di tipo show, ovvero non copre completamente il VC precedente, e non necessita di un backbutton, perchè è sufficiente scorrere con il dito verso il basso, per chiuderlo e visualizzare nuovamente l'homepage. Tra i due VC non c'è alcuno scambio di informazioni. Questo VC offre la possibilità di transitare ancora, con un segue verso il VC che abbiamo

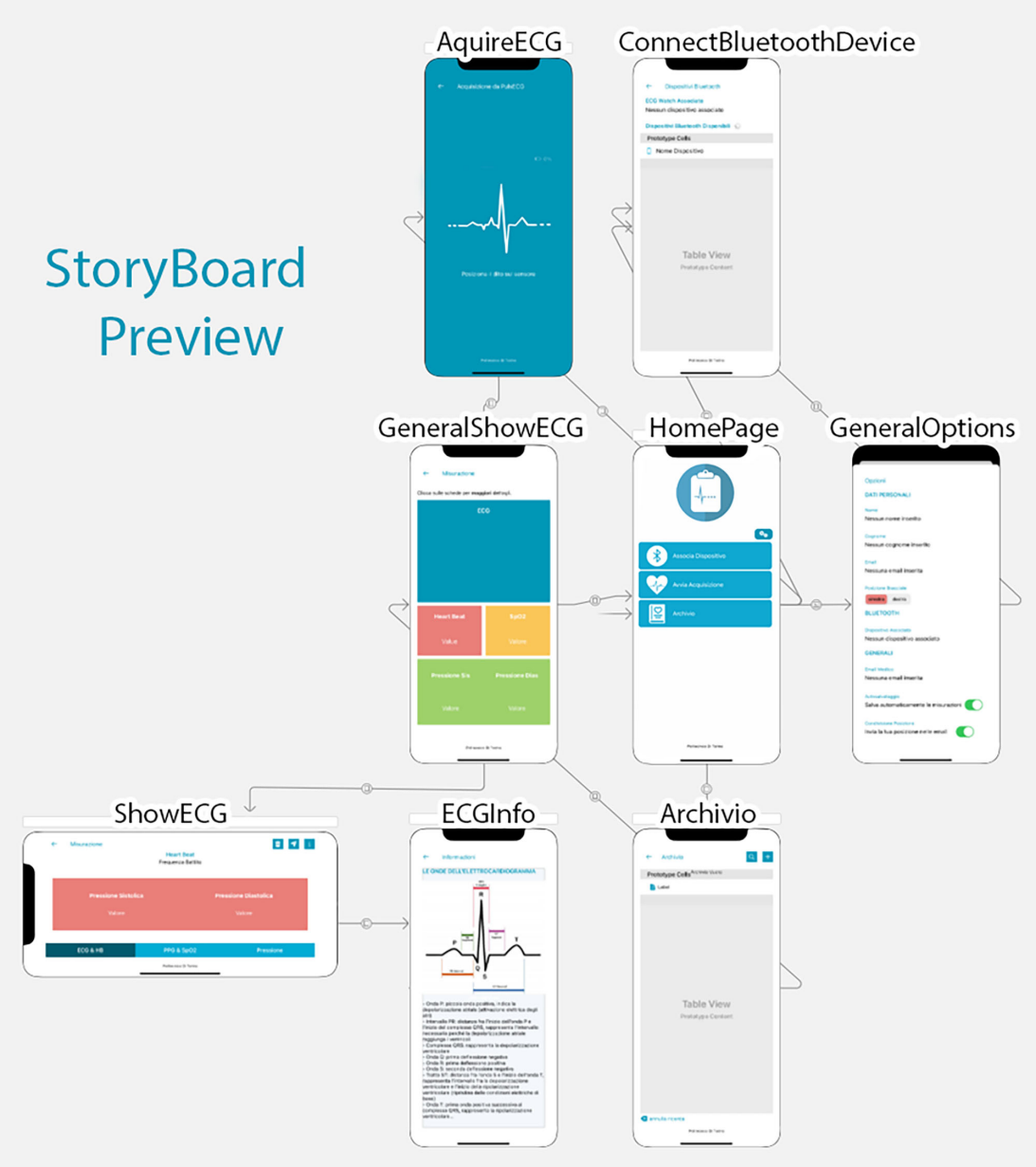


Figura 6.5: Anteprima dello storyboard dell'applicazione, raffigurante i View Controllers e la navigazione tra di essi

presentato al punto 1 (in tal caso per tornare all'homepage bisognerà fare due dismiss consevutive, una con backbutton e una con pull verso il basso).

3. **AquireECG** - è il VC che permette di acquisire. Viene presentato in present modally mode (quindi ha un back button) e non riceve dati in input

(nessun override della prepare). Questo VC transita automaticamente con un altro segue al GeneralShowECG VC quando l'acquisizione è terminata, trasferendo i parametri della misurazione grazie alla funzione prepare. Se il GeneralShowECG è stato raggiunto dall'AquireECG e viene poi dismesso, sarà automaticamente chiuso anche il VC di acquisizione, riportando l'utente alla home.

4. **Archivio** - si tratta del VC adibito alla consultazione dello storico delle acquisizioni. Anche in questo caso il segue è present modally privo di parametri. Al click su un qualunque file dell'elenco, avviene un segue present modally verso il GeneralShowECG, passandogli il nome e il path del file come parametri.

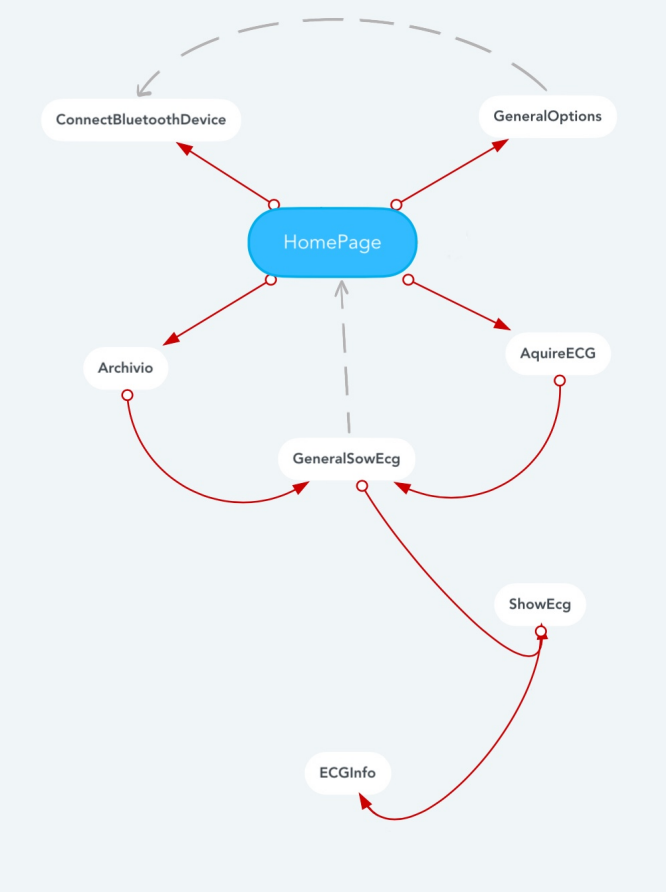


Figura 6.6: Schema di navigazione tra View Controllers

Archivio e AquireECG portano al **GeneralShowECG VC**. Questo a sua volta permette, con un click su qualunque sua scheda, di transitare (sempre in present modally con passaggio di parametri) allo **ShowECG VC**. A fronte di

una cancellazione del file, il cui input è dato nello ShowECG, viene dismesso sia quest'ultimo che il precedente View Controller, riportando automaticamente all'HomePage.

6.3 Schermata principale e impostazioni

Le prime schermate con cui l'utente entra in contatto, prendendo familiarità con l'applicazione, sono la Homepage (Fig. 6.7a) e la schermata di configurazione (6.7b).

1. **Homepage** è la pagina che permette di muoversi tra le varie funzionalità offerte dall'applicazione. In altre parole, si tratta del menu di PulsEcg App. La view presentata sullo schermo è composta dal logo dell'applicazione, da una serie di bottoni impilati che permettono di spostarsi ad altre schermate: la gestione bluetooth; l'acquisizione tramite bracciale; la consultazione dell'archivio. In alto a destra è poi presente un bottoncino per aprire le impostazioni, ovvero l'altra schermata di cui parliamo in questa sezione.

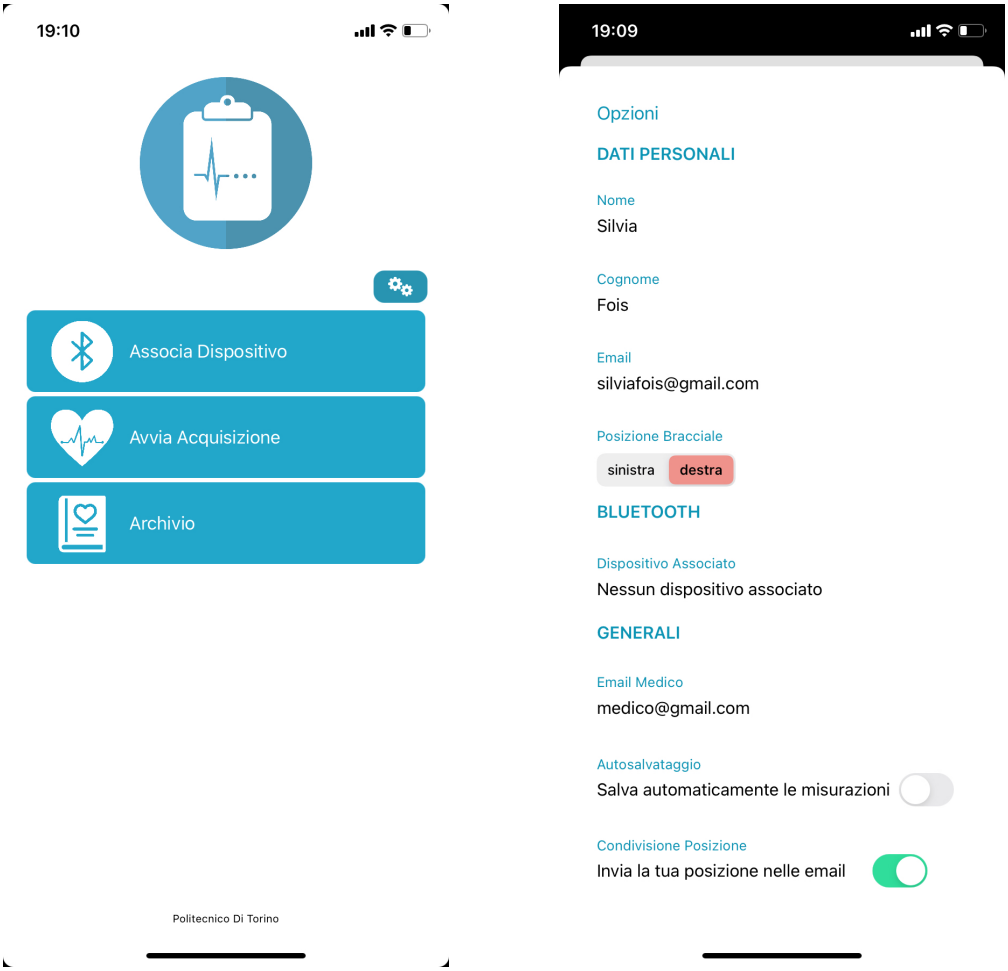
2. La pagina di **configurazione delle preferenze utente** è estremamente semplice, utilizzata dall'utente per inserire i propri dati, quelli del medico, ed eventualmente associare un dispositivo. Inoltre, è possibile decidere se condividere la propria posizione, se salvare automaticamente i file acquisiti e configurare il braccio d'acquisizione preferito.

Le due schermate sono implementate grazie a due View Controller, chiamati nel codice HomePage e GeneralOptions. La transizione tra i due avviene mediante uno show segue. Quando si usa un segue di tipo *show*, la view del nuovo VC non copre completamente quella del precedente, perciò non è necessario creare ed includere un back-button con implementazione esplicita dell'eliminazione del VC, perchè è sufficiente tirare la view verso il basso con un dito.

6.3.1 HomePage VC

La view di questo ViewController è composta da una ImageView per il logo, da un UIButton per le opzioni e da una UITableView per il menu 6.8.

Il menu è stato realizzato con una tabella e non con dei bottoni, per una questione estetica, in modo da poter posizionare le icone come desiderato e da poter arrotondare i bordi delle celle. Un oggetto **UITableView** deve essere gestito esplicitamente dal programmatore mediante codice (come avviene per i Recycler View in Android). Per utilizzare questo oggetto è necessario definire sia il layout che la logica di ogni item, ovvero di ogni oggetto TableViewCell. Per questo motivo è stata definita una classe di nome HomePageMenuOption, contenente un'immagine



(a) HomePage VC **(b)** GeneralOptions VC

Figura 6.7: UI dei View Controller HomePage e GeneralOptions

ed una descrizione. La tabella viene popolata alla creazione del VC, a partire da un vettore contenente gli oggetti HomePageMenuOption da inserire (Cod. 6.1).

Listing 6.1: Popolamento della UITabelView dell'HomePage VC.

```
let option1 = HomePageMenuOption(titolo: "Associa Dispositivo",im: #
    imageLiteral(resourceName: "bluetooth"))
        let option2 = HomePageMenuOption(titolo: "Avvia Acquisizione"
    ,im: #imageLiteral(resourceName: "cuore_acquisizione"))
        let option3 = HomePageMenuOption(titolo: "Archivio",im: #
    imageLiteral(resourceName: "archivio"))
        menuOptions.append(option1)
        menuOptions.append(option2)
        menuOptions.append(option3)
```

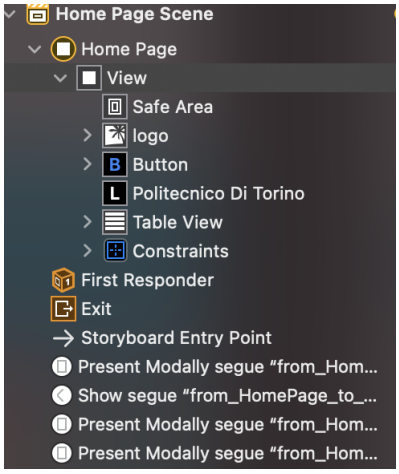


Figura 6.8: Struttura delle views e segue connessi all'HomePage VC

La classe HomePage VC è stata estesa in modo da poter essere la delegata della tabella. L'oggetto **UITableViewDelegate** offre tre fondamentali metodi, che verranno chiamati dal sistema a fronte di certi eventi. I metodi sono stati personalizzati in questo modo:

1. **numberOfRowsInSection** - restituisce la dimensione della tabella. Ovviamente deve restituire il numero di elementi nel vettore menuOptions contenente gli oggetti HomePageMenuOption.

2. **cellForRowAt** - viene chiamata dall'SO per un numero di volte successive pari alla dimensione ritornata dal precedente metodo, ovvero nel nostro caso pari alla dimensione del vettore menuOptions. Questa funzione associa un elemento del vettore ad ogni cella.

3. **didSelectRowAt** - viene chiamato quando l'utente clicca su una cella della tabella. Si tratta della funzione più importante perchè si occupa di trasferire il controllo ad un nuovo VC, previa verifica delle condizioni d'accesso, segnalandolo all'utente in caso negativo. La sua implementazione è consultabile in 6.2. La funzione riceve di default l'indice della cella come parametro, perciò le varie logiche sono separate da uno switch, che differenzia le azioni da eseguire in base al bottone premuto. Il primo bottone permette all'utente di gestire l'associazione di un dispositivo hardware, perciò la funzione deve controllare se il bluetooth sia attivo: in caso positivo, performa un segue verso ConnectBluetoothDevice VC; in caso negativo, avvisa l'utente con un alert, offrendogli una scorciatoia alle impostazioni di sistema. Il secondo caso deve permettere di avviare una acquisizione. La funzione deve fare tre controlli: se il bluetooth sia acceso, se un dispositivo sia stato associato, se il dispositivo

associato sia attualmente connesso. In caso di verifica positiva per il tutto, allora trasferisce il controllo all'AquireVC, mentre in caso negativo, mostra un alert munito di una scorciatoia: alle opzioni di sistema nel primo caso; al ConnectBluetoothDevice VC nella seconda. La terza cella invece permette una transizione all'Archivio VC. Questa transizione è sempre concessa. Tutti i segue del menu sono realizzati in modalità *present modally*, ovvero il nuovo VC copre completamente il vecchio, e quindi servirà creare un bottone che realizzi il dismiss per permettere all'utente di tornare indietro.

Listing 6.2: Funzione didSelectRowAt personalizzata per la gestione del menu.

```
1  func tableView(_ tableView: UITableView, didSelectRowAt
   indexPath: IndexPath){
2      switch indexPath.row{
3      case 0:
4          if BleManager.sharedInstance.isOn() == false {
5          let alertController = UIAlertController(title: "
   Bluetooth Spento", message: "Per avviare una misurazione
   accendi il bluetooth", preferredStyle: UIAlertController.Style
   .alert)
6          alertController.addAction(UIAlertAction(title: "Vai a
    Impostazioni", style: UIAlertAction.Style.default, handler: {
    action in
7                  self.apriImpostazioni()
8              }))
9          alertController.addAction(UIAlertAction(title: "
   Annulla", style: UIAlertAction.Style.default, handler: nil))
10         alertController.view.tintColor = UIColor(red: 12/255,
    green: 149/255, blue: 182/255, alpha: 1)
11         present(alertController, animated: true, completion:
   nil)
12         }
13
14         performSegue(withIdentifier: "
   from_HomePage_to_ConnectBluetoothDevice", sender: self)
15         break;
16     case 1:
17         let dev = UserDefaults.standard.object(forKey: "
   BLEDevice") as? String ?? String()
18         if !dev.isEmpty && dev != "" && BleManager.
   sharedInstance.isConnectedToDevice() &&
19             BleManager.sharedInstance.getConnectedDevice().
   name == dev && BleManager.sharedInstance.isOn() {
20             performSegue(withIdentifier: "
   from_HomePage_to_AquireECG", sender: self)
21         }
22         else{
23             var alertController : UIAlertController?
```

```
24                if dev.isEmpty || dev == "" || BleManager.
    sharedInstance.isConnectedToDevice()==false {
25                    alertController = UIAlertController(title: "
    PulsECG non associato", message: "Per avviare una misurazione
    associa un orologio.", preferredStyle: UIAlertController.Style
    .alert)
26                    alertController!.addAction(UIAlertAction(
    title: "Associa", style: UIAlertAction.Style.default, handler:
     { action in
27                        self.apriconnessione()
28                    }))
29                }
30                else {
31                    alertController = UIAlertController(title: "
    Bluetooth Spento", message: "Accendi il bluetooth per
    acquisire da dispositivo.", preferredStyle: UIAlertController.
    Style.alert)
32                    alertController!.addAction(UIAlertAction(
    title: "Vai a Impostazioni", style: UIAlertAction.Style.
    default, handler: { action in
33                        self.apriImpostazioni()
34                    }))
35                }
36                alertController!.addAction(UIAlertAction(title: "
    Annulla", style: UIAlertAction.Style.default, handler: nil))
37                alertController!.view.tintColor = UIColor(red:
    12/255, green: 149/255, blue: 182/255, alpha: 1)
38                present(alertController!, animated: true,
    completion: nil)
39            }
40            break;
41        case 2:
42            performSegue(withIdentifier: "
    from_HomePage_to_Archivio", sender: self)
43            break;
44        default:
45            break;
46        }
47
48    }
49
50
```

Per finire, la classe HomePage VC è anche responsabile dell'inizializzazione del bluetooth manager. Si tratta di una classe di gestione della connessione bluetooth che opera in background (verrà spiegata approfonditamente nel capitolo 7.1). Il manager viene attivato, e la ricerca di dispositivi bluetooth ha inizio. Nel caso in cui in precedenza fosse già stato associato un dispositivo, e questo sia tuttora

disponibile, la connessione avviene senza spostarsi dall'homepage e un meccanismo di notifica asincrono permette alla view di aggiornarsi, mostrando l'associazione sul primo bottone.

6.3.2 GeneralOptions VC

Il GeneralOptions VC è un View Controller composto **Stack View (SV)** verticali ed orizzontali annidati.

Lo SV è un oggetto che funge da contenitore e gestore di layout di altre view. Esso fornisce un'interfaccia semplificata per disporre una raccolta di viste impilate in una colonna o in una riga. Inoltre, consente di sfruttare la potenza del layout automatico, creando interfacce utente in grado di adattarsi dinamicamente all'orientamento del dispositivo, alle dimensioni dello schermo e a qualsiasi modifica nello spazio disponibile [23].

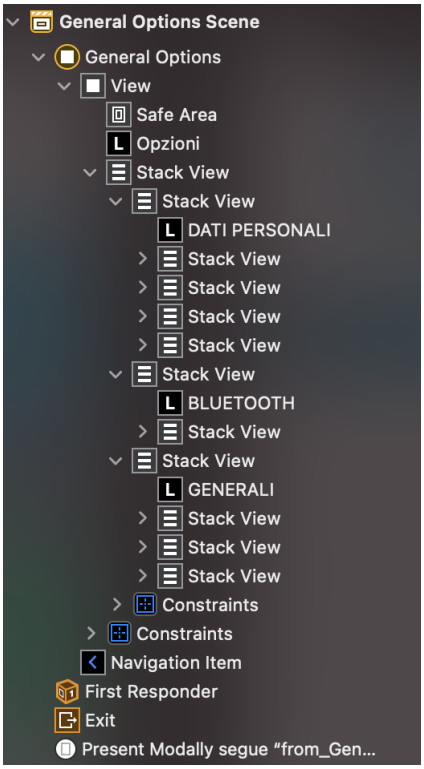


Figura 6.9: Struttura delle views e segue connessi al GeneralOptions VC

Come si può osservare nella figura 6.9, gli SV che compongono il VC sono disposti per livelli. Il primo livello serve per suddividere le opzioni in macrocategorie: dati personali, bluetooth e impostazioni dell'applicazione. Ognuno di questi SV contiene a sua volta una UILabel (view contenente testo) per mostrare il nome

della categoria, ed uno o più SV verticali composti da una UILabel con il nome dell'opzione, nonchè da un oggetto interattivo, che può essere un bottone, un segmented control (per scegliere tra due opzioni) oppure uno switch (per accendere o spegnere una funzionalità). Cliccando sui bottoni è possibile modificare il valore dei parametri testuali.

Ogni volta che l'utente modifica il valore/lo stato degli oggetti interattivi, scatena l'esecuzione dell'action ad essi connessa. L'action è l'handler dell'evento scatenato dall'interazione. All'interno degli handler, le opzioni inserite dall'utente vengono salvate in un database offerto dal sistema operativo. La classe **UserDefaults** offre un'interfaccia per accedere a tale database, in cui sono archiviate in modo persistente delle coppie chiave-valore. UserDefaults memorizza le informazioni nella cache per evitare di dover aprire il database ogni volta che un'applicazione ha bisogno di un valore. Quando un valore predefinito è impostato, viene modificato sincronicamente all'interno del processo e in asincronicamente in memoria non volatile, così da non interferire nell'esecuzione [23]. Ogni volta che si accede al VC, l'user defaults è sfruttato per reperire le impostazioni dell'utente, in modo che la view venga caricata con tali valori. Ogni volta che l'user modifica un valore, questo è settato al posto del precedente nel database.

Particolare attenzione va data allo switch di gestione della localizzazione. Il suo handler, oltre a salvare la preferenza nel database, si occupa di avviare la richiesta dei permessi per l'accesso al GPS di sistema da parte dell'applicazione (nel caso in cui sia la sua prima attivazione). Per questo motivo il General Options VC possiede un'istanza di LocationManager (vedi capitolo 5.5).

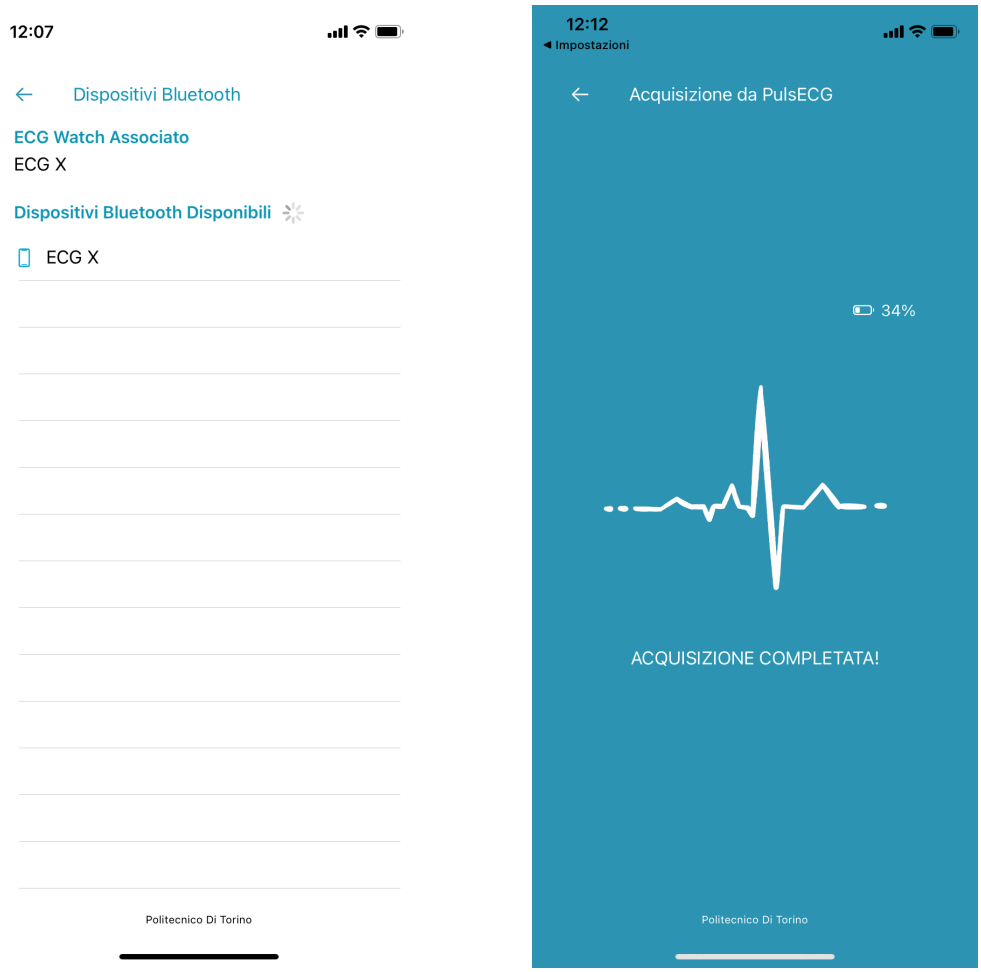
L'handler del bottone di bluetooth, a differenza degli altri, non salva dati nel database ma reindirizza l'utente, con un segue, al ConnectBluetoothDevice VC, la schermata di associazione di un dispositivo. Sarà questa schermata ad occuparsi di salvare il nome del dispositivo associato nelle database delle preferenze utente.

6.4 Connessione al bracciale e acquisizione

Le schermate che verranno analizzate in questa sezione sono quelle che permettono all'utente di connettersi ad un bracciale, creando una nuova associazione, e di acquisire una nuova misurazione dei parametri vitali. Si tratta delle pagine raggiunte cliccando rispettivamente sul primo e sul secondo bottone dell'HomePage VC.

1. **Pagina di associazione** - è usata dall'utente per avviare una scansione dei bracciali PulsEcg disponibili intorno a lui e per creare con uno di essi una nuova associazione. Se l'applicazione è associata ad un bracciale, si connetterà in automatico ad esso, ogni volta che riesce a trovarlo. Da questa schermata è anche possibile dissociare un dispositivo. Il View Controller responsabile di questa schermata è chiamato ConnectBluetoothDevice.

2. **Schermata di acquisizione** - è la schermata visualizzata dall'utente mentre acquisisce i suoi parametri usando il bracciale PulsEcg. La pagina mostra una animazione che simula un segnale ECG, il livello di batteria del bracciale (viene visualizzato non appena si tocca il sensore del bracciale) e fornisce indicazioni sullo stato dell'acquisizione (countdown e avvisi vari). Il View Controller responsabile di questa schermata è chiamato AquireECG.



(a) ConnectBluetoothDevice VC **(b)** AquireECG VC

Figura 6.10: UI dei View Controller AquireECG e ConnectBluetoothDevice

6.4.1 ConnectBluetoothDevice VC

Questo View Controller permette all'utente di avviare una scansione, associare un dispositivo tra quelli disponibili ed eventualmente dissociarlo. Come si può vedere

in Fig. 6.11 la view è composta da: un bottone per tornare indietro, alcune labels per visualizzare il dispositivo attualmente associato e una UITableView per ospitare i bracciali PulsEcg.

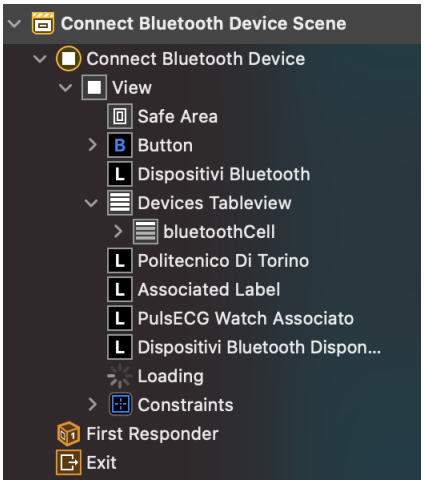


Figura 6.11: Struttura delle views e segue connessi al ConnectBluetoothDevice VC

Per realizzare tutte le funzionalità relative al bluetooth, il VC sfrutta l'istanza di una classe wrapper delle funzionalità offerte dalla libreria CoreBluetooth: **BleManager**. Questa classe opera in maniera nascosta, in background, dopo che ConnectBluetoothDevice ha chiamato alcuni suoi metodi, e poi notifica i suoi risultati con dei meccanismi asincroni, intercettati dal VC mediante degli observer, registrati al momento della sua creazione. Nell'istante in cui si entra in questo VC, il bluetooth è già attivo, e con lui la scansione di nuove periferiche, poichè è stato inizializzato dall'HomePage VC. Ogni volta che un nuovo dispositivo viene rilevato dallo scanner, la BleManager manda una notifica al ConnectBluetoothDevice VC, che reagisce aggiornando la vista della tabella, in modo che rispecchi i dati.

Esattamente come nella tableview dell'Homepage, anche in questo caso è stato necessario rendere il VC delegato della tabella e customizzare le sue callback. D'altronde, le prime due verranno chiamate immediatamente, all'atto della creazione della view. In questo caso le funzioni si interfacciano direttamente con la classe BleManager. La *numberOfRowsInSection* restituisce la dimensione di un vettore ottenuto dalla classe wrapper. Questo vettore contiene tutti i riferimenti alle periferiche che sono state trovate in fase di scanning. Analogamente la *cellForRowAt* inizializza ogni cella con una periferica del vettore, di cui verrà visualizzato il nome. La terza funzione, la *didSelectRowAt* (Cod. 6.3) avvia la connessione con la periferica selezionata dall'utente: la periferica corrispondente è restituita alla funzione; lo scanning è interrotto; viene richiesta la connessione con la periferica.

Se la connessione avverrà con successo, il BleManager notificherà il VC. L'handler di gestione di quella notifica aggiornerà la view, segnalando nella label apposita il nome del nuovo dispositivo connesso.

Listing 6.3: Funzione didSelectRowAt del TableViewDelegate di ConnectBluetoothDevice VC

```
1 func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
    IndexPath){
2         let dispositivoSelezionato = BleManager.sharedInstance.
    getScannedPeripherals()[indexPath.row]
3         loading.stopAnimating()
4         loading.isHidden = true
5         BleManager.sharedInstance.stopScanning()
6         BleManager.sharedInstance.connect(peripheral:
    dispositivoSelezionato)
7         tableView.deselectRow(at: indexPath, animated: true)
8     }
```

Il ConnectBluetoothDevice, oltre ad occuparsi dello scanning, controlla anche la disponibilità del dispositivo precedentemente associato (se ne esiste uno). Anche in questo caso quindi si sfrutta la classe UserDefaults per reperire il nome del PulsEcg preferito. Una volta ottenuto, si controlla se il dispositivo sia stato rilevato in fase di scan. In caso positivo, si avvia una connessione automatica grazie ad un metodo della BleManager e si interrompe lo scanning. Quando la connessione è riuscita, la classe BleManager notifica il VC, e l'handler dell'observer aggiorna un'apposita label con il nome del dispositivo connesso.

La scansione ha un tempo limitato, della durata di 15 secondi. Ciò è implementato mediante un timer. Al suo scadere, viene chiamato un handler, passato come parametro al costruttore del timer, che, quando eseguito, interromperà lo scanning.

Lo scanning però può anche essere riattivato, utilizzando una tipica gesture tap-to-refresh sulla tabella. Ciò è possibile grazie all'istanza di un oggetto **UIRefreshControl**, che, agganciato ad una Scroll View (come ad esempio l'UITableView), permette di avviare l'aggiornamento dei suoi contenuti. Quando l'utente trascina verso il basso la parte superiore dell'area del contenuto scorrevole, la Scroll View rivela una zona di controllo dell'aggiornamento, in cui è presente un indicatore di avanzamento animato, e invia una notifica all'app che verrà gestita da un handler [23]. Come si può vedere nel codice 6.4, l'oggetto di refresh viene creato, gli viene associato un messaggio da visualizzare ed un handler e viene agganciato alla tableview. La funzione di gestione chiude il controllo di aggiornamento, riavvia lo scanning e crea nuovamente un timer che interromperà lo scan tra altri 15 secondi.

Listing 6.4: Inizializzazione

```
1
```

```
refreshControl.attributedTitle = NSAttributedString(string: "Pull to
    refresh")
refreshControl.addTarget(self, action: #selector(self.refresh(_:)),
    for: .valueChanged)
devicesTableview.addSubview(refreshControl)

@objc func refresh(_ sender: AnyObject) {
        refreshControl.endRefreshing()
        BleManager.sharedInstance.startScanning()
        loading.isHidden = false
        loading.startAnimating()
        timer = Timer.scheduledTimer(withTimeInterval: 15.0, repeats:
    false) { timer in
            self.loading.stopAnimating()
            self.loading.isHidden = true
            BleManager.sharedInstance.stopScanning()
            print("Timer fired!")
        }
```

6.4.2 AquireECG VC

La schermata di acquisizione viene avviata a partire dall'Homepage, quando un dispositivo associato è disponibile e connesso al bluetooth del telefonino. L'interfaccia associata a questo view controller è costituita esclusivamente da un bottone per tornare indietro, da una gif e da due label aggiornate dinamicamente.

Considerando che questa pagina sarà visualizzata per tutta la durata dell'acquisizione (durata minima di 10 secondi), il suo periodo di intenso sforzo computazionale è in realtà concentrato nell'ultimo secondo. Fatta eccezione per la prima istruzione, la quale invia il comando di inizio acquisizione al bracciale, durante tutto il tempo necessario alla procedura di misurazione, la view si limita a mostrare un contatore ed una GIF animata a forma di onda ECG. In questa fase infatti, la vera funzione del view controller è quella di accompagnare l'utente durante l'acquisizione, dandogli costantemente dei feedback grafici in modo da evitare che pensi che l'applicazione si sia bloccata e provi una conseguente frustrazione. Parte dei feedback visualizzati sullo schermo sono comunicati all'utente dopo aver ricevuto delle notifiche sullo stato dell'acquisizione. Anche questo view controller infatti si appoggia alla classe wrapper BleManager, la quale, dopo aver ricevuto il comando di write sul dispositivo, scatena diversi tipi di notifiche, intercettate dall'Aquire VC e visualizzate sullo schermo. Le notifiche riguardano l'inizio dell'acquisizione, la ricezione del valore di batteria e il completamento della misurazione. Inoltre la view viene aggiornata ogni secondo per poter decrementare il contatore.

Come abbiamo detto precedentemente, tutto lo sforzo del VC è concentrato nella parte terminale del suo tempo di vita. L'inizio di questa attività è sancito dalla ricezione della notifica *acquisitionComplete*. L'handler della notifica (Cod. 6.5) avvia quindi la sua esecuzione, la quale si articola attraverso diversi passi:

1. reperire i vettori di campioni dell'ECG e PPG dal BleManager;
2. preprocessare i segnali ECG e PPG per rimuovere gli effetti sulla misurazione dovuti all'hardware;
3. convertire i segnali in double;
4. filtrare e pulire i segnali;
5. calcolare i picchi dell'ECG per la determinazione dell'heartbeat e rilevare l'eventuale presenza di fibrillazione atriale;
6. calcolare i massimi e i minimi locali del PPG per poter determinare il valore di SpO2;
7. dare i segnali ECG e PPG filtrati in input alla rete neurale, per ricavare i valori delle pressioni. Questo punto, insieme ai due precedenti, è particolarmente importante. Infatti determinare questi valori ora e salvarli nel file non rende necessario ripetere tutti i calcoli, ogni volta che si visualizza una misurazione dallo storico;
8. creare un file (se il salvataggio automatico è abilitato) in cui salvare nome, cognome, data, valore di SpO2, valore di HB, eventuale presenza di FA, valore di SBP e valore DBP. Nel file inoltre devono essere salvati i segnali nello stato in cui erano al termine del punto 2.

L'elaborazione dei segnali è realizzata chiamando le funzioni della classe SignalProcessing e mandando delle richieste di servizio alla classe singleton NeuralNetworkManager. Queste procedure verranno affrontata nel dettaglio nel capitolo 7.2 (signal processing) e nel capitolo 7.4 (derivazione della pressione).

Listing 6.5: Handler della notifica di termine acquisizione

```
@objc private func acquisitionComplete(_ notification: Notification)
    {
        timerAquire!.invalidate()
        info.text = "ACQUISIZIONE COMPLETATA!"

        //genero il nome del file
        let defaults = UserDefaults.standard
        var cog = defaults.object(forKey: "Cognome") as? String ??
    String()
```

```
8           var nom = defaults.object(forKey: "Nome") as? String ??
    String()
9           if cog.isEmpty {
10              cog = "Cognome"
11          }
12          if nom.isEmpty {
13              nom = "Nome"
14          }
15          fileName = generateFileName(nome: nom, cognome: cog)
16          let components = fileName?.components(separatedBy: " ")
17          let giorno : String = components?[2] ?? "01/01/01"
18          var ora : String = components?[3] ?? "01:01:01"
19          ora = ora.replacingOccurrences(of: ",", with: ":", options: .
    literal, range: nil)
20          let date = giorno + " " + ora
21
22          //converto dati in double
23          let ECGDouble = fromUInt16ToDouble(segnale: BleManager.
    sharedInstance.getECGSamples())
24          let PPGIRDouble = fromUInt32ToDouble(segnale: BleManager.
    sharedInstance.getPPGIRSamples())
25          let PPGREDDouble = fromUInt32ToDouble(segnale: BleManager.
    sharedInstance.getPPGREDSamples())
26          //inverto i segnali ppg sull'asse y
27          let PPGIRInvertito = inversioneAsseY(segnale: PPGIRDouble)
28          let PPGREDInvertito = inversioneAsseY(segnale: PPGREDDouble)
29          //elimino amplificazione ecg
30          var ECGdeAmplificati = deamplifica(segnale: ECGDouble)
31          //inverto segnale ECG se ho impostato braccio sinistro
32          let arm = defaults.integer(forKey: "Braccio")
33          if arm == 0 {
34              ECGdeAmplificati = inversioneAsseYECG(segnale:
    ECGdeAmplificati)
35          }
36
37          //FASE DI FILTRAGGIO E CALCOLO HB,SPO2,PRESS
    ----------------------------------------
38          //filtraggio e detrending ppg
39          let PPGIRFiltrato = mediaMobile(segnale: mediaMobile(segnale:
     PPGIRInvertito, finestra: 20), finestra: 20)
40          let PPGREDFiltrato = mediaMobile(segnale: mediaMobile(segnale
    : PPGREDInvertito, finestra: 20), finestra: 20)
41          let PPGIRFiltratoEDetrend = detrending(segnale: PPGIRFiltrato
    )
42          let PPGREDFiltratoEDetrend = detrending(segnale:
    PPGREDFiltrato)
43
44          //filtraggio ECG
45          var ECGfiltrato = notchFilter_10(x: ECGdeAmplificati)
```

```
46          ECGfiltrato = mediaMobile(segnale: mediaMobile(segnale:
    ECGfiltrato, finestra: 10), finestra: 10)
47          for i in 0 ..< ECGfiltrato.count {
48              ECGfiltrato[i] = 2 * ECGfiltrato[i]
49          }
50
51          //calcolo HB, fibrillazione e SpO2
52          var dataPointsPeaks : [ChartDataEntry] = [ChartDataEntry](
    repeating: ChartDataEntry(x:0,y:0), count: 2 * ((Constants.MAX_BPM
     / 60) * Constants.SECONDS_OF_AQUISITION))
53          let absoluteMax = averaged_max(input : ECGfiltrato)
54          findPeak(input: ECGfiltrato, GVDataPeaks: &dataPointsPeaks,
    absolute_max: absoluteMax)
55          let heart_rate = heartBeatRate(dataPoint : dataPointsPeaks)
56          let HB : Int = heart_rate[0]
57          var atrialFibrillation : Bool = false //da fare
58          if(heart_rate[3] == 1) {
59                atrialFibrillation = true
60          }
61          let SpO2 : Int = calcolaSpO2(filteredPPG_IR: PPGIRFiltrato,
    filteredPPG_RED: PPGREDFiltrato)
62
63          //Applica rete per calcolo di PressioneSis, PressioneDia
64          let rete = NeuralNetworkManager.sharedInstance
65          rete.computePressioni(ECG: <#T##[Double]#>, PPG_RED:
    PPGREDFiltratoEDetrend)
66          let pressSis : Int = 0//rete.getPressSis()
67          let pressDia : Int = 0//rete.getPressDia()
68          //FINE FILTRAGGIO E CALCOLO HB,SPO2,PRESS
    ------------------------------------------------------------
69
70          //qui salvo vettori filtrati
71          filterMisurazione = Misurazione(name: nom, surname: cog, date
    : date,HB: HB, SpO2: SpO2, pressSis: pressSis, pressDia: pressDia,
     ECGSamples: ECGfiltrato, PPGSamplesIR: PPGIRFiltratoEDetrend,
    PPGSamplesRED: PPGREDFiltratoEDetrend, atrialFibrillation:
    atrialFibrillation)
72          //qui salvo vettori non filtrati
73          misurazione = Misurazione(name: nom, surname: cog, date: date
    , HB: HB, SpO2: SpO2, pressSis: pressSis, pressDia: pressDia,
    ECGSamples: ECGdeAmplificati, PPGSamplesIR: PPGIRInvertito,
    PPGSamplesRED: PPGREDInvertito, atrialFibrillation:
    atrialFibrillation)
74
75          //se l'autosave è attivo salvo la misurazione in un file json
76          let autosave = defaults.bool(forKey: "AutosaveAttivo")
77          if autosave == true {
78              print("Salvataggio automatico - Scrivo il file")
79
```

```
80
81              let jsonEncoder = JSONEncoder()
82              jsonEncoder.outputFormatting = .prettyPrinted
83              var jsonData : Data? = nil
84
85              do {
86                  jsonData = try jsonEncoder.encode(misurazione)
87              }catch{print("Errore di encoding")}
88              if jsonData != nil {
89                  let json = String(data: jsonData!, encoding: String.
    Encoding.utf8)
90                  let dir = FileManager.default.urls(for: .
    documentDirectory, in: .userDomainMask).first!
91                  filePath = dir.appendingPathComponent(fileName!)
92                  filePath = filePath!.appendingPathExtension("json")
93                  do{
94                      try json!.write(to: filePath!, atomically: true,
    encoding: String.Encoding.utf8)
95                  }catch{ print("errore di scrittura file")}
96              }
97              performSegue(withIdentifier: "
    from_AquireECG_to_GeneralShowECG", sender: nil)
98          }
99          else{
100             print("Salvataggio NON automatico - Visualizzo
    direttamente")
101             performSegue(withIdentifier: "
    from_AquireECG_to_GeneralShowECG", sender: nil)
102         }
103     }
```

Nel codice 6.5 è presente una sezione in cui viene generato il file json da salvare in memoria. Il file verrà salvato solo nel caso in cui l'utente abbia precedentemente richiesto il salvataggio automatico delle misurazioni. Questa informazione è ottenuta interrogando la classe UserDefaults e reperendo le preferenze di applicazione dell'utente. Il file si ottiene utilizzando un'istanza della classe User Defined Type (UDT) chiamata **Misura**. La classe Misura è una semplicissima classe composta da diversi membri privati (vedi Fig. 6.12) che descrivono i parametri acquisiti durante una misurazione e da metodi di costruzione, setting e getting. L'AquireECG, ottenuti i segnali dal BleManager, crea due oggetti Misurazione che modellano l'acquisizione: il primo possiede i valori così come devono essere salvati nel file; il secondo i valori che devono essere visualizzati sullo schermo, ovvero prefiltrati. Si è scelto di non salvare nel file i segnali filtrati allo scopo di non perdere l'informazione originale ottenuta dal dispositivo. Ciò permetterebbe infatti di aggiungere in seguito ulteriori analisi del segnale, o di aggiornare i filtri ad una nuova versione, facendo sì che tutte le vecchie acquisizioni siano compatibili con i nuovi algoritmi. I due oggetti

di tipo Misurazione, filtrato e non, vengono prodotti dall'AquireECG e scambiati tra le classi GeneralShowECG e ShowECG, in modo da evitare di dover rileggere il file e rielaborare i dati ad ogni segue.

```
class Misurazione : Codable{
    private var name : String
    private var surname : String
    private var date : String
    private var HB : Int
    private var SpO2 : Int
    private var pressSis : Int
    private var pressDia : Int
    private var ECGSamples : [Double] = []
    private var PPGSamplesIR : [Double] = []
    private var PPGSamplesRED : [Double] = []
    private var atrialFibrillation: Bool
```

Figura 6.12: Definizione dei membri che compongono la classe Misurazione

6.5 Archivio

La schermata dell'Archivio (gestita dall'omonimo VC, il cui aspetto è raffigurato in Fig. 6.13) è visualizzata dall'utente quando desidera navigare tra i file che compongono lo storico delle acquisizioni. Attraverso questa pagina è possibile importare nuovi file dall'esterno, cercare dei file per nome e visualizzare il dettaglio di ogni misurazione nell'elenco.

La view di questo VC è composta da alcuni bottoni (backbutton, bottone di importazione e bottone di ricerca) e da una complessa UITableView per elencare i file. Questa tabella infatti, a differenza di quelle utilizzate nell'Homepage e nel ConnectBluetoothDevice VC, è suddivisa in **sezioni** di file, identificate dalla coppia <mese-anno> (in cui il file è stato acquisito).

Per ottenere questo risultato, il vettore che ospita gli elementi della tabella ha un formato molto particolare. Si tratta di un dizionario contenente coppie <chiave-valore> che sfrutta come chiave una stringa, e come valore un vettore di oggetti FileWrapper. FileWrapper è una classe UDT che modella una cella della tabella, ovvero un file (descrive il mese, l'anno, il giorno e il timestamp d'acquisizione, nonchè il nome e il path del file). Ogni stringa del dizionario indica il nome di una sezione. Il vettore di files associato a quella stringa è la collezione di celle della tabella che dovranno essere impilate dentro quella sezione.

Le callback del TableView Delegate, per gestire la funzionalità della divisione in sezioni, sono in numero maggiore rispetto a quelle usate fin ora. Le funzioni *cellForRowAt* e *didSelectRowAt*, si comportano come nelle tabelle prive di sezioni.

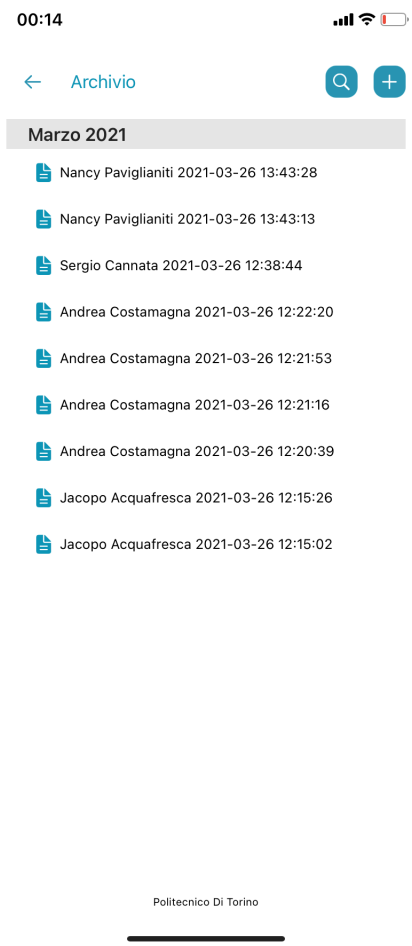


Figura 6.13: UI del View Controller Archivio. Nello screenshot sono presenti file appartenenti ad un'unica sezione mese-anno

L'unica differenza sta nell'accesso al vettore, poichè è necessario specificare sia una stringa di sezione, che un indice di file interno alla sezione. Quando il metodo didSelectRowAt viene scatenato dal click di un file, avviene un trasfrimento al GeneralShowECG VC, mediante un segue di tipo present modally, a cui viene passato il file come parametro. Negli altri casi, il metodo *numberOfRowsInSection* veniva usato come se la tabella contenesse una sola sezione. In questo caso però le sezioni sono più di una, quindi, invece che ritornare tutti gli elementi della tabella, ritornerà il numero di elementi contenuti nel vettore che corrisponde all'indice di sezione ricevuto come argomento, chiave del dizionario. I nuovi metodi sono: **numberOfSections**, che restituisce il numero di sezioni; **titleForHeaderInSection**, che associa ad ogni sezione un titolo in formato stringa.

Per riempire il dizionario di vettori di files, la directory dedicata alla memoria

d'applicazione viene scansionata. I nomi dei files che vengono individuati in questa fase sono restituiti all'applicazione. Il nome di ogni file viene scandito, in modo da identificarne l'anno e il mese d'acquisizione. Se la coppia <mese-anno> è già presente tra le sezioni del dizionario, il file viene aggiunto al suo vettore. Altrimenti, prima viene creata la sezione, poi viene inizializzato un vettore ad essa associato, ed infine il file è inserito nell'array.

Il **bottone di importazione** è connesso ad un'action che scatena la creazione di un DocumentPicker VC. Tale View Controller è gestito come descritto nel capitolo 5.7.2. Una volta che il file importato è stato copiato in directory e il suo oggetto FileWrapper è aggiunto al giusto vettore in base alla data, la tableview viene aggiornata per mostrare una nuova riga.

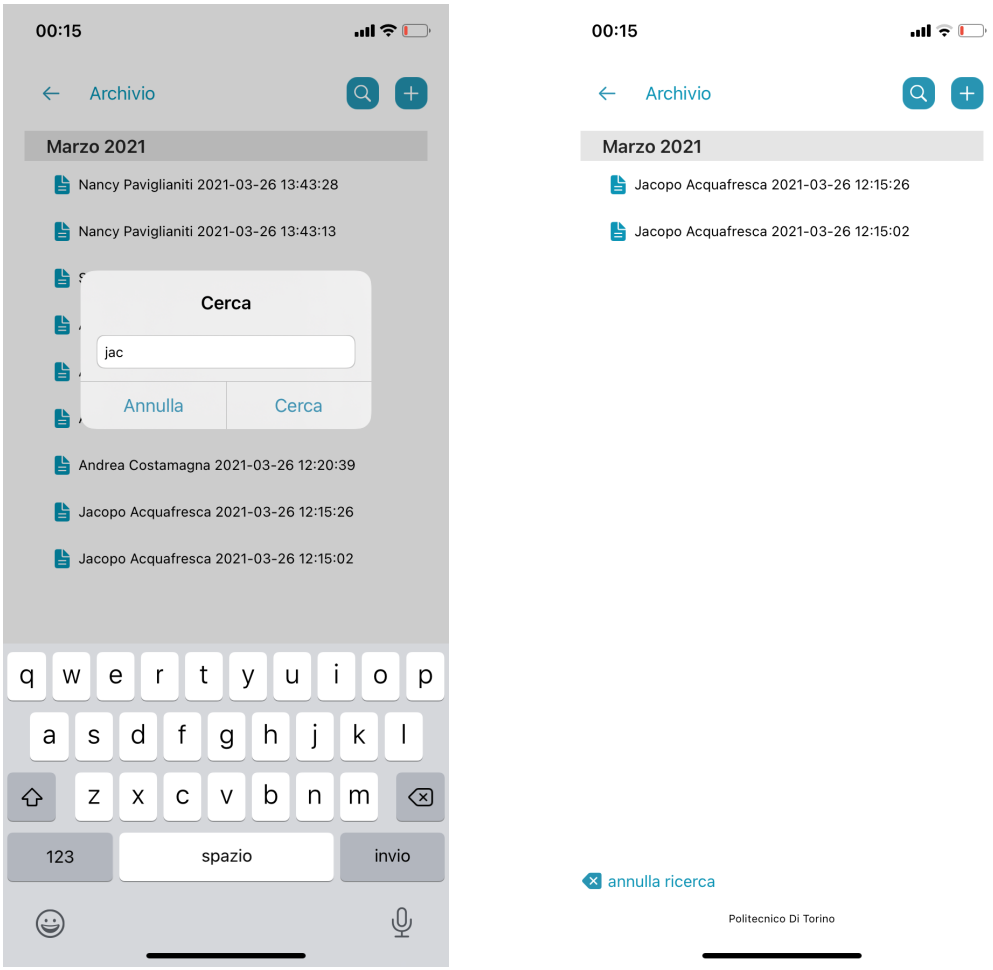
Il **bottone di ricerca** avvia un'action che mostra sullo schermo un alert editabile (Fig. 6.14a). Una volta inserita la parola da cercare, ed una volta che si è premuto sul bottone "Cerca", un nuovo metodo è lanciato. Tale metodo applica un filtro sui file disponibili in archivio. Il filtro lascia passare solo i file il cui nome contiene la parola inserita nella zona editabile dell'alert. I file risultato del filtraggio sono inseriti nel vettore dei dati della tableview e la vista della tabella viene aggiornata. Un bottone che sino ad ora è stato nascosto, viene a questo punto mostrato. Al bottone è connessa un'action che ricarica nel vettore tutti i file presenti nella directory e avvia un refresh della tableview, ovvero, in parole più semplici, annulla il filtraggio e riporta la vista allo stato originale.

6.6 Visualizzazione della misurazione

In questo capitolo verranno presentate le schermate usate dall'utente per visualizzare l'esito di una misurazione. Queste schermate sono il comune punto di incontro della navigazione proveniente dalla consultazione di un file nell'archivio e dalla acquisizione di nuovi parametri vitali. La visualizzazione è suddivisa in due pagine:

1. schermata di riepilogo della misurazione, implementata attraverso il View Controller **GeneralShowECG**. Questa pagina è la prima ad aprirsi dopo aver cliccato su un file dello storico o dopo aver terminato una nuova acquisizione da bracciale. Attraverso questa vista, l'utente può prendere rapidamente nota dei valori di HB, SpO2, delle pressioni e dell'eventuale rilevazione di fibrillazione atriale. Il grafico PPG non è mostrato e quello ECG è appena abbozzato, giusto per dare un idea dell'andamento. I dati sono organizzati in schede cliccabili dall'utente, per prendere visione di maggiori dettagli.

2. schermata di dettaglio della misurazione, gestita dallo **ShowECG** View Controller. Questa pagina viene aperta quando l'utente interagisce con le schede



(a) HomePage VC **(b)** Risultato di una ricerca dell'Archivio VC

Figura 6.14: Ricerca tra i file dell'Archivio VC

del GeneralShowECG VC. Lo stesso View COntroller permette di muoversi tra 3 possibili views: il dettaglio dell'ECG; il dettaglio del PPG; il dettaglio delle pressioni. Rispetto alla scheda cliccata nel VC precedente, verrà aperto un determinato dettaglio. L'utente può navigare tra i dettagli utilizzando degli appositi bottoni. A partire da questo VC è anche possibile gestire l'eliminazione del file e la sua esportazione.

6.6.1 GeneralShowECG VC

La view della classe riepilogativa della misurazione (la cui struttura è raffigurata in Fig. 6.15) sembrerebbe essere costituita da un bottone, alcune labels e da diverse

schede. In realtà, per via di problemi implementativi nell'inserire l'anteprima del grafico, le schede non sono delle specifiche view contenitore, ma sono composte da UILabel affiancate, a cui è stato assegnato un colore di background. Il grafico ECG è invece realizzato con una LineChartView (libreria Charts, vedi Cap. 5.6) opportunamente configurato per non mostrare alcuna griglia e per non essere interattivo (non è zoomabile o scrollabile). Il grafico appare sullo schermo gradualmente, con un'animazione. Si è scelto di mostrare nell'anteprima del grafico solo 5 secondi di acquisizione, e non i primi 5, così da evitare quelle sgradevoli deformazioni tipiche della fase iniziale, dovute al ritardo di posizionamento delle dita dell'utente sugli elettrodi. L'aspetto della pagina risultante può essere osservato in Fig. 6.16, raffigurante un'acquisizione dello schermo del cellulare quando il GeneralShowECG VC è attivo.

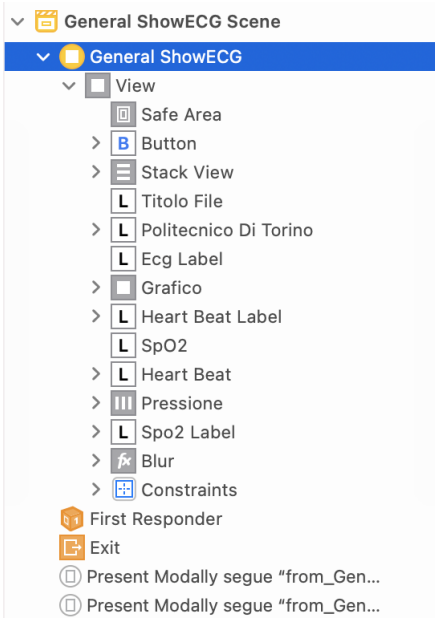


Figura 6.15: Struttura delle views e segue connessi al GeneralShowECG VC

La logica di presentazione dei dati nel view controller dipende da quale VC sia stato sorgente del segue.

Se il GeneralShowECG VC è istanziato **a partire dall'AquireVC**, questo si è già occupato della creazione di due oggetti di tipo UDT Misurazione: uno contenente i segnali filtrati, pronti alla visualizzazione; uno contenente i segnali non filtrati, così come sono stati salvati nel file. Questi oggetti vengono poi passati alla schermata di riepilogo, grazie all'override della funzione di prepare al segue.

Nel caso in cui invece fossimo giunti a questo VC **passando per l'Archivio**, il reperimento dei dati dell'acquisizione dovrà essere implementato esplicitamente,

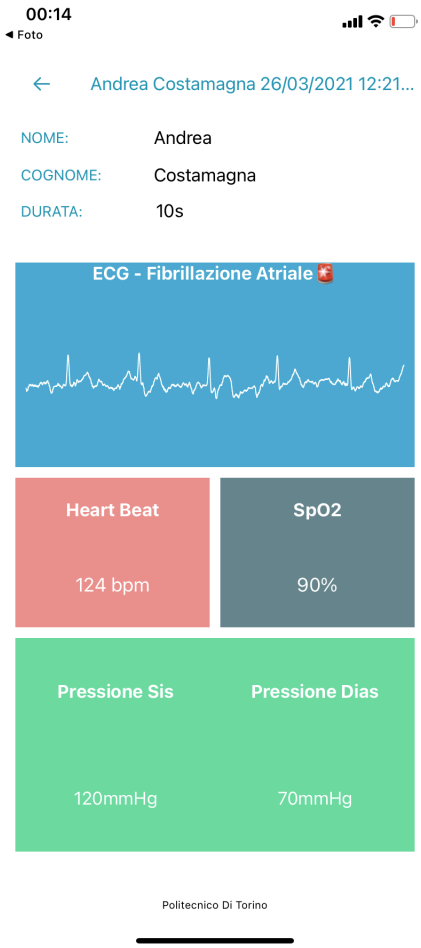


Figura 6.16: GeneralShowECG VC

con la lettura del file Json corrispondente. Il file può essere automaticamente convertito in un oggetto Misurazione, utilizzando un Json Decoder che trasformi la stringa contenuta nel documento in un oggetto json. Sebbene a questo punto non sia necessario ricalcolare i valori di SpO2, HB, pressioni e fibrillazione atriale, i segnali devono essere nuovamente filtrati per poterli visualizzare sullo schermo. In realtà, escluso un piccolo anteprima del segnale ECG, le onde verranno mostrate nel dettaglio solo nel ShowECG VC. Si è comunque scelto di filtrare già da ora i segnali, in modo da unificare la logica di preparazione al segue verso il prossimo VC.

Sia che si provenga dall'AquireECG che dall'Archivio, a seguito dell'esecuzione del GeneralShowECG, il segue verso lo ShowECG verrà decorato con i due oggetti Misurazione pronti all'uso, esattamente come sono stati ricevuti nel primo caso, appena calcolati a partire dal file nel secondo.

A questo punto, il VC preleva i dati dalla Misurazione e li posiziona nelle varie label della vista, connesse allo storyboard con numerosi outlet. Oltre alla mera visualizzazione, il VC si impegna nel controllo dei valori, così da poter segnalare le situazioni anomale, avvisaglie di una condizione di salute a rischio:

1. presenza della fibrillazione atriale;
2. percentuale dell'SpO2 nel sangue troppo bassa, inferiore al 90
3. heart beat troppo alto (>=140 bpm) o troppo basso (<=60 bpm).
4. ipertensione arteriosa grave (SBP >=180 oppure DBP >= 110).

I valori problematici sono accompagnati da un emoji raffigurante una luce di allarme. Inoltre, è stato fatto un override del metodo viewDidAppear (chiamato dal SO nel momento in cui la vista del VC ha terminato la sua costruzione e presentazione sullo schermo) in modo da: mostrare una schermata di loading nel caso in cui si debba filtrare il segnale; mostrare un avviso (sotto forma di alert) nel caso in cui si provenga dall'AquireECG e siano state rilevate delle anomalie nei parametri ricevuti.

Per poter accedere allo ShowECG VC toccando le schede, è stato necessario utilizzare dei riconoscitori di gesture. Infatti, visto che le schede sono realizzate con delle semplice labels, non è previsto un meccanismo automatico di associazione del tocco ad una action come per gli UIButton. Per forzare un'interazione con le labels, si è usato un oggetto **UIGestureRecognizer**.

Un oggetto di questo tipo possiede la logica per riconoscere una sequenza di tocchi e agire di conseguenza. Quando un UIGestureRecognizer rileva un certo gesto, invia una notifica ad uno o più destinatari designati. I riconoscitori di gesti devono essere creati e poi associati ad una o più funzioni destinatarie e ad una view che diventerà interattiva. L'associazione ad una vista (e di conseguenza a tutte le sue sottoviste) viene dichiarata chiamando sull'oggetto UIView il metodo *addGestureRecognizer* [23].

Come è possibile osservare nel codice 6.6, sono stati creati sette TapGesture-Recognizer (è l'UIGestureRecognizer in grado di riconoscere il tap su una view). Sebbene le schede siano solo 4, i riconoscitori di gesture sono sette perchè, come anticipato precedentemente, gran parte dei riquadri sono costituiti da due labels, ed entrambi i componenti devono essere reattivi, così da illudere l'utente di essere parte di un unico oggetto grafico. Il costruttore del riconoscitore del tap riceve come argomento il VC e una funzione handler dell'evento di tocco. Le gesture sono state poi associate agli outlet delle views, connesse al VC mediante Storyboard. Le funzioni handler si limitano ad realizzare un segue verso il VC di dettaglio, decorandolo con una stringa che specifica il dettaglio che dovrà essere visualizzato.

Tale stringa, nella funzione di prepare al segue, verrà passata allo ShowECG VC insieme ai due oggetti della classe Misurazione, al nome ed al path del file, e ad un riferimento del VC sorgente.

Listing 6.6: Definizione delle tap gesture e associazione alle schede ed ai metodi handler

```
1     override func viewDidAppear(_ animated: Bool){
2         ..............
3         let tapGrafico = UITapGestureRecognizer(target: self, action:
      #selector(self.tap_ECG(_:)))
4         let tapECGLabel = UITapGestureRecognizer(target: self, action
      : #selector(self.tap_ECGLabel(_:)))
5         let tapHBLabel = UITapGestureRecognizer(target: self, action:
      #selector(self.tap_HBLabel(_:)))
6         let tapHB = UITapGestureRecognizer(target: self, action: #
      selector(self.tap_HB(_:)))
7         let tapspo2Label = UITapGestureRecognizer(target: self,
      action: #selector(self.tap_spo2Label(_:)))
8         let tapspo2 = UITapGestureRecognizer(target: self, action: #
      selector(self.tap_spo2(_:)))
9         let tappressione = UITapGestureRecognizer(target: self,
      action: #selector(self.tap_pressione(_:)))
10        grafico.addGestureRecognizer(tapGrafico)
11        ecgLabel.addGestureRecognizer(tapECGLabel)
12        HeartBeat.addGestureRecognizer(tapHB)
13        heartBeatLabel.addGestureRecognizer(tapHBLabel)
14        spo2Label.addGestureRecognizer(tapspo2Label)
15        SpO2.addGestureRecognizer(tapspo2)
16        pressione.addGestureRecognizer(tappressione)
17
18        ..............
19    }
20
21    @objc func tap_spo2Label(_ sender: UITapGestureRecognizer? = nil)
      {
22        performSegue(withIdentifier: "from_GeneralShowECG_to_ShowECG"
      , sender: "spo2")
23    }
24    @objc func tap_spo2(_ sender: UITapGestureRecognizer? = nil) {
25        performSegue(withIdentifier: "from_GeneralShowECG_to_ShowECG"
      , sender: "spo2")
26    }
27    @objc func tap_pressione(_ sender: UITapGestureRecognizer? = nil)
      {
28        performSegue(withIdentifier: "from_GeneralShowECG_to_ShowECG"
      , sender: "press")
29    }
30    @objc func tap_ECG(_ sender: UITapGestureRecognizer? = nil) {
```

```
31          performSegue(withIdentifier: "from_GeneralShowECG_to_ShowECG"
        , sender: "ECG")
32      }
33      @objc func tap_ECGLabel(_ sender: UITapGestureRecognizer? = nil)
        {
34          performSegue(withIdentifier: "from_GeneralShowECG_to_ShowECG"
        , sender: "ECG")
35      }
36      @objc func tap_HB(_ sender: UITapGestureRecognizer? = nil) {
37          performSegue(withIdentifier: "from_GeneralShowECG_to_ShowECG"
        , sender: "ECG")
38      }
39      @objc func tap_HBLabel(_ sender: UITapGestureRecognizer? = nil) {
40          performSegue(withIdentifier: "from_GeneralShowECG_to_ShowECG"
        , sender: "ECG")
41      }
```

6.6.2 ShowECG VC

Questo View Controller ha la funzione di mostrare con precisione i grafici acquisiti dal bracciale PulsEcg. Come mostrato nella sua struttura gerarchica dello storyboard (Fig. 6.17), la view è composta da:

1. diversi bottoni. Uno per tornare indietro al VC precedente, uno per eliminare la misurazione, uno per accedere all'InfoECG VC e uno per esportare il file.

2. due labels per descrivere il dettaglio sottostante con un titolo e con i valori associati.

3. uno Stack View orizzontale composto da bottoni. Questi bottoni permettono di aggiornare la view, in modo da mostrare il dettaglio della misurazione desiderato.

4. una LineChartView per mostrare il grafico. A seconda del grafico che si desidera visualizzare, i dati di input della view vengono sostituiti, ma la view resta sempre la stessa. Questa view viene nascosta quando si desidera consultare il dettaglio sulle pressioni.

5. un SV orizzontale che contiene a sua volta due SV verticali per mostrare i valori di pressione. Questa view viene nascosta quando si visualizzano i grafici.

Il view controller inizializza il contenuto della propria view in base alla scheda che è stata selezionata nel VC precedente. Ciò è possibile perchè la sua variabile *schermataDaVisualizzare* è stata inizializzata dalla funzione di prepare del GeneralShowECG VC, in modo da contenere una stringa che indichi il dettaglio da visualizzare. Le possibilità sono tre:

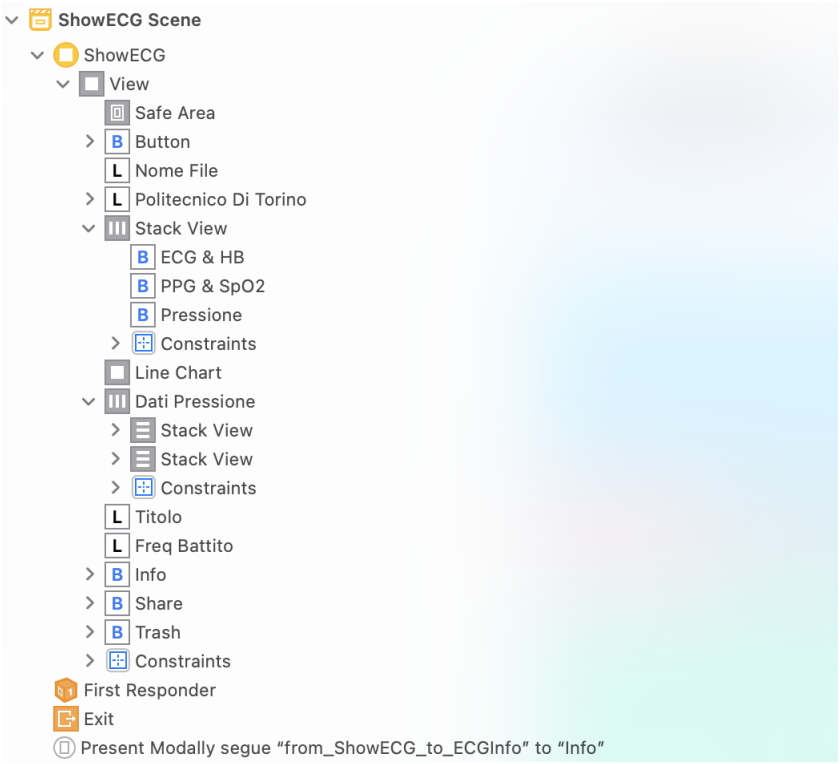


Figura 6.17: Struttura delle views e segue connessi al ShowECG VC

1. l'utente clicca sulla scheda dell'ECG o su quella dell'HB -> viene reindirizzato al **dettaglio sull'ECG**, mostrato in Fig. 6.18

2. l'utente clicca sulla scheda dell'SpO2 -> viene reindirizzato al **dettaglio sul PPG**, mostrato in Fig. 6.19

3. l'utente clicca sulla scheda delle pressioni -> viene reindirizzato al **dettaglio sulle pressioni**, mostrato in Fig. 6.20

Nel metodo *viewDidLoad* del VC vengono chiamate due funzioni: la funzione di preparazione dei dati da visualizzare e la funzione per configurare il VC in modo che mostri la giusta pagina sullo schermo.

La **preparazione dei dati** consiste semplicemente nel richiamare i getter sull'oggetto Misurazione filtrato ricevuto dal VC precedente. I getter restituiscono le informazioni da inserire nelle labels della view ed i campioni dei segnali da porre negli oggetti *LineChartData*, dati poi in input alla LineChartView per mostrare i grafici. Per quanto riguarda il segnale ECG, è necessario ricalcolare i picchi, così da poter essere aggiunti al grafico. Quando dovrà essere visualizzato l'ECG, la LineChartView riceverà come argomento di configurazione un LineChartData

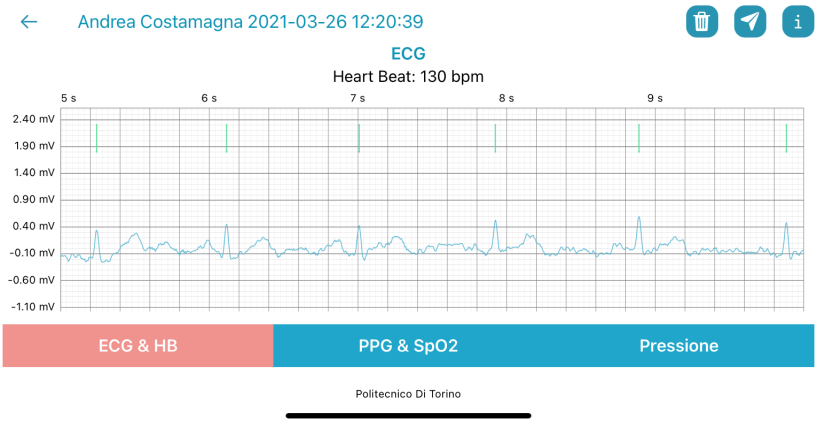


Figura 6.18: Dettaglio sull'ECG dello ShowECG VC

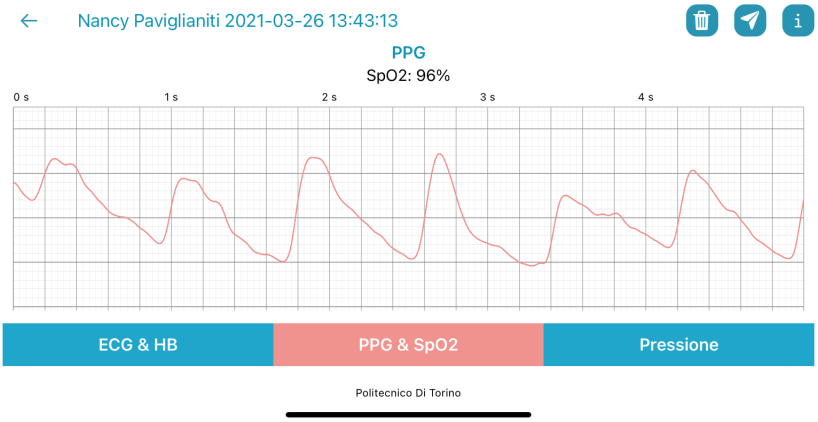


Figura 6.19: Dettaglio sul PPG dello ShowECG VC

contenente un vettore di set: un set per i campioni del segnale e tanti set quanti sono i picchi da mostrare.

La **funzione di inizializzazione della schermata iniziale** chiama, rispetto al valore della variabile *schermataDaVisualizzare*, una funzione specifica per visualizzare i giusti contenuti, scegliendo tra: mostraECG(), mostraPPG() e mostraPressione(). Oltre alla configurazione iniziale, il VC modificherà nuovamente la view per rispondere agli input dati dall'utente, nel selezionare una scelta dal menu in basso. Ogni metodo action connesso mediante lo storyboard ad uno dei bottoni del menu, chiamerà una funzione tra quelle precedentemente elencate. Le funzioni in causa sono:

1. **mostraECG()** - è la funzione che prepara la view per mostrare l'ECG. Prima

Figura 6.20: Dettaglio sulla pressione dello ShowECG VC

di tutto viene salvato un identificativo della schermata che verrà mostrata (questa azione è necessaria per il corretto ripristino della view al termine della generazione del pdf, come spiegato al capitolo Cap. 5.7.1). Segue la modifica dei colori dei bottoni mostrati nel menu in basso: il bottone della view che verrà mostrata è più scuro rispetto agli altri, per simulare la sua "pressione", come fosse un pulsante incassato. Successivamente la linechartview viene resa visibile e i suoi dati sono inizializzati con quelli preparati al punto precedente per l'ecg. Le labels sono rese visibili e riempite con i dati sull'HB e sulla fibrillazione. La view con i dati sulla pressione è nascosta (nel caso fosse visibile precedentemente) e il linechart è configurato.

2. **mostraPPG()** - è la funzione che prepara la view per mostrare il PPG. Anche in questo caso viene salvato l'identificativo della nuova schermata, viene scurito il tasto PPG del menu e schiariti gli altri due per simulare la pressione, viene preparata la view del grafico configurandone i giusti dati di input e vengono aggiornate le labels (evidenziando eventuali anomalie). Tutte le view non necessarie in questo momento vengono nascoste.

3. **mostraPressione()** - è la funzione che prepara la view per mostrare il dettaglio sulle pressioni. Anche in questo caso viene salvata la nuova schermata, viene evidenziata con i colori la pressione sul giusto bottone, viene nascosta la linechartview, viene mostrato lo SV delle pressioni. Le labels sono aggiornate e le eventuali anomalie segnalate.

Nell'angolo in alto a destra della schermata sono presenti tre bottoni che permettono all'utente di ottenere diversi servizi: la cancellazione del file corrente; la sua esportazione; la consultazione di una guida alla lettura dei segnali.

La **funzione di cancellazione**, action scatenata dall'evento di pressione del bottone raffigurante il bidone della spazzatura, utilizza il path del file ricevuto dal segue per cancellarlo dalla memoria del telefono.

Il **bottone delle informazioni** performa un segue all'InfoECG, un View Controller che non è stato raccontato nel dettaglio, in quanto estremamente semplice. Questo VC è composto da una Scroll View in cui sono stati inseriti il testo e le immagini necessarie per spiegare come leggere i segnali. La sua view presenta anche un bottone per dismettere il VC e tornare allo ShowECG VC.

Il **bottone di esportazione** è quello a cui è connesso più codice. La funzione di action mostra un alert e chiama, in base alle scelte prese dall'utente, altre funzioni annidate che mostreranno un secondo alert. Il primo alert chiede all'utente il formato d'esportazione del file, tra pdf e json. Il secondo offre due possibilità: inviare il file come allegato di una email autogenerata destinata al medico (scelta rapida per comunicare con il dottore); esportare il file mediante gestore d'esportazione di sistema. Nel secondo caso sarà possibile scegliere come esportare il file, cioè se lo si vuole salvare nel filesystem del dispositivo o se lo si vuole aprire con una nuova applicazione, che può essere di modifica oppure di condivisione (come Whatsapp o Telegram). Per quanto riguarda l'esportazione in formato pdf, per prima cosa verrà chiamata la funzione generatrice del file che è stata analizzata al capitolo 5.7.1 e poi, in base al metodo d'esportazione scelto dall'utente, verrà istanziato un ActivityVC oppure un MessageComposerVC, come illustrato nel capitolo 5.7. Nel caso il formato fosse il Json invece, prima di tutto è necessario verificare se il file esista già nella memoria dell'applicazione. Infatti, se stiamo visualizzando una misurazione appena acquisita, e se l'utente non aveva precedentemente configurato l'app in modo che il salvataggio avvenga in automatico, il file non è mai stato creato, e l'unico riferimento alla misurazione che possediamo è appunto l'oggetto Misurazione. In questo caso, il file json viene scritto nella cache a partire da tale oggetto, così che se ne perderà memoria alla chiusura dell'applicazione. Infine il file è dato come parametro all'Activity VC o al MessageComposerVC in base al tipo di esportazione selezionato.

In fase di esportazione, qualore ad un interrogazione dell'oggetto UserDefaults, l'applicazione scopra che l'utente ha abilitato la condivisione della posizione, sarà necessario allegare quest'ultima all'email ed al pdf. Lo ShowECG VC ha una proprietà del tipo LocationManager ed estende il suo delegate, in modo da gestire le posizioni come spiegato al capitolo 5.5.

Capitolo 7

Back-End

In questo capitolo verrà affrontata l'ultima parte relativa all'implementazione in codice Swift dell'applicazione PulsEcg: il back-end. Il **back-end** è costituito da tutte quelle classi che si occupano, attraverso complesse procedure, di offrire servizi al front-end, ovvero all'interfaccia utente e alla porzione di codice che ne aggiorna la vista.

I grossi blocchi dell'applicazione PulsEcg che fanno parte del back-end sono: la classe di gestione del bluetooth; la classe in cui sono raccolte le funzioni d'elaborazione dei segnali acquisiti dal bracciale; le funzioni di calcolo dei parametri vitali derivati dai segnali, quali l'HB e l'SpO2, e di segnalazione delle relative anomalie; la classe wrapper della rete neurale, che si occupa di importare il modello TensorFlow, di preprocessare i dati, di darli in pasto alla rete e di elaborarne l'output per la determinazione dei valori di pressione e per la segnalazione di valori fuori soglia.

7.1 Classe BleManager

La comunicazione Bluetooth è stata gestita tra bracciale e applicazione è stata gestita da una classe, chiamata BleManager, priva di interfaccia grafica. Questa classe opera in modalità nascosta e interagisce con diversi View Controllers sfruttando un meccanismo di notifica asincrona, il Notification Center. Le notifiche vengono mandate agli osservatori dei View Controllers, non appena i dati della classe BleManager sono pronti per essere visualizzati o per determinare lo scatenarsi di alcuni eventi.

La classe BleManager è **Singleton**. Questo significa che tutti i suoi riferimenti nei vari View Controllers puntano ad una stessa istanza del BleManager. La libreria necessaria alla sua implementazione è CoreBluetooth che, come affrontato nel capitolo 5.4.1, offre diversi metodi per la gestione della comunicazione BLE con le periferiche.

BleManager funge da **wrapper dell'oggetto CBCentralManager**, classe di CoreBluetooth che si occupa dello stato del bluetooth di sistema, dello scan, della connessione/disconnessione a periferica, dell'esplorazione di servizi e caratteristiche e dell'uso delle caratteristiche per la comunicazione. Oltre a proteggere il manager del bluetooth, la classe è decorata da tutti gli attributi utili al caso: la periferica attualmente connessa; le periferiche trovate in fase di scan; dei vettori e degli indici/offset per salvare l'acquisizione dei campioni ricevuti; dei flag di sincronizzazione; la lista di caratteristiche trovate nel dispositivo. Inoltre, tutta la logica implementativa si rifà ad una serie di costanti contenute nella struttura BleConstants.

I VC possono chiamare solo i metodi pubblici del wrapper. Questi metodi si dividono in due categorie:

1. metodi che fungono semplicemente da getter del risultato dell'acquisizione o dello stato della connessione;

2. metodi per avviare o interrompere lo scan, mandare il comando di inizio acquisizione, ottenere informazioni sulla connessione e ricevere i risultati di una acquisizione. Tutto ciò avviene perchè i metodi pubblici nascondono internamente la logica necessaria a scatenare l'esecuzione delle callback della libreria CoreBluetooth.

Tali callback, la cui funzione è introdotta nel capitolo 5.4.1, sono state customizzate come segue:

1. **writeCommandStartAcquisition()** (Cod. 7.1)- è il metodo che scrive il comando di inizio acquisizione nella caratteristica dedicata e che inizializza a questo scopo tutti i parametri interni alla classe. Il metodo è chiamato dall'AquireECG Activity.

Listing 7.1: Implementazione delle callback writeCommandStartAcquisition()

```
func writeCommandStartAcquisition(){
        var dataToSend = Data()
        dataToSend.append(BLEConstants.COMMAND_WRITE_START_ACQ)
        let char = foundChar[BLEConstants.COMMAND_WRITE_UUID]

        expectedSeqNumECG = 0
        offsetECG = 0;
        offsetPPG[0] = 0
        offsetPPG[1] = 0
        expectedSeqNumPPG = 0;
        ECGSamples = []
        PPGSamplesIR = []
        PPGSamplesRED = []
        complete[0] = false
```

```
15          complete[1] = false
16
17          watch?.writeValue(dataToSend, for: char!, type:
    CBCharacteristicWriteType.withResponse)
18      }
19
```

2. **centralManagerDidUpdateState** - è il metodo la cui esecuzione è scatenata al variare dello stato del CentralManager. In particolare, quando lo stato è on viene avviato lo scan delle periferiche, quando è off vengono notificati i VC.

3. **didDiscoverPeripheral** (Cod. 7.2) - chiamata ogni volta che viene trovata una nuova periferica. Se la periferica è un PulsEcg, viene aggiunta ad un array interno alla classe. A questo punto viene mandata una notifica al VC ConnectBluetoothDevice. Se questo è stato aperto dall'utente, l'osservatore del VC interrogherà l'istanza della classe BleManager per ricevere il vettore di periferiche e visualizzarlo nella sua tabella, aggiornando la view. La ricerca delle periferiche ha inizio quando viene instanziata la classe BleManager, perché nel suo costruttore è nascosto l'init dell'oggetto Central Manager.

Listing 7.2: Implementazione delle callback didDiscoverPeripheral()

```
1   func centralManager(_ central: CBCentralManager, didDiscover
    peripheral: CBPeripheral, advertisementData: [String : Any],
    rssi RSSI: NSNumber) {
2       if peripherals.contains(peripheral) == false{
3           if peripheral.name != nil && peripheral.name!.
    contains("ECG"){
4               //se la periferica è nuova e si chiama ECG, la
    salvo nell'array di periferiche trovate
5               peripherals.append(peripheral)
6               //notifico il VC ConnectBluetoothDevice così che
    aggiorni la lista di periferiche trovate
7               NotificationCenter.default.post(name: .
    newPeripheralFound, object: peripheral)
8           }
9       }
10      //gestione di connessione automatica alla periferica
    associata
11      let defaults = UserDefaults.standard
12      let bledev = defaults.object(forKey: "BLEDevice") as?
    String ?? String()
13      if bledev.isEmpty || bledev == "" {
14          return
15      }
16      else {
17          if peripheral.name != bledev {
18              return
```

```
19              }
20              else {
21                  //se la periferica che ho trovato coincide con
    quella associata allora avvio la connessione
22                  //altrimenti non faccio nulla
23                  currentDiscoveredDevice = peripheral
24                  centralManager!.connect(currentDiscoveredDevice!,
     options: nil)
25              }
26          }
27      }
28
```

4. **didDisconnectPeripheral** - chiamata per via di una disconnessione volontaria o involontaria dalla periferica. Manda una notifica a tutti i VC che mostrano graficamente l'assenza di una connessione o che devono impedire certe azioni (ad esempio l'HomePage non permette di avviare una acquisizione senza un dispositivo connesso).

5. **didConnect** - è chiamata in due situazioni: dalla didDiscoverPeripheral nel caso in cui abbia trovato la periferica associata; al click di una cella della tabella del ConnectBluetoothDevice VC, che richiamerà un metodo pubblico della classe BleManager in grado di avviare la connessione con la periferica selezionata. La connessione, per poter avvenire, necessita una esplorazione dei servizi e delle caratteristiche. Verranno infatti chiamate sia la didDiscoverServices che la didDiscoverCharacteristics che verificheranno la disponibilità dei servizi e delle caratteristiche attese nel dispositivo PulsEcg.

6. **didUpdateValueFor characteristic** (Cod. 7.3)- è il metodo cuore dell'acquisizione. Il dispositivo manda diversi pacchetti. Il primo è della batteria, gli altri sono ECG e PPG misti. Qualunque pacchetto arrivi è intercettato da questa funzione. Il metodo discrimina i pacchetti in base alla caratteristica da cui li ha ricevuti e gestisce i casi separatamente. Ogni pacchetto contiene solo un certo numero di campioni che andranno a riempire progressivamente i vettori della classe. Quando l'acquisizione è completa perché sono stati ricevuti tutti i pacchetti, entrambi i flag di completamento sono posti a true e l'AquireECG VC è notificato, così che possa filtrare i segnali, calcolare HB, SpO2 e pressioni, salvare il file e visualizzarlo spostandosi con un segue al VC GeneralShowECG.

Listing 7.3: Implementazione delle callback didUpdateValueFor characteristic()

```
1   func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor
    characteristic: CBCharacteristic, error: Error?) {
2       let data = characteristic.value
```

```
3           var pacchetto = [UInt8]()
4           pacchetto.append(contentsOf: data!)
5           switch characteristic.uuid {
6           //Ho ricevuto pacchetto ECG
7           case CBUUID(string: BLEConstants.ECG_PAYLOAD_CHAR_UUID):
8               if data?.count != BLEConstants.ECGpayloadSize {
9                   print("Dimensione del pacchetto ECG errato --> PACCHETTO PERSO")
10                  return;
11              }
12              var i : Int = 0
13              //leggo e decodifico tutti i campioni --> li metto nell'array ECGSamples
14              while i < data!.count {
15                  var bytes : [UInt8] = []
16                  bytes.append(data![i])
17                  bytes.append(data![i+1])
18                  let u16 = UnsafePointer(bytes).withMemoryRebound(to: UInt16.self, capacity: 1) {
19                      $0.pointee
20                  }
21                  ECGSamples.append(u16)
22                  offsetECG = offsetECG + 1
23                  i = i + 2
24              }
25              //incremento il numero di campioni ricevuti e controllo se li ho finiti
26              //se si, setto il flag di completamento
27              //se anche i PPG sono finiti, notifico la AquireECG
28              expectedSeqNumECG = expectedSeqNumECG + 1
29              if offsetECG == Constants.NO_OF_SAMPLES {
30                  complete[0] = true
31                  if complete[1] == true {
32                      complete[0] == false
33                      NotificationCenter.default.post(name: .acquisitionComplete, object: nil)
34                  }
35              }
36              break;
37
38          //Ho ricevuto pacchetto Batteria
39          case CBUUID(string: BLEConstants.BATTERY_CHAR_UUID):
40              NotificationCenter.default.post(name: .acquisitionStart, object: nil)
41              var bytes1 : [UInt8] = []
42              var bytes2 : [UInt8] = []
43              bytes1.append(data![0])
44              bytes1.append(data![1])
45              bytes2.append(data![2])
```

```
46              bytes2.append(data![3])
47              var vcell = UnsafePointer(bytes1).withMemoryRebound(
    to: UInt16.self, capacity: 1) {
48                  $0.pointee
49              }
50              vcell = vcell * UInt16((78.125e-6))
51              var socbattery = UnsafePointer(bytes2).
    withMemoryRebound(to: UInt16.self, capacity: 1) {
52                  $0.pointee
53              }
54              socbattery = socbattery / 256
55              NotificationCenter.default.post(name: .
    newBatteryValue, object: String(socbattery))
56              break;
57
58          //Ho ricevuto pacchetto PPG
59          case CBUUID(string: BLEConstants.PPG_PAYLOAD_CHAR_UUID):
60              print("Ho ricevuto pacchetto da PPG numero " + String
    (expectedSeqNumPPG) )
61              if characteristic.value?.count != BLEConstants.
    PPGpayloadSize{
62                  print("Dimensione del pacchetto PPG errato -->
    PACCHETTO PERSO")
63                  return;
64              }
65              var i : Int = 0
66              //Leggo tutti i campioni del pacchetto, li decodifico
67              //discrimino RED da IR e inserisco nel giusto vettore
68              while i < data!.count {
69                  var bytes : [UInt8] = []
70                  bytes.append(data![i])
71                  bytes.append(data![i+1])
72                  bytes.append(data![i+2])
73                  bytes.append(data![i+3])
74                  let temp = Data(bytes: bytes)
75                  var u32 = UInt32(littleEndian: temp.
    withUnsafeBytes { $0.pointee })
76                  let tag = u32 >> BLEConstants.ppgDatasize
77                  u32 = u32 & UInt32(BLEConstants.ppgMask)
78                  if tag == BLEConstants.ppgTypeIR {
79                      PPGSamplesIR.append(u32)
80                      offsetPPG[0] = offsetPPG[0] + 1
81                  }
82                  else if tag == BLEConstants.ppgTypeRed {
83                      PPGSamplesRED.append(u32)
84                      offsetPPG[1] = offsetPPG[1] + 1
85                  }
86                  else {
87                     // print("PPG tag sbagliato")
```

```
88                  }
89                  i = i + 4
90              }
91              //incremento il numero di campioni ricevuti e
    controllo se li ho finiti
92              //se si, setto il flag di completamento
93              //se anche i PPG sono finiti, notifico la AquireECG
94              expectedSeqNumPPG = expectedSeqNumPPG + 1
95              if expectedSeqNumPPG == 200 {
96                  complete[1]=true
97                  if complete[0] == true {
98                      complete[1] == false
99                      print("Ho finito da ppg")
100                     NotificationCenter.default.post(name: .
    acquisitionComplete, object: nil)
101                 }
102             }
103
104             break;
105         default:
106             return
107         }
108
109     }
110 }
111
```

7.2 Elaborazione dei segnali

Uno dei problemi più impegnativi nell'elaborazione dei segnali digitali è quello di riuscire a ricevere campioni informativi senza alcuna perdita. E' estremamente importante ridurre il rumore casuale e migliorare le prestazioni del segnale. Il segnale digitale non è un fenomeno naturale, ma viene generato per conversione del segnale analogico. Il segnale analogico viene campionato uniformemente ad una certa frequenza di campionamento, e poi viene quantizzato e codificato. Quando il segnale viene trasmesso al sensore/elettrodo d'acquisizione, subisce alcune perturbazioni casuali dovute all'ambiente. Qualcosa di simile avviene durante il processo di conversione da analogico a digitale. Questa perturbazione è normalmente distribuita nel tempo ed è chiamata **rumore gaussiano bianco additivo**. La somma tra questo ed il segnale originale rende le informazioni non analizzabili, perché la qualità dell'informazione è persa. Ciò che bisogna fare è estrarre le informazioni originali dal segnale sporco. Si tratta di un lavoro complesso perché il rumore che altera il segnale non è prevedibile nell'intensità e a volte può essere addirittura più potente del segnale che si è volutamente trasmesso. Lo scopo principale dei

filtri nell'elaborazione del segnale digitale è proprio quello di ridurre il rumore ed estrarre le informazioni desiderate. I filtri non lasciano passare alcune frequenze, ovvero, in altre parole, estraggono solo le frequenze utili all'analisi del segnale. [37].

I segnali ECG e PPG, allo stato in cui si trovano nel momento in cui sono ricevuti dalla classe BleManager, sono per l'appunto inutilizzabili per via di un forte rumore. L'elaborazione dei segnali comprende operazioni di conversione di tipo, deamplificazione, detrending e filtraggio.

Tutte le funzioni e le costanti necessarie a questo scopo sono state raccolte nella classe **SignalProcessing**. Nei prossimi paragrafi saranno illustrate le tecniche e gli algoritmi impiegati.

7.2.1 Elaborazione ECG

Se visualizzassimo il segnale ECG così come è stato acquisito dal bracciale, sarebbe praticamente illeggibile a causa del rumore (Fig. 7.1). Il rumore può essere rimosso grazie ad alcuni step di manipolazione dei campioni, realizzati ad opera delle funzioni contenute dalla classe SignalProcessing.

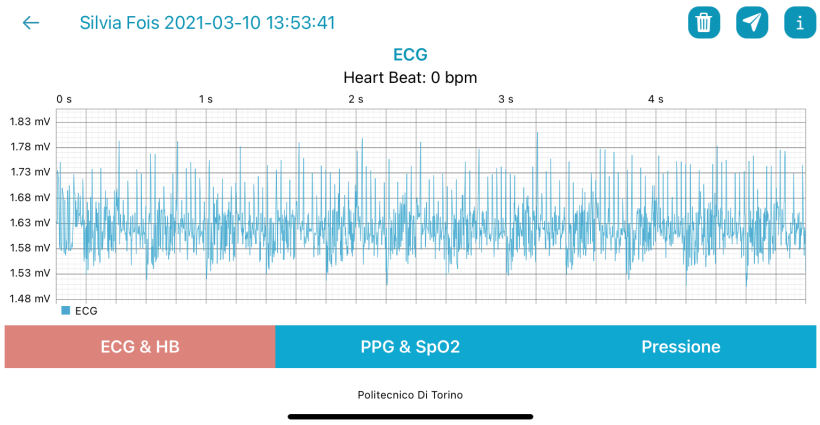


Figura 7.1: Vecchio screenshot del segnale ECG, acquisito quando l'app non era ancora munita della logica necessaria all'elaborazione dei segnali.

L'elaborazione del segnale ECG può essere suddivisa in due fasi, quella di preprocessamento e quella di filtraggio. Al termine del processamento, i campioni possono essere salvati nel file, ma solo dopo il filtraggio potranno essere visualizzati. Per essere utilizzati dalla rete i segnali devono essere preprocessati, filtrati e poi dovranno subire un altro step di elaborazione da parte della classe NeuralNetworkManager (Cap. 7.4).

Preprocessing

Il codice contenente le funzioni di preprocessing descritte nel seguente elenco è allegato in 7.4. Le funzioni sono riportate nell'ordine in cui vengono chiamate:

1. **conversione in Double**. Ciò è necessario perché le funzioni della classe SignalProcessing lavorano su vettori di Double, mentre l'ECG è stato ricevuto dall'applicazione sotto forma di array di Uint16 (perché il BLE invia pacchetti composti da byte che vanno poi interpretati a coppie).

2. **eliminazione dell'influenza dell'hardware** di acquisizione sul segnale. Ciò è realizzato deamplificando il segnale di un fattore 1650 e successivamente dividendolo per un fattore 750.

3. **inversione del segnale sull'asse y=1.65**, da effettuare solo nel caso in cui l'acquisizione fosse avvenuta indossando l'orologio sul **braccio sinistro**, invece che sul destro. Per invertire un segnale rispetto ad un certo asse y, è sufficiente ricalcolare i campioni come differenza tra il fattore 2*y e il valore originale.

Listing 7.4: Preprocessing dell'ECG

```
//conversione in double
func fromUInt16ToDouble(segnale : [UInt16]) -> [Double] {
    var result : [Double] = []
    for v in segnale {
        result.append(Double(v))
    }
    return result
}
//inversione sull'asse y=1.65
func inversioneAsseYECG(segnale : [Double] ) -> [Double]{
    let asseY : Double = 1.65
    let fattore = 2*asseY
    var segnaleInvertito : [Double] = []
    for i in 0 ..< segnale.count {
        segnaleInvertito.append(fattore-segnale[i])
    }
    return segnaleInvertito
}
//deamplificazione
func deamplifica(segnale: [Double]) -> [Double] {
    var risultato : [Double] = []
    for i in 0 ..< segnale.count {
        var temp = segnale[i]-1650
        risultato.append(temp/750)
    }
    return risultato}
```

Filtraggio

Per filtrare il segnale ECG, è stato utilizzato un **filtro Notch IIR di ordine 10 e fattore di qualità pari a 0.005**.

I filtri Notch (chiamati anche filtri elimina banda) vengono solitamente utilizzati in applicazioni che richiedono la reiezione di una banda di lunghezze d'onda elettromagnetica durante il passaggio di parti alte e basse dello spettro[38].

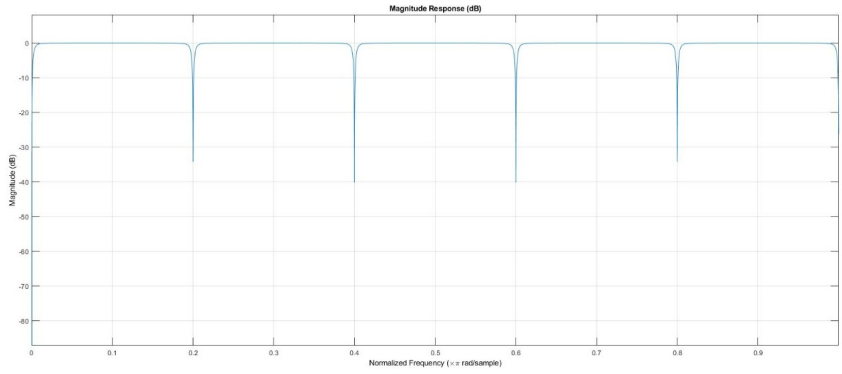


Figura 7.2: Filtro Notch usato da PulsEcg per rimuovere il rumore dal segnale ECG

Il filtro Notch utilizzato nella pulizia del segnale ECG è caratterizzato da una forma a "pettine". I lobi che si possono osservare nel suo grafico (Fig. 7.2) rappresentano le bande eliminate, equidistanti. L'ordine del filtro (10) ne determina il numero, ovvero la frequenza alla quale agiscono. Visto che la larghezza di banda è 500Hz, con un ordine 10 esse agiranno a 50Hz. Il fattore di qualità incide infine sulla larghezza dei lobi.

La funzione di trasferimento di un filtro Notch di questo tipo è riportata in Eq. 7.1, in cui n rappresenta l'ordine. Per convertire tale filtro in software Swift dobbiamo considerare che la funzione di trasferimento di un filtro **Infinite Impulse Response (IIR)** può essere implementata come una **cascata di blocchi biquad** [39]. Tale blocco calcola l'uscita con la formula 7.2. Il filtro può essere realizzato scrivendo una funzione biquadratica Swift che applichi questa trasformazione ad ogni campione del segnale. I valori dei coefficienti a e b di ogni blocco biquadratico in cascata vanno determinati utilizzando uno script Matlab che inizializzi il filtro con i parametri precedentemente illustrati. In questo modo sono stati ottenuti i coefficienti rappresentati in Tab. 7.1.

$$H(z) = b * \frac{(1 - z^{-n})}{(1 - a * z^{-n})} \tag{7.1}$$

$$y[n] = b_0 * x[n] + b_1 * x[n-1] + b_2 * x[n-2] - a_0 * y[n-1] - a_1 * y[n-2] \quad (7.2)$$

Tabella 7.1: Coefficienti dei cinque blocchi biquadratici costituenti il filtro Notch IIR

BiquadBlock	b0	b1	b2	a1	a2
1	0.9622	0	-0.9622	0	-0.9844
2	1	1.618	1	1.6054	0.9844
3	1	-1.618	1	-1.6054	0.9844
4	1	0.618	1	0.6132	0.9844
5	1	-0.618	1	-0.6132	0.9844

Il codice è stato quindi strutturato in due funzioni consultabili in Cod. 7.5. La funzione bquad modella un blocco biquadratico, ricevendo in input un segnale e il valore dei coefficienti. La funzione non fa altro che applicare la formula 7.2 ad ogni campione. La funzione notch_filter10 invece, modella il filtro completo, mettendo in cascata i 5 blocchi biquad. Il suo corpo chiama cinque volte di fila la funzione biquad, passandogli come argomenti i coefficienti ricavati con lo script matlab e schematizzati in Tab. 7.1.

Listing 7.5: Funzioni di implementazione del filtro notch

```
1     private func bquad(x: [Double], b0: Double, b1: Double, b2:Double
      , a1: Double, a2: Double)->[Double]{
2     var y : [Double] = []
3
4     var x_2 : Double = 0
5     var x_1 : Double = 0
6     var y_2 : Double = 0
7     var y_1 : Double = 0
8
9     for i in 0 ..< x.count {
10        let part1 = b0 * x[i] + b1 * x_1 + b2 * x_2
11        let part2 = a1 * y_1 + a2 * y_2
12        y.append(part1 - part2) // IIR difference equation
13        x_2 = x_1               // shift delayed x, y samples
14        x_1 = x[i]
15        y_2 = y_1
16        y_1 = y[i]
17    }
18    return y
19 }
20
21 func notchFilter_10(x: [Double])->[Double]{
```

```
22      var result : [Double] = []
23      result = bquad(x: x, b0: 0.9622, b1: 0, b2: -0.9622, a1: 0, a2:
        -0.9844)
24      result = bquad(x: result, b0: 1, b1: 1.618, b2: 1, a1: 1.6054, a2
        : 0.9844)
25      result = bquad(x: result, b0: 1, b1: -1.618, b2: 1, a1: -1.6054,
        a2: 0.9844)
26      result = bquad(x: result, b0: 1, b1: 0.618, b2: 1, a1: 0.6132, a2
        : 0.9844)
27      result = bquad(x: result, b0: 1, b1: -0.618, b2: 1, a1: -0.6132,
        a2: 0.9844)
28      return result
29  }
```

L'output del filtro Notch IIR viene poi dato in pasto ad un **filtro a media mobile con finestra 10**, per due volte consecutive.

Una media mobile è un tipo di filtro **Finite Impulse Response** utilizzato per eliminare il rumore dei segnali, creando una serie di medie applicate a diversi sottoinsiemi dei dati d'input. Data una serie di numeri e data una dimensione fissa del sottoinsieme, detta finestra, l'output di un campione a cui è applicata una media mobile si ottiene calcolando il valor medio del sottoinsieme di cui fa parte il campione. Quindi il sottoinsieme viene modificato "spostandosi in avanti", ovvero escludendo il primo numero della serie e includendo il numero successivo che segue il sottoinsieme originale nella serie. Questo crea un nuovo sottoinsieme di numeri, che viene mediato. Questo processo viene ripetuto per l'intera serie di dati. La dimensione della finestra è molto importante per un accurato denoising dei segnali [40].

Poiché il filtro a media mobile verrà utilizzato anche in successive elaborazioni, scegliendo ogni volta una ampiezza della finestra che si adatti al caso, la funzione Swift che la implementa (Cod. 7.6) è stata scritta perché sia indipendente da tale valore, ricevuto come parametro insieme al segnale da filtrare. Il codice scandisce il segnale di input e per ogni cella con indice i del vettore produce una cella con indice i dell'array di output, calcolando la media del sottoinsieme di valori contenuti nella finestra di ampiezza w, ovvero nelle celle del vettore comprese tra l'indice *i-w/2* e l'indice *i+w/2*. Ovviamente, in base alla cella del vettore che si sta esplorando, si può ricadere in tre casi:

1. $$i <= w/2$$

   In tal caso il sottoinsieme di dati di cui calcolare la media è composto dalla metà di destra della finestra, dalla cella all'indice i, e da quante celle sono disponibili alla sua sinistra (valore che va da 0 a w-1).

2. $$(i > w/2) \cap (i + w/2 < nSamples)$$

   In tal caso il sottoinsieme di dati di cui calcolare la media è composto dalla metà di destra della finestra, dalla cella all'indice i, e dalla metà di sinistra della finestra.

3. $$i + w/2 >= nSamples$$

   In tal caso il sottoinsieme di dati di cui calcolare la media è composto dalla metà di sinistra della finestra, dalla cella all'indice i, e da quante celle sono disponibili alla sua destra (valore che va da 0 a w-1).

Al termine del filtraggio il segnale ECG è finalmente riconoscibile. Si è preferito non pulire il segnale in maniera troppo invasiva, in modo da non perdere l'informazione sottostante, al contrario di altri competitor sul mercato che prediligono un segnale più pulito ma meno significativo a livello medico, perché porta con se meno informazione.

Listing 7.6: Filtro a media mobile

```
func mediaMobile(segnale: [Double], finestra: Int)->[Double]{
    //in realtà la media è fatta su finestra+1 elementi, perchè ne
   prendo finestra/2 prima e finestra/2 dopo
    let metaFinestra = finestra/2
    var segnaleMediaMobile : [Double] = []
    for i in 0 ..< segnale.count {
        var acc = segnale[i]
        var nDiv : Double = 0
        var result : Double = 0
        if i<metaFinestra {
            //caso in cui mi mancano valori nella prima metà della
   finestra
            var counter : Double = 0
            for j in 0 ..< (i+metaFinestra+1) {
                if i != j {
                    acc = acc + segnale[j]
                }
                counter = counter + 1
            }
            nDiv = counter
            result = acc / nDiv
        }
        else if i > segnale.count-metaFinestra-1 {
            //caso in cui mi mancano valori nella seconda metà della
   finestra
            var counter : Double = 0
            for j in (i-metaFinestra) ..< segnale.count {
```

```
                if i != j {
                    acc = acc + segnale[j]
                }
                counter = counter + 1
            }
            nDiv = counter
            result = acc / nDiv
        }
        else {
            //caso in cui ho tutti i w+1 valori
            for j in (i-metaFinestra) ..< (i+metaFinestra+1) {
                if i != j {
                    acc = acc + segnale[j]
                }
            }
            nDiv = Double(finestra + 1)
            result = acc / nDiv
        }
        segnaleMediaMobile.append(result)
    }
    return segnaleMediaMobile
}
```

7.2.2 Elaborazione PPG

Anche il segnale PPG, sia rosso che infrarosso, così come quello ECG, ha bisogno di una fase di elaborazione suddivisa in preprocessamento e filtraggio.

Il **preprocessing** consiste in:

1. **conversione in Double**. Ciò è necessario perché le funzioni della classe SignalProcessing lavorano su vettori di Double, mentre il PPG è stato ricevuto dall'applicazione sottoforma di array di Uint32 (perchè il BLE invia pacchetti composti da byte che vanno poi interpretati a gruppi da 4).

2. **eliminazione dell'influenza dell'hardware** di acquisizione sul segnale. Ciò è realizzato invertendo il segnale rispetto all'asse y=0. Per invertire un segnale rispetto ad un certo asse y, come abbiamo già visto in precedenza, è sufficiente ricalcolare i campioni come differenza tra il fattore 2*y e il valore originale.

Il codice di implementazione di queste due funzioni è stato omesso, perchè estremamente simile a quello visto nel preprocessing dell'ECG.

Dopo il preprocessamento, la fase di filtraggio è realizzata applicando, sia al PPG Red che al PPG IRed:

1. **due filtri a media mobile con finestra w=20**, applicati in cascata. La funzione chiamata per realizzare questo filtraggio è la stessa di cui è stata spiegata l'implementazione nel capitolo precedente. Il suo codice è riportato in Cod. 7.6.

2. un'operazione di **detrending**, ovvero di eliminazione delle tendenze lineari del segnale. Il detrending (Cod. 7.7) si realizza: calcolando il valor medio del segnale (ovvero l'asse y che lo attraversa per metà in altezza); calcolando l'output di una media mobile con finestra 1200 del segnale; sottraendo ad ogni campione il valore della media mobile e sommandogli il valor medio. Il problema di questa funzione è la **lentezza della media mobile**. Infatti la funzione di media mobile ha complessità quadratica, dovuta ai due cicli for annidati. Questo costo non incide sulle performance dell'applicazione quando l'ampiezza della finestra è piccola, ma diventa rilevante con un valore alto come quello richiesto dal detrending (=1200). Questa funzione è lenta e inoltre deve essere chiamata due volte di fila (una per il PPG red e una per il PPG ir), perciò finisce per diventare il collo di bottiglia dell'elaborazione dei dati, tanto da richiedere una schermata di loading di 2 secondi. In una successiva versione dell'applicazione, questo problema può essere risolto trasformando la funzione di media mobile in una procedura ricorsiva.

Listing 7.7: detrending

```
1
2  func detrending(segnale: [Double])->[Double] {
3      var meanedSig : [Double] = mediaMobile(segnale: segnale,
       finestra: 1200)
4      var mean = segnale.avg()
5      var risultato : [Double] = []
6
7      for i in 0 ..< segnale.count {
8          risultato.append(segnale[i]-meanedSig[i]+mean)
9      }
10     return risultato
11 }
```

7.3 Calcolo di HeartBeat e SpO2

Giunti a questo punto, i segnali sono stati preprocessati e filtrati, quindi sono pronti per poter essere da un lato visualizzati e dall'altra analizzati per la derivazione di altri parametri vitali e per la conseguente segnalazione di eventuali anomalie.

7.3.1 HB e Fibrillazione Atriale

Il segnale elettrocardiografico è utilizzato sia per il calcolo numero di battiti al minuto (HB) che per la rilevazione della fibrillazione atriale.

Il **numero di battiti al minuto** può essere calcolato a partire da un segnale ECG attraverso i suoi **picchi**, i quali devono essere rilevati usando **l'algoritmo di Pan–Tompkins** [41]. Prima di tutto è stata quindi definita una funzione che implementasse tale algoritmo. Per trovare i picchi locali, l'algoritmo (Cod. 7.8) cerca iterativamente il massimo assoluto di una finestra di 150 ms fatta traslare di volta in volta. In seguito controlla se la zona del segnale precedente e successiva al massimo hanno un andamento rispettivamente crescente e descrescente. Questo controllo viene effettuato verificando che la differenza tra il picco ed un valore ad esso precedente o successivo sia minore del 35%-40% rispetto al picco massimo totale. Se il picco precedente è stato trovato in un tempo <300ms, allora i due picchi sono confrontati per capire quale possa essere quello corretto. Se tutte le richieste vengono soddisfatte, il picco è aggiunto definitivamente alla lista.

Listing 7.8: Funzioni per la rilevazione dei picchi dell'ECG

```
func averaged_max(input : [Double])->Double {
    var absolute_max : Double = 0
    let length : Int = input.count;
    let step : Int = length/Constants.MAXIMA_PEAK_NUMBER;
    var max : [Double] = [Double](repeating: 0, count: Constants.
   MAXIMA_PEAK_NUMBER)
    //Find 10 local maxima for each tenth of signal
    for j in 0..<Constants.MAXIMA_PEAK_NUMBER{
        for i in j*step ..< step*(j+1){
            if input[i] > max[j]{
                max[j] = input[i]
            }
        }
    }
    // defining the threshold level for the R-peaks
    max.sort()      // sorting in ascending order
    for i in 3..<7 {  // averaging
        absolute_max = absolute_max + max[i]
    }
    absolute_max = absolute_max/4
    return absolute_max;
}

func findPeak(input : [Double], GVDataPeaks : inout [ChartDataEntry],
    absolute_max : Double) {
    var GVDataPeak : [ChartDataEntry] = [ChartDataEntry](repeating:
   ChartDataEntry(x: 0, y: 0), count: 4)
    let length : Int = input.count
```

```
26      let interval : Int = 150
27      var maxima : [Int] = [Int](repeating: 0, count: length/200)
28      var temp_max : Int
29      var k : Int = 0
30
31      // FIND PEAKS
32      var i : Int=0
33      while (i*interval) < length {
34          i = i + 1
35          temp_max = (i-1)*interval
36          var limite : Int
37          if length < (i)*interval {
38              limite = length
39          }
40          else{
41              limite = (i)*interval
42          }
43          for j in ((i-1)*interval) ..< limite{
44              if input[j] > input[temp_max] {
45                  temp_max = j;
46              }
47          }
48          //a peak cannot be at the beginning
49          if temp_max < 35 ||
50              temp_max > Constants.NO_OF_SAMPLES - 35 ||
51             (input[temp_max]-input[temp_max+30]) < (absolute_max-1.5)
    *0.40 ||
52             (input[temp_max]-input[temp_max-20]) < (absolute_max-1.5)
    *0.35 {
53                  continue;
54          }
55          if k>0 {
56              if temp_max-maxima[k-1] < 300 {// Two peaks cannot be too
     close
57                  let slope_temp : Double = (input[temp_max]-input[
    temp_max+30])
58                  let slope_old : Double = (input[maxima[k-1]]-input[
    maxima[k-1]+30]);
59                  if (slope_temp>slope_old){
60                      maxima[k-1]=temp_max
61                      GVDataPeaks[k-1] = ChartDataEntry(x: Double(
    temp_max), y: input[temp_max])
62                  }
63                      continue;
64              }
65          }
66          maxima[k] = temp_max
67          GVDataPeaks[k] = ChartDataEntry(x: Double(temp_max), y: input
    [temp_max])
```

```
68          k = k + 1
69      }
70      while k < GVDataPeaks.count {
71          if k==0 {
72              GVDataPeaks[0] = ChartDataEntry(x: 0, y: 0)
73              GVDataPeaks[1] = ChartDataEntry(x: 0, y: 0)
74              return;
75          }
76          GVDataPeaks[k] = ChartDataEntry( x: GVDataPeaks[k-1].x, y:
    GVDataPeaks[k-1].y)
77          k = k+1
78      }
79  }
```

La funzione successiva (Cod. 7.9) si occupa sia della derivazione del HB che della rilevazione della FA, utilizzando i picchi calcolati al punto precedente.

Sia il battito per minuto che il ritmo complessivo vengono analizzati, per verificare se le loro variazioni nel tempo superano le soglie predefinite (gli esperimenti hanno permesso di indentificare una soglia pari a 3 bpm). In caso positivo, la registrazione è classificata come fibrillante. In caso negativo, il ritmo è considerato normale. Viene quindi eseguito un controllo finale sull'onda P. Infatti, in caso di fibrillazione atriale, le onde P saranno assenti. Tuttavia, alcune persone con fibrillazione atriale avranno onde fibrillatorie con andamento simil-sinusoidale, che ricordano l'onda P. Per questo motivo, il blocco finale dell'algoritmo cerca le onde P per mezzo dei massimi antecedenti il picco R. Quando un'onda simile viene trovata, la sua ampiezza, durata e distanza dai complessi QRS precedenti e successivi vengono controllati per determinare se si tratta di un'onda P vera o fibrillatoria [6].

Listing 7.9: Funzione per il calcolo dell'HB a partire dai picchi dell'ECG

```
1  func heartBeatRate(dataPoint : [ChartDataEntry]) -> [Int]{
2      var min : Int = 0
3      var max : Int = 0
4      var sum : Int = 0
5      var RR : Int
6      var i : Int
7      var heart_rate : [Int] = [0, 0, 0, 0]
8
9      //added for FA detection
10     var FAbpmCounter : Int = 0 //number of FA beats of the entire
    acquisition
11     var previousBpm : Int = 0 // end FA
12     i = 1
13     while dataPoint[i].x != 0 && dataPoint[i].x != dataPoint[i-1].x {
14         if i==1 {
15             sum = Int(dataPoint[i].x) - Int(dataPoint[i-1].x)
16             min = Int(dataPoint[i].x) - Int(dataPoint[i-1].x)
17             max = min;
```

```
18              //added for FA detection
19              previousBpm = Constants.HEARTBEAT/min
20              // end FA
21          }
22          else{
23              RR = Int(dataPoint[i].x-dataPoint[i-1].x)
24              sum = sum + RR
25              if RR < min {
26                  min = RR
27              }
28              if RR > max {
29                  max = RR
30              }
31              //added for FA detection
32              if (Constants.HEARTBEAT/RR) < previousBpm - 5 ||
33                  (Constants.HEARTBEAT/RR) > previousBpm + 5 {
34                  FAbpmCounter = FAbpmCounter + 1
35              }
36              previousBpm = Constants.HEARTBEAT/RR
37              // end FA
38          }
39          i = i+1
40      }
41      sum = sum / i //average RR time
42      i = i - 1 // restore the last increment (now i is the number of R
    -R interval)
43      if i<5 || i > Constants.SECONDS_OF_AQUISITION * 4 {// heart rate
    not possible maxBPM = 240
44          heart_rate[0] = -1// signaling error
45          heart_rate[1] = -1// signaling error
46          heart_rate[2] = -1// signaling error
47      }
48      else {
49          heart_rate[0] = Int( Double(Constants.HEARTBEAT) * 0.88 ) /
    sum
50          heart_rate[1] = Constants.HEARTBEAT/max //average beat
51          heart_rate[2] = Constants.HEARTBEAT/min  //average beat
52          //added for FA detection
53          if FAbpmCounter > (i - FAbpmCounter) {
54              heart_rate[3]=1;
55          }
56          // end FA
57      }
58      return heart_rate;
59  }
```

7.3.2 SpO2 e segnalazione delle anomalie

Come spiegato nel capitolo 1.1.2, l'**SpO2** può essere calcolata a partire dalle **componenti DC e AC dei segnali pletismografici** acquisiti dal fotorilevatore a seguito della riflessione dei led rosso e infrarosso. Per ognuno dei due segnali viene calcolato il rapporto tra componenti e poi i due segnali sono messi in relazione con il rapporto dei rapporti (Eq. 1.2). Il rapporto dei rapporti *Rtot* è poi moltiplicato per un fattore 17, ed il risultato è sottratto al valore 104 (come indicato in Eq. 1.3) per ottenere la percentuale di SpO2 nel sangue.

La componente DC di un segnale PPG è composta da campioni che corrispondono ai suoi minimi locali. I campioni della componente AC sono invece ottenuti sottraendo ad ogni massimo locale del segnale PPG il suo corrispondente minimo. Una volta ottenuti questi due vettori, l'algoritmo precedentemente descritto è completo. Il tutto deve essere convertito in delle funzioni Swift.

Le funzioni di calcolo dell'SpO2 sono:

1. La funzione **calcolaSpO2()** (Cod. 7.10) è, tra le tre, l'unica funzione pubblica, ovvero quella richiamata dal frontend per ottenere il valore da visualizzare sullo schermo. Questa funzione funge da wrapper. Al suo interno vengono chiamate altre funzioni in una catena annidata. Essa inizia la sua esecuzione chiamando per entrambi i segnali PPG la funzione *calcolaR*, che ritorna il vettore dei rapporti tra i campioni della componente AC e quelli della componente DC. I due vettori di Ratio sono poi messi in relazione per calcolare il vettore di rapporto dei rapporti *Rtot*. Ad ogni cella di questo vettore è applicata la formula 1.3, e l'SpO2 è ritornata come valor medio del vettore finale.

Listing 7.10: Funzione calcolaSpO2()

```
1   func calcolaSpO2(filteredPPG_IR: [Double], filteredPPG_RED: [
    Double])->Int{
2   let R_RED : [Double] = calcolaR(PPGIR_or_RED: filteredPPG_RED
    )
3   let R_IR : [Double] = calcolaR(PPGIR_or_RED: filteredPPG_IR)
4   var vetSpO2 : [Double] = []
5
6   var i = 0
7   while (i<R_RED.count) && (i<R_IR.count) {
8       //calcolo vettore Rtot --> Rtot.i = R_RED.i / R_IR.i
9       let Rtot_i = R_RED[i] / R_IR[i]
10
11      //calcolo vettore VetSpo2 --> VetSpo2.i = 104 - 17 * Rtot
    .i
12      vetSpO2.append( 104 - (17 * Rtot_i) )
13      i = i + 1
14  }
15
```

```
16      //calcolo SpO2 come valor medio di VetSpO2
17      let SpO2 = vetSpO2.avg()
18
19      //ritorno Spo2 convertito in intero
20      return Int(SpO2)
21  }
22
```

2. La funzione **calcolaR()** (Cod. 7.11) permette di ricavare il rapporto tra la componente AC e la componente DC del segnale PPG passato come parametro di input. Per ricavare le componenti, è necessario determinare i massini e i minimi del segnale. Ciò è implementato scandendo il vettore dei campioni e chiamando per ognuno di essi la funzione *controlMaxorMin*. Quest'ultima controlla se il campione corrente è un massimo, oppure un minimo, in modo alternato rispetto all'iterazione (perché si sa che un segnale PPG è caratterizzato da massimi e minimi alternati). Una volta calcolati i vettori min e max, si calcola il vettore di rapporto tra AC/DC, equivalente a (max-min)/min. Tale vettore è reso al chiamante come risultato.

Listing 7.11: Funzione calcolaR()

```
1      private func calcolaR(PPGIR_or_RED: [Double])->[Double]{
2      var check = true
3      var max : [Double] = []
4      var min : [Double] = []
5      var R : [Double] = []
6
7      //calcolo vettore dei massimi e vettore dei minimi
       corrisponendi
8      for i in 0 ..< PPGIR_or_RED.count {
9          if check {
10             if controlMaxorMin(ppgR: PPGIR_or_RED, i: i, maxMin:
       true){
11                 max.append(PPGIR_or_RED[i])
12                 check=false
13             }
14         }
15         else {
16             if controlMaxorMin(ppgR: PPGIR_or_RED, i: i, maxMin:
       false){
17                 min.append(PPGIR_or_RED[i])
18                 check=true
19             }
20         }
21     }
22     //calcolo vettore R = AC/DC = (Max - Min)/Min
23     var i = 0
24     while (i<min.count) && (i<max.count) {
```

```
25            let temp = (max[i] - min[i]) / min[i]
26            R.append(temp)
27            i = i + 1
28        }
29        return R
30    }
31
```

3. La funzione **controlMaxorMin()** (Cod. 7.12) controlla se il campione posizionato alla cella i del segnale passato come parametro è un massimo (se il parametro booleano è a true) o un minimo (altrimenti) di una finestra di 101 elementi. Il controllo è effettuato semplicemente scandendo il vettore e controllando che ogni cella successiva a quella analizzata sia sempre minore di quella a posizione i nel primo caso, minore nel secondo. La funzione ritorna true se la condizione di massimo o minimo (esplicitata con il booleano) è verificata nella finestra, altrimenti false.

Listing 7.12: Funzione controlMaxorMin()

```
1     private func controlMaxorMin(ppgR : [Double], i: Int, maxMin:
      Bool) ->Bool {
2     if(maxMin) {
3         var j=1
4         while j < 101 && i + j < ppgR.count {
5             if ppgR[i] < ppgR[i + j] {
6                 return false
7             }
8             j = j+1
9         }
10        return true
11    }
12    else {
13        for j in 1 ..< 101 {
14            if (i + j >= ppgR.count) || (ppgR[i] > ppgR[i + j]) {
15                return false
16            }
17        }
18        return true
19    }
20}
21
```

Nello sviluppo dell'applicazione, si è deciso di segnalare tutte quelle situazioni in cui l'SpO2 ha un valore preoccupante, ovvero quando il sangue ossigenato è inferiore al 90% del sangue totale nel corpo. In tal caso, la segnalazione è demandata al frontend, con un semplice controllo del valore ricevuto dalla funzione *calcolaSpO2* e conseguente aggiornamento della view.

7.4 Derivazione della pressione

Il calcolo della pressione è gestito da una classe Singleton priva di interfaccia grafica chiamata **NeuralNetworkManager**. Questa classe si occupa di importare la rete neurale utilizzando la libreria **TensorFlowLite**, come esposto nel capitolo 5.8. Il modello è gestito da un **interprete**. Una volta caricato il file del modello, l'interprete preprocessa i dati e li manda in input per ottenere un output corrispondente. La classe AquireECG, terminata l'acquisizione, deamplificati i segnali, ed infine filtrati, istanzia la classe NeuralNetworkManager e chiama il suo metodo pubblico di calcolo delle pressioni, fornendogli come input i vettori contenenti i campioni dell'ECG e del PPG acquisito dal led rosso. Nell'istanziazione della classe è nascosta l'importazione del modello. Nella funzione di calcolo sono celati il preprocessing, l'invocazione della rete per ottenere l'output, la calibrazione dell'output e il calcolo del suo valor medio. A questo punto l'AquireECG richiede il risultato con degli appositi getter.

7.4.1 Data Preprocessing

Quando la rete è interpellata per il calcolo delle pressioni, i segnali ECG e PPG sono già passati per diversi step di filtraggio, ma non sono ancora pronti per essere mandati in input al modello: è necessaria una fase di preprocessamento dei segnali.

Listing 7.13: Preprocessing dei dati di input del modello

```
private func dataPreprocessing(ECG : [Double], PPG_RED: [Double]){
        //resampling da 500Hz a 125Hz
        let ECG_125Hz = reSampling(segnale: ECG)
        let PPG_RED_125Hz = reSampling(segnale: PPG_RED)

        //divido i segnali in batches
        var j = 0
        for i in 0 ..< ECG_125Hz.count {
            ECGBatches[j]?.append(ECG_125Hz[i])
            PPG_REDBatches[j]?.append(PPG_RED_125Hz[i])
            if (i + 1) % NeuralNetworkConstants.nBatches == 0 {
                j = j + 1
            }
        }
        //sottraggo la media del batch ad ogni valore per ogni batch
    e normalizzo per ogni batch
        for i in 0 ..< NeuralNetworkConstants.nBatches {
            //ecg
            ECGBatches[i] = sottraiMediaBatch(segnale: ECGBatches[i
    ]!)
            ECGBatches[i] = normalizza(segnale: ECGBatches[i]!)
            //ppg
```

```
21              PPG_REDBatches[i] = sottraiMediaBatch(segnale:
    PPG_REDBatches[i]!)
22              PPG_REDBatches[i] = normalizza(segnale: PPG_REDBatches[i
    ]!)
23          }
24      }
```

Gli step che compongono la pre-elaborazione dei dati mostrata in Cod. 7.13 sono:

1. **Resampling** (Cod. 7.14) - Il modello è stato allenato con campioni a 125 Hz, perciò la prima fase di preprocessamento consiste nel ricampionare i segnali. Il PulsEcg è stato progettato per campionare a frequenza 500Hz, quindi esattamente il quadruplo. Il resampling in questo caso si traduce semplicemente nel prendere un campione ogni 4. Per rendere la funzione adattabile a possibili modifiche future, il calcolo dell'intervallo di campionamento è calcolato utilizzando delle costanti. Chiaramente questo codice funzionerà solo nel caso in cui la frequenza di campionamento sia più alta, nonché multiplo, di 125Hz. Nel caso fosse più piccola, servirebbe una funzione di interpolazione.

Listing 7.14: Funzione di resampling

```
1       private func reSampling(segnale: [Double])->[Double]{
2           var segnaleRicampionato : [Double] = []
3           var i = 0
4           let fattoreDaAggiungere = Constants.NO_OF_SAMPLES /
    NeuralNetworkConstants.No_of_resampled_samples
5           while i < segnale.count {
6               segnaleRicampionato.append(segnale[i])
7               i = i + fattoreDaAggiungere
8           }
9           return segnaleRicampionato
10      }
11
```

2. **Divisione in batch** - il modello è stato allenato con tensori di input di dimensione 1250, di cui i primi 625 sono campioni PPG del led rosso, gli altri 625 campioni ECG. Ipotizzando una frequenza di campionamento di 500Hz e 10 secondi di acquisizione però, i campioni totali sono 5000 ECG e 5000 PPG RED. Una volta ricampionati, diventano 1250 ECG e 1250 PPG RED. Questo significa che ogni vettore di samples dovrà essere scomposto in due batches da 625.

3. **traslazione sullo zero** (Cod. 7.15) - prima di unificare i due vettori in un unico tensore da 1250 elementi, le porzioni di segnale contenute in ogni batch vengono centrate sullo zero, sottraendo ad ogni campione il valor medio del segnale, per il batch d'appartenenza.

Listing 7.15: Funzione di centramento

```
    private func sottraiMediaBatch(segnale: [Double])->[Double] {
        let valorMedio = segnale.avg()
        var risultato : [Double] = []
        for i in 0 ..< segnale.count {
            risultato.append(segnale[i] - valorMedio)
        }
        return risultato
    }

```

4. **Standardizzazione** (Cod. 7.16) - l'ultimo step del preprocessamento dei dati, precedente all'unione del batch ECG e del batch PPG per comporre il tensore di input, è la standardizzazione. L'obiettivo è quello di rendere i campioni parte di una distribuzione standard, ovvero avente media zero e varianza pari a 1. La standardizzazione consiste nel sottrarre ad ogni campione il valor medio e dividere il risultato per la deviazione standard.

Listing 7.16: Funzione di standardizzazione

```
    private func normalizza(segnale: [Double])->[Double] {
        let media = segnale.avg()
        let devStd = segnale.std()

        var risultato : [Double] = []
        for i in 0 ..< segnale.count {
            let ris = (segnale[i] - media)/devStd
            risultato.append(ris)
        }
        return risultato
    }

```

A questo punto, batch per batch i segnali sono unificati in un unico tensore e dati in input alla rete.

7.4.2 Elaborazione dell'output e ipertensione

Il modello estrae per ogni batch un valore di pressione sistolica ed uno di pressione diastolica. Il risultato dell'elaborazione saranno quindi due vettori composti da tanti elementi quanti sono i batch. In fase di implementazione si è notato che i valori restituiti in output dalla rete necessitano di una **calibrazione**. Questo è dovuto al fatto che il dataset MIMIC con cui è stata allenata la rete possiede solo misurazioni di pazienti in terapia intensiva, ovvero persone con valori di pressione tendenzialmente irregolari, frutto di diverse patologie. Nel blocco di codice 7.17

è riportato l'algoritmo di calibrazione per entrambe le pressioni: la pressione diastolica deve essere alzata, in rapporto inversamente proporzionale al suo valore, così da colmare gli effetti che i pazienti fortemente ipotesi del dataser hanno causato sulla rete; analogamente la pressione sistolica deve essere ridotta.

Listing 7.17: Funzioni di calibrazione delle pressioni

```
func calibrazioneDia(pressioni: [Float32])->[Float32] {
        var dia = pressioni
        for i in 0 ..< dia.count {
            //calibrazione dias
            if dia[i] <= 55 {
                dia[i] = dia[i] + 10
            }
            else if dia[i] > 55 && dia[i] <= 60 {
                dia[i] = dia[i] + 5
            }
            else if dia[i] > 80 {
                dia[i] = dia[i] - 15
            }
        }
        return dia
    }

    func calibrazioneSis(pressioni: [Float32]) ->[Float32]{
        var sis = pressioni
        var mean_diff_sis : Float32 = 55
        for i in 0 ..< sis.count {
            //calibrazione dias
            if sis[i] <= 155 {
                sis[i] = sis[i] - mean_diff_sis + 20
            }
            else if sis[i] > 155 && sis[i] < 165 {
                sis[i] = sis[i] - mean_diff_sis + 10
            }
            else if sis[i] >= 165 && sis[i] < 185 {
                sis[i] = sis[i] - mean_diff_sis
            }
            else if sis[i] >= 185 {
                sis[i] = sis[i] - mean_diff_sis - 11
            }
        }
        return sis
    }
```

Una volta che i valori delle pressioni sono stati calibrati, il risultato comprende un vettore di pressioni sistoliche ed uno di pressioni diastoliche. I valori definitivi di SPB e DPB sono calcolati come valor medio di questi valori se la dimensione dei vettori è pari a due. Nel caso in cui i valori fossero più di 2, si è scelto di restituire

come risultato il valor medio dei soli valori centrali, in modo da considerare solo le pressioni ottenute dal tratto "più pulito" dei segnali. Infatti spesso i primi e gli ultimi secondi di un'acquisizione producono campioni meno affidabili, dovuti al movimento dell'utente per posizionarsi/spostarsi dall'elettrodo/sensore o da un ritardo nel poggiare il dito. La funzione complessiva di calcolo della pressione (Cod. 7.18), la quale è chiamata dall'AquireECG VC sull'istanza di NeuralNetworkManager, si occupa quindi del preprocessing, dell'inferenza del modello, della calibrazione e del calcolo del valor medio dell'output. I valori di pressione sistolica e diastolica sono memorizzati in due proprietà private della classe che possono essere ottenute dai VC mediante getter pubblici.

Listing 7.18: Funzione calcolo della pressione completa

```
func computePressioni(ECG : [Double], PPG_RED: [Double]){
        do {
            //preprocessing
            dataPreprocessing(ECG: ECG, PPG_RED: PPG_RED)

            //itero per ogni batch
            for i in 0 ..< NeuralNetworkConstants.nBatches {
                //costruzione del tensore di input
                let PPG_ECG_1Batch = PPG_REDBatches[i]! + ECGBatches[
    i]!
                let inputData: Data = PPG_ECG_1Batch.
    withUnsafeBufferPointer {Data(buffer: $0)}
                try self.interpreter!.copy(inputData, toInputAt: 0)
                // Invocazione del modello
                try self.interpreter!.invoke()
                // Reperimento del tensore risultato
                let outputTensor = try self.interpreter!.output(at:
    0)
                let outputSize = outputTensor.shape.dimensions.reduce
    (1, {x, y in x * y})
                let outputData = UnsafeMutableBufferPointer<Float32>.
    allocate(capacity: outputSize)
                outputTensor.data.copyBytes(to: outputData)
                pressioniSis.append(outputData[0])
                pressioniDia.append(outputData[1])
            }
            //calibrazione
            pressioniDia = calibrazioneDia(pressioni: pressioniDia)
            pressioniSis = calibrazioneSis(pressioni: pressioniSis)

            //se ho solo due valori di pressione ne faccio la media
            if pressioniDia.count == 2 {
                pressSis = Int(pressioniSis.avg())
                pressDia = Int(pressioniDia.avg())
            }
```

```
31              // se ne ho più di due faccio la media dei valori
    centrali
32              else if pressioniDia.count > 2 {
33                  var IndiceStart = pressioniDia.count/2 - 1
34                  var accSis : Float32 = 0
35                  var accDia : Float32 = 0
36                  for i in IndiceStart ..< (IndiceStart+2){
37                      accSis = accSis + pressioniSis[i]
38                      accDia = accDia + pressioniDia[i]
39                  }
40                  pressSis = Int(accSis/2)
41                  pressDia = Int(accDia/2)
42              }
43
44          } catch {
45            print("Errore")
46          }
47      }
```

La segnalazione di un'eventuale **ipertensione** è demandata al frontend, in diverse parti dell'applicazione (ShowECG VC, GeneralShowECG VC e generazione del pdf). Il controllo viene effettuato con dei semplici blocchi if-else e la segnalazione avviene modificanto il contenuto di alcune UILabels o mostrando degli output. Per far si che l'applicazione non sostituisca il medico, ma segnali solo le situazioni realmente preoccupanti, per invitare il paziente a intraprendere una consultazione con lui, si è ritenuto di dover segnalare solo i casi di ipertensione gravi, caratterizzati da una SPB maggiore o uguale a 180 mmHg e da una DPB maggiore o uguale a 110 mmHg.

Capitolo 8

Conclusione

L'obiettivo di questa tesi è stato lo sviluppo di una applicazione iOS che potesse completare il sistema PulsEcg, affiancandosi al bracciale con cui si interfaccia. Tale sistema permette la misurazione di diversi parametri vitali, in particolare dei valori di pressione, attraverso tecniche non invasive, rapide, adatte a tutte le età ed economicamente accessibili. Nei precedenti capitoli, è stata offerta una panoramica piuttosto dettagliata di come le strategie d'implementazione delle specifiche di sistema siano state selezionate abbracciando una vasta gamma di discipline, dalla teoria d'elaborazione dei segnali al machine learning, dalle nozioni di elettronica al design software. Il risultato di questa commistione è un prodotto piuttosto soddisfacente, ma non privo di **possibilità di miglioramento**.

Degli esempi potrebbero essere:

1. integrazione nell'hardware di un **sensore di temperatura**. Perché il dato possa essere inviato all'applicazione, il servizio ECG per la comunicazione BLE deve essere dotato di una nuova caratteristica, che dovrà essere gestita dall'app. La temperatura corporea dovrà poi essere aggiunta ai dati della misurazione, in modo da poter essere salvata e visualizzata opportunamente. Anche in questo caso sarebbe consigliata la segnalazione di temperature problematiche (ad esempio al di sopra del 37.5).

2. integrazione nell'hardware di un **sensore di movimento**. Questo nuovo componente apre nuovi orizzonti funzionali all'applicazione: la rilevazione di movimenti allarmanti (ad esempio una forte accelerazione seguita da un periodo troppo lungo di immobilità, probabile segno di una caduta problematica) che scateni una qualche routine d'emergenza. Si potrebbe pensare di chiamare automaticamente un numero salvato nelle preferenze, o di mandare un sms allegando la posizione corrente del paziente. Anche in questo caso è necessaria l'aggiunta di una nuova caratteristica BLE. Inoltre l'applicazione deve essere

aggiornata in modo da poter lavorare in background, anche quando l'utente non la sta usando.

3. aggiunta di una **schermata di statistiche** nell'applicazione. Le statistiche potrebbero mostrare l'andamento dei parametri vitali nel tempo, per evidenziare peggioramenti o miglioramenti nella salute del paziente.

4. migliorare l'efficienza dell'applicazione, cercando di **ridurre il costo computazionale della funzione di media mobile** per il filtraggio dei segnali. Il problema potrebbe essere risolto utilizzando la ricorsione.

Allo stato attuale, considerando anche la pandemia da SARS-CoV-2, un sistema del genere potrebbe essere di grande aiuto alle persone con problemi cardiaci tali da richiedere un monitoraggio continuo dei parametri vitali misurati dal PulsEcg. L'applicazione ed il bracciale permettono infatti di ridurre al minimo indispensabile gli incontri tra medico e paziente.

Inoltre, il sistema PulsEcg apre un nuovo scenario per la misurazione della pressione, caratterizzato dall'abbandono dello sfigmomanometro.

Infine, a fronte dei possibili miglioramenti elencati in precedenza, la presente tesi potrebbe garantire un punto di partenza già piuttosto soddisfacente per rendere il sistema ancora più performante, affidabile e utile.

Bibliografia

[1] H. A. Snellen. *Willem Einthoven (1860–1927) Father of Electrocardiography*. Springer Netherlands, 2012 (cit. a p. 2).

[2] R.D. Peña H. Azucena E. Ríos e J. Díaz. «Design and implementation of a simple portable biomedical electronic device to diagnose cardiac arrhythmias». In: *Elsevier* (2015) (cit. a p. 3).

[3] Anthony DupreSarah VincentPaul A. Iaizzo. *Handbook of Cardiac Anatomy, Physiology, and Devices*. Humana Press, 2005 (cit. a p. 3).

[4] Stefano Bovani Giuseppe Felitti. *L'ECG Moderno - My Easy Test*. Lulu, 2019 (cit. alle pp. 4, 5).

[5] A. R. Hidalgo-Munoz L. N. Ribeiro e G. Favier. «A tensor decomposition approach to noninvasive atrial activity extraction in atrial fibrillation ECG». In: 2015 (cit. a p. 5).

[6] J. Ferretti V. Randazzo e E. Pasero. «A low-cost wearable device for anytime ECG monitoring». In: *MDPI* (2020) (cit. alle pp. 5, 123).

[7] M. Sekine T. Tamura Y. Maeda e M. Yoshida. «Wearable Photoplethysmographic Sensors-Past and Present». In: *MDPI* (2014) (cit. a p. 6).

[8] A. Nguyen N. Bui. «PhO2: Smartphone based Blood Oxygen Level Measurement Systems using Near-IR and RED Wave-guided Light». In: nov. 2017 (cit. a p. 6).

[9] Soumil Chugh. «Effect of Different Signal Processing Techniques on a Calibration Free Pulse Oximeter». In: 2017 (cit. a p. 7).

[10] I. Nitzan O. Yossef M. Cohen e M. Nitzan. «Pulse Oximetry with Two Infrared Wavelengths without Calibration in Extracted Arterial Blood». In: *MDPI* (2019) (cit. a p. 7).

[11] J. G. Webster. *Design of Pulse Oximeters*. CRC Press, 1997 (cit. a p. 7).

[12] V. Beltrán-Campos Y. Triunfo-Méndez N. Padilla-Raygoza e C. Sandoval-Salazar. «Comparison of Blood Pressure Measurement with Two Digital Sphygmomanometers». In: *Journal of Advances in Medicine and Medical Research* (2019) (cit. a p. 7).

[13] Z. Mao R. Wang W. Jia e M. Sun. «Cuff-Free Blood Pressure Estimation Using Pulse Transit Time and Heart Rate». In: 2014 (cit. a p. 8).

[14] A. Pantelopoulos e N. G. Bourbakis. «A Survey on Wearable Sensor-Based Systems for Health Monitoring and Prognosis». In: *IEEE* (2010) (cit. a p. 9).

[15] F. Cafarelli. «Wearable Bluetooth ECG Sensor for Biometric Identification». In: (2014) (cit. alle pp. 12, 14, 29).

[16] D. Motta. «Design and Development of a Multi-Parameter Wearable Medical Device». In: (2019) (cit. alle pp. 14, 15).

[17] Alexander Jung. *Machine Learning. The Basics.* 2021 (cit. a p. 19).

[18] Ayushi Chahal e Preeti Gulia. «Machine Learning and Deep Learning». In: *IJITEE* (2019) (cit. a p. 19).

[19] Humayun Kabir. «Convolutional Neural Network». In: (2021) (cit. a p. 22).

[20] S. Villata. «Cuffless Blood Pressure Measurement». In: (2020) (cit. alle pp. 23, 24).

[21] A. Panerati. «Reti Neurali Ricorrenti per la misurazione non invasiva della pressione arteriosa». In: (2019) (cit. a p. 24).

[22] C. Garcia-Bustelo C. Garcia J. Espada e J.Lovelle. «Swift vs. Objective-C: A New Programming Language». In: (2015) (cit. a p. 45).

[23] Apple. *Apple Developer Documentation*. `https://developer.apple.com/documentation/` (cit. alle pp. 48, 50, 60, 62, 64, 82, 83, 86, 98).

[24] S. Jagannath M. Desmarais C. Hayne e R. Keller. «A Survey on User Expectations for Interface Builders». In: (1996) (cit. a p. 49).

[25] Stephen H. Kaisler. *Software Paradigms*. Wiley, 2005 (cit. a p. 50).

[26] Mats Andersson. «Use case possibilities with Bluetooth Low Energy in IoT applications». In: *U-blox* (2014) (cit. a p. 51).

[27] Naresh Gupta. *Inside Bluetooth Low Energy*. Artech House, 2013 (cit. a p. 51).

[28] Slawomir Jasek. «Gattacking Bluetooth smart devices». In: *SecuRing* (2017) (cit. a p. 52).

[29] Giacomo Andreucci. *Pro iOS Geo*. Apress, 2013 (cit. a p. 55).

[30] Wikipedia. *Wikipedia - l'Enciclopedia Libera*. `https://it.wikipedia.org/` (cit. a p. 68).

[31] R. WangXing e Z. Yang. «Research on User Experience Design Consistency of Internet Products Based on User Experience». In: (2020) (cit. a p. 69).

[32] C. Lallemand e V. Koenig. «Measuring the Contextual Dimension of User Experience: Development of the User Experience Context Scale (UXCS)». In: (2020) (cit. a p. 69).

[33] A. L. Osvalder Y. Liu e M. Karlsson. «Considering the Importance of User Profiles in Interface Design». In: (2010) (cit. alle pp. 69, 70).

[34] Qutaiba Abdulla. «Color Emotion Guide». In: 2014 (cit. alle pp. 70, 71).

[35] Adobe. *Adobe Color*. `https://color.adobe.com/` (cit. a p. 71).

[36] mcmurryjulie. *Icona di Prognosi*. `https://pixabay.com/it/illustrations/la-prognosi-icona-scheda-del-paziente-2803190/` (cit. alle pp. 73, 74).

[37] A. L. Osvalder Y. Liu e M. Karlsson. «Analysis of Signal Noise Reduction by Using Filters». In: (2018) (cit. a p. 113).

[38] N. A. Jasem A. N. Abdalgaffar e A. I. Abbo. «Notch filters design with enhanced performance». In: (2019) (cit. a p. 115).

[39] J. O. Smith J. Liski B. Bank e V. Valimaki. «Converting Series Biquad Filters Into DelayedParallel Form: Application to Graphic Equalizers». In: (2019) (cit. a p. 115).

[40] Vishakha Pandey. «High frequency noise removal from ECG using moving average filters». In: (2016) (cit. a p. 117).

[41] J. Pan–Tompkins. «A Real-Time QRS Detection Algorithm». In: (1985) (cit. a p. 121).