



Laboratorio Applicazioni Mobili

Informatica, Università di Bologna

Report progetto progetto iOS

Filippo Bartolucci

2019/2020

Mail:

filippo.bartolucci2@studio.unibo.it

Matricola:

0000838531

Indice

Applicazione: Health Monitor	3
Sviluppo	4
Design	5
Report Data	6
Content View	6
HealthTab	7
Add Report	8
ReportList	9
ReportDetail	10
Settings	11
Dashboard	12
Tema scuro	13

Applicazione: Health Monitor

Health Monitor è un'applicazione iOS per il monitoraggio della salute personale attraverso l'inserimento di dati sanitari. I report sono visualizzabili in app attraverso una lista alla quale si possono applicare filtri.

Funzionalità:

- Creazione e modifica report:

Un report è una collezione di dati sanitari che l'utente salva nell'applicazione. Ogni report contiene dati riguardanti temperatura, peso, battito cardiaco e glicemia. Per ognuno di questi valori è associato un intero da 1 a 5 che ne indica l'importanza. Ogni report viene identificato dalla data associata, in caso di aggiunta di report con stessa data viene fatta una media dei valori salvati. Per ogni report è possibile aggiungere una nota testuale opzionale. I report possono essere consultati attraverso una lista che permette di filtrare in base all'importanza di un certo tipo di valore.

- Grafici:

L'applicazione ha una sezione che mostra la media dei valori e dei grafici che mostrano la variazione dei valori nel tempo

- Notifiche:

Imposta un promemoria giornaliero che ti ricorda di creare un report. Possibilità di monitorare l'andamento dei valori che si desidera e ricevere una notifica se si supera un dato valore di soglia

Sviluppo

L'applicazione è stata sviluppata con Xcode 11.6 e Swift 5.2 su macOS Catalina 10.15.6. L'applicazione è stata testa su un iPhone 11 Pro e sui vari modelli di iPhone del simulatore iOS, tutti aggiornati alla versione di iOS 13.6. Per lo sviluppo della UI ho usato il framework SwiftUI e per la gestione dei dati persistenti Core Data.

SwiftUI segue un'approccio dichiarativo per lo sviluppo delle interfacce, non ci sono controller da creare. Tutto si basa sulla composizione dei vari tipi di View per creare UI e la parte logica del programma è rigorosamente separata dalla costruzione delle View.

Un suo grande vantaggio è la semplicità con cui si può costruire un'interfaccia. Si possono utilizzare le funzioni di *Drag 'n Drop* di Xcode per costruire View lasciando che il codice venga scritto automaticamente. In alternativa si può scrivere manualmente il codice sorgente e vedere in tempo reale le modifiche in un canvas a lato, senza bisogno di ricompilare nulla. È anche possibile interagire con la View nel canvas senza mai uscire da Xcode.

```
1 //  
2 //  boxView.swift  
3 //  HealthMonitor_LAM  
4 //  
5 //  Created by Filippo Bartolucci on 12/08/2020.  
6 //  Copyright © 2020 Filippo Bartolucci. All rights reserved.  
7 //  
8  
9 import SwiftUI  
10  
11 struct boxView: View {  
12     @Environment(\.colorScheme) var colorScheme  
13     var content : AnyView  
14  
15     var body : some View {  
16         Group{  
17             self.content  
18         }  
19         .frame(maxWidth : widthBound)  
20         .background(Color("boxBackground"))  
21         .clipShape(RoundedRectangle(cornerRadius: 14, style: .continuous))  
22     }  
23 }  
24  
25 struct boxView_Previews: PreviewProvider {  
26     static var previews: some View {  
27         boxView(content: AnyView(Text("\nHello World\n")))  
28     }  
29 }
```

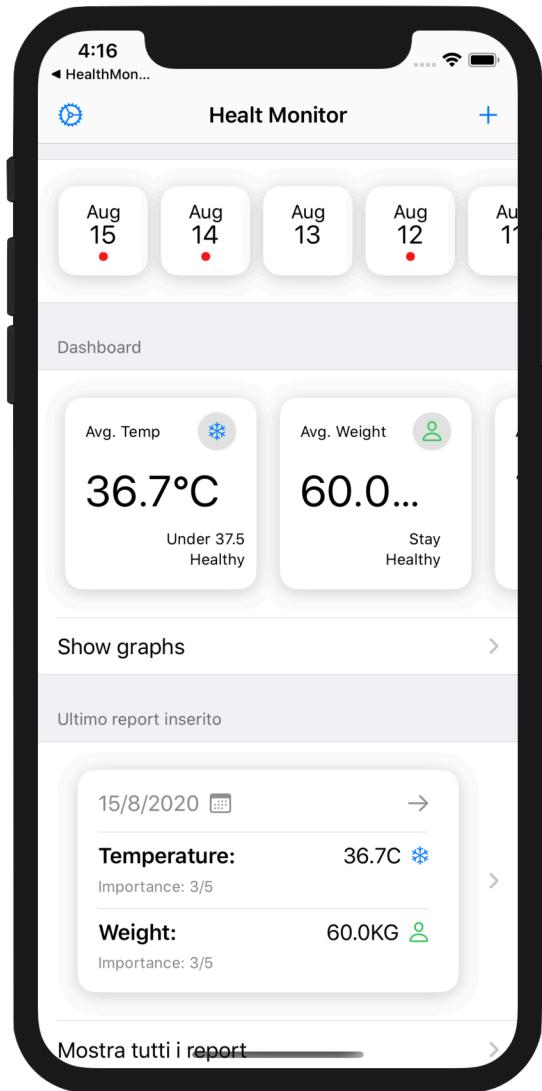


Editor di Xcode con Canvas

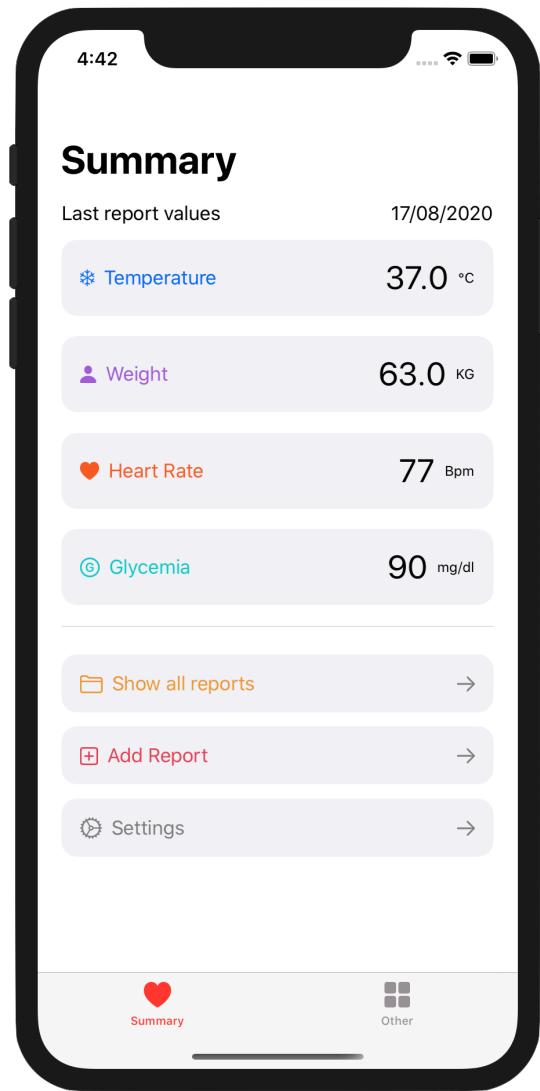
È un framework giovane e quindi più limitato rispetto a UIKit, però continua a crescere e molte nuove funzionalità, tipo LazyStack e GridStack, arriveranno con il prossimo iOS 14. Data la giovinezza non sempre è facile trovare risorse online adeguate per tutti i tipi di problemi.

Per la creazione dei grafici ho utilizzato il package [SwiftUICharts](#) (MIT License)

Design



Design iniziale



Design finale

Non avendo, all'inizio del progetto, un'idea precisa di come l'app dovesse apparire, ho iniziato a sperimentare diversi stili.

Inizialmente ho pensato che avere un'unica pagina con una lista di elementi grafici di vario tipo potesse essere una buona idea, ma durante lo sviluppo l'interfaccia diventava sempre più complessa, meno intuitiva e coerente. Per questo ho deciso di rifare la UI aggiungendo una tab bar in basso, semplificando gli elementi grafici e cercando di adottare stile e dimensioni coerenti per tutte le view.

Ho cercato di ispirarmi all'applicazione Salute di iOS.

Report Data

Attribute	Type
I id	UUID
D date	Date
S note	String
N weight	Float
N temperature	Float
N weightImportance	Integer 16
N temperatureImportance	Integer 16
N heartRateImportance	Integer 16
N heartRate	Integer 16
N glycemiaImportance	Integer 16
N glycemia	Integer 16

Report Entity

I report dell'app seguono il modello rappresentato dalla seguente Entity di CoreData:

- Date e note sono, rispettivamente, il giorno e la nota (opzionale) associati al singolo report.

- Temperature, weight, heartRate e glycemia sono i valori sanitari che vengono salvati in ogni report.

- Per ogni valore sanitario c'è un intero (da 1 a 5) che indica quanto importanza l'utente dà a quel valore.

I report possono essere modificati in ogni momento dall'utente.

ContentView

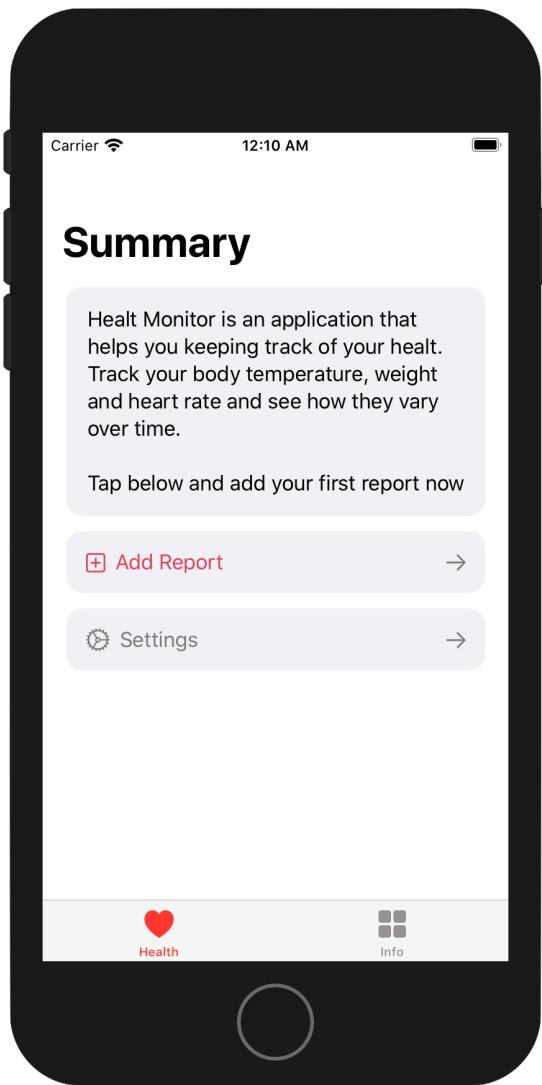
```
// MARK: -CoreData Setup
@Environment(\.managedObjectContext) var managedObjectContext
@FetchRequest(
    entity: Report.entity(),
    sortDescriptors: [NSSortDescriptor(keyPath: \Report.date, ascending: false)])
var reports: FetchedResults<Report>

// MARK: -View
var body: some View {
    TabView(){
        Tab1View(reports: self.reports).tabItem {
            HStack{
                Image(systemName: "heart.fill").font(.system(size: tabIconSize))
                Text("Health")
            }
        }
        Tab2View(reports:self.reports).environment(\.managedObjectContext, managedObjectContext).tabItem {
            HStack{
                Image(systemName: "square.grid.2x2.fill").font(.system(size: tabIconSize))
                Text("Info")
            }
        }
    }
    .accentColor(.red)
}
```

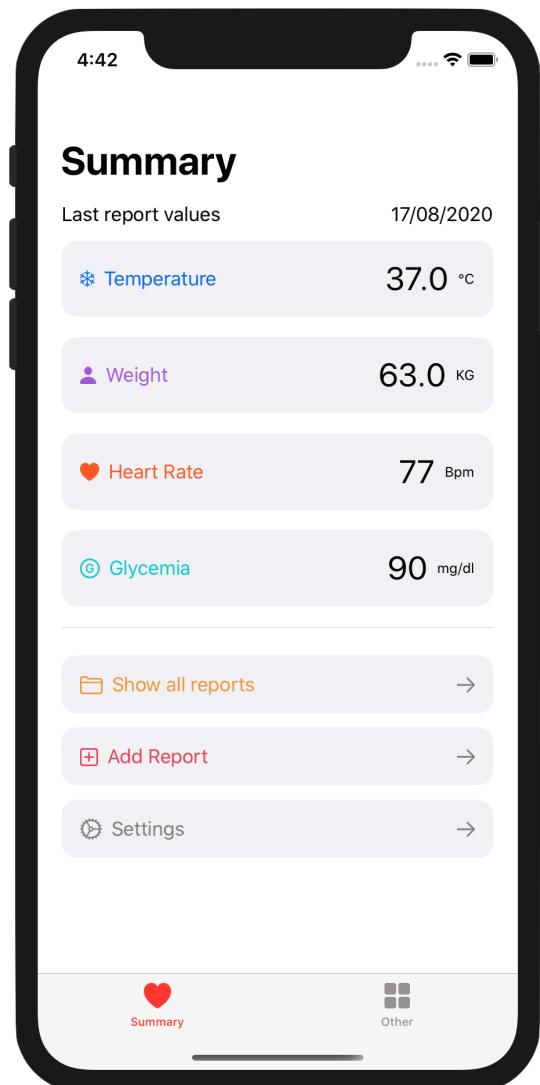
ContentView.swift

La view principale, lanciata da SceneDelegate, è “ContentView”. Si occupa della richiesta dati per CoreData, salvati per data in un array, e della creazione delle due tab, Health e Dashboard.

HealthTab



HealthTab senza report salvati



Riepilogo dei dati ultimo report

È la view che si vede al lancio. Se non ci sono report salvati mostra una messaggio e un pulsante per creare il primo report. Altrimenti c'è un riepilogo dei dati sanitari dell'ultimo report.

"Show Report", "Add Report" e "Settings" sono dei NavigationLink che portano alla view con lista dei report, alla view "addReport" e alle impostazioni. Per distinguere i pulsanti dagli elementi non tappabili ho aggiunto una freccia a destra dei Navigation Link

```
struct boxView: View {
    @Environment(\.colorScheme) var colorScheme
    var content : AnyView

    var body : some View {
        Group{
            self.content
        }
        .frame(maxWidth : widthBound)
        .background(Color("boxBackground"))
        .clipShape(RoundedRectangle(cornerRadius: 14, style: .continuous))
    }
}
```

Il box che circonda tutti gli elementi all'interno della view è a sua volta una view che riuso per avere lo stesso stile.

Add Report

View che permette la creazione di un nuovo report. I vari pulsanti sono NavigationLink che portano ai TimePicker e TextField che permettono di inserire i dati. Per ogni valore TextField c'è un Picker associato usato per inserire l'importanza. Può andare da 1 a 5 e di default è 3.

Il pulsante "Add Report" è disabilitato finché non vengono rispettate le seguenti condizioni:

- La temperatura deve essere compresa tra 33.0 e 44.0 gradi
- Il peso deve essere maggiore di 0
- Il battito deve essere maggiore di 30
- La glicemia deve essere maggiore di 50

I valori vengono salvati in variabili @State della view. Sono tutti di tipo string perché TextField lavora con le stringhe. Per evitare che si possano scrivere caratteri alfanumerici in campi che dovrebbero essere numerici ho messo come tastiera decimalPad e numeralPad.

Quando si preme "Add Report" prima di creare un nuovo report viene effettuato un controllo tra quelli già esistenti. Se c'è già un report con la stessa data viene fatta la media tra i valori del vecchio e nuovo report, altrimenti ne viene creato uno nuovo.

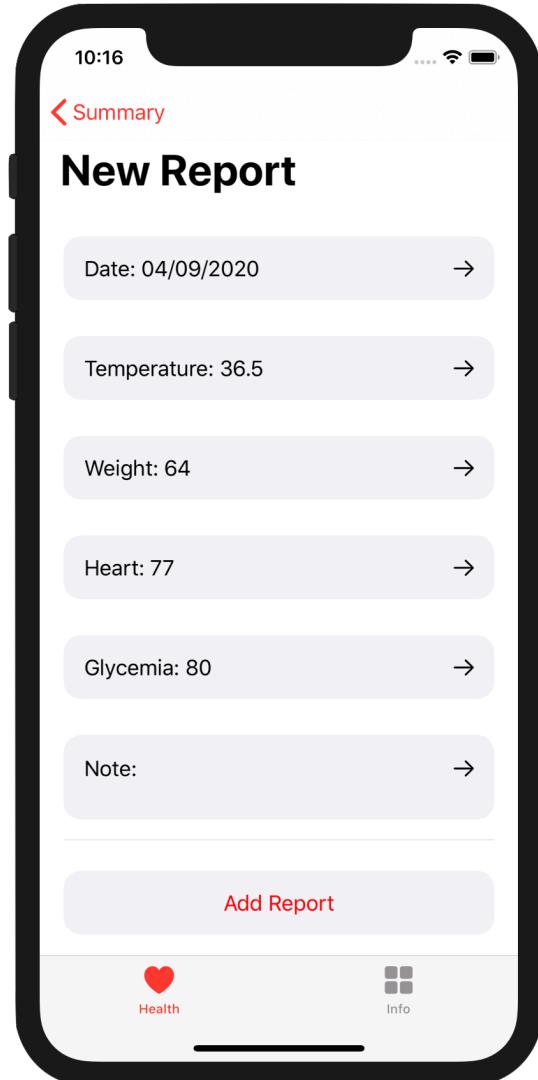
Siccome i valori nella view sono stringhe e nel report sono float o int16, devo effettuare il cast di ogni valore, facendo attenzione con gli optional per evitare nil.

In Swift un numero decimale, scritto come stringa, viene trasformato correttamente in un float solo se il simbolo che viene usato per la parte decimale è un punto. Per alcune lingue (come l'italiano) il simbolo usato per la parte decimale è la virgola e il decimalPad viene adattato di conseguenza. Però il cast da string a float di un numero decimale con la virgola è nil e questo porterebbe a non avere il valore il valore che si sta inserendo.

Per evitare tutto questo, prima del cast, sostituisco nelle stringhe ogni occorrenza di "," con "." riuscendo così a utilizzare entrambi le notazioni per la parte decimale senza problemi

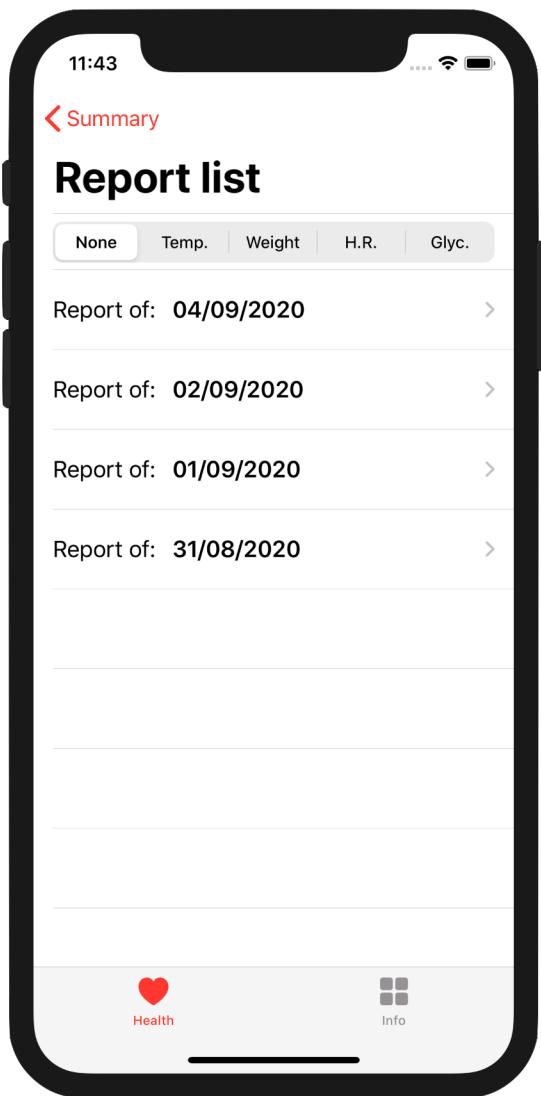
```
Float(self.temperature.replacingOccurrences(of: ",", with: ".")) ?? Float(0)
```

Sostituzione di "," con "."



AddReport View

ReportList



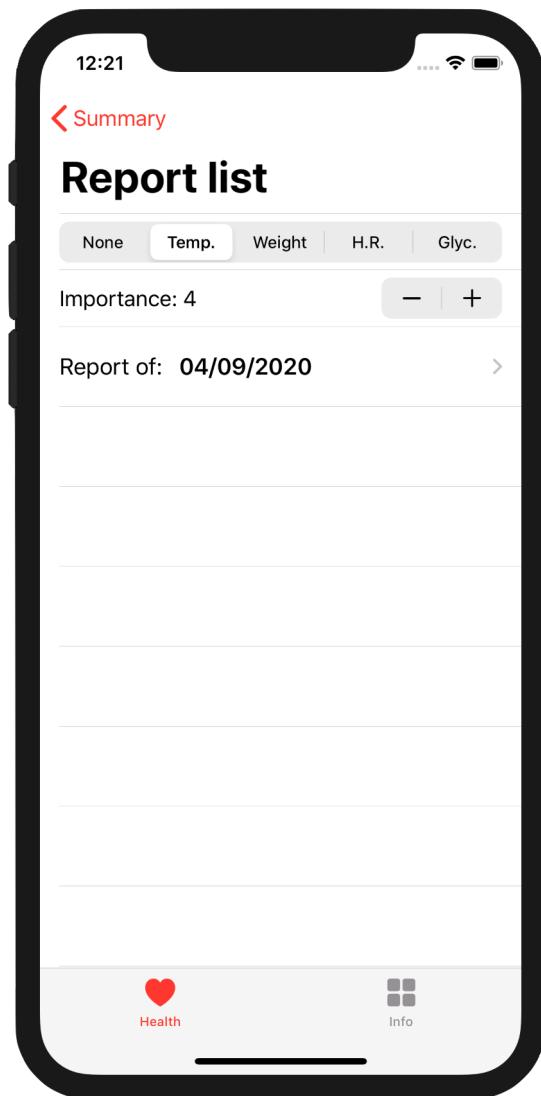
Lista di report senza filtri

In questa view vengono visualizzati tutti i report salvati. Il corpo della view è una list che crea per ogni report salvato una ReportRow.

Il SegmentedPicker permette di filtrare la lista in funzione di un valore e della sua importanza. La lista dei report visibili viene creata dalla funzione filterList.

Se il valore del picker è impostato su "None" nessun valore della lista viene filtrato. Quando si seleziona un altro valore, compare nella view uno stepper per aumentare o diminuire l'importanza. La lista viene filtrata attraverso la funzione filter() degli array.

Si può eliminare un report scorrendo verso sinistra dal bordo destra della ReportRow.

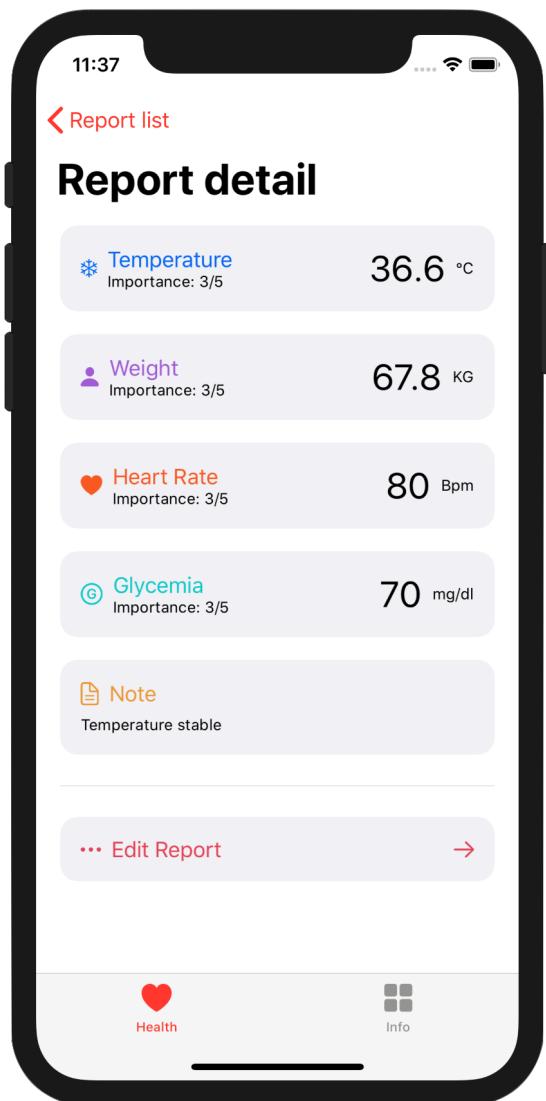


Lista con attivo filtro temperatura

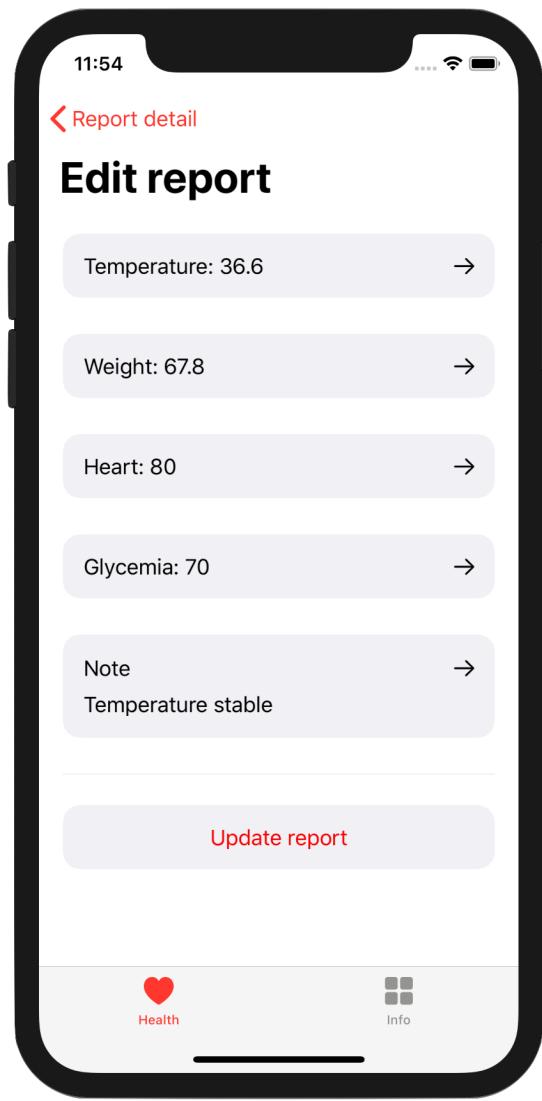
```
var body: some View {
    List{
        Section{
            Picker(selection: self.$pickerValue, label: Text("Filter by:")) {
                ForEach(0..
```

Body della view

ReportDetail



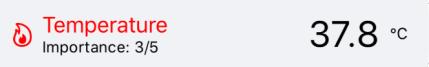
Dettaglio di un report



View modifica report

View che permette di vedere i valori di un report con la loro importanza.

Se la temperatura supera i 37.5°C il box con il valore cambia per segnalare il superamento della temperatura consigliata.



Tocando "Edit Report" si possono modificare i valori del report in una nuova view.

Settings

In questa view si gestiscono le impostazioni per le notifiche.

Al primo avvio dell'applicazione vengono chiesti i permessi per poter inviare notifiche. Se vengono negati viene mostrato un collegamento alle impostazioni per abilitarle.

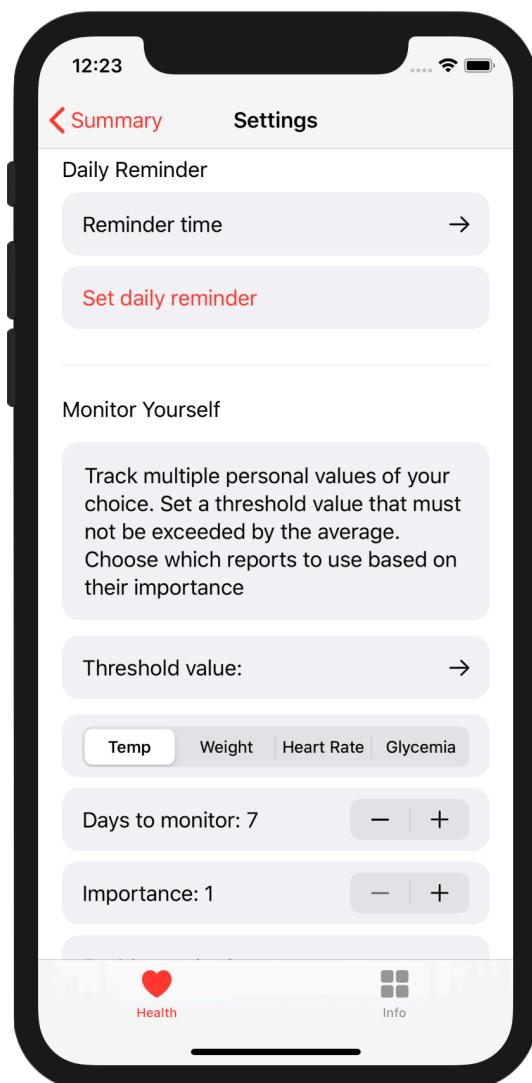
Si può impostare un orario per ricevere un promemoria giornaliero che ci ricorda di inserire un report.

È anche possibile monitorare uno o più valori per vedere se la media supera un certo valore soglia in un dato periodo di tempo.

Per realizzare questa funzionalità, potendo utilizzare soltanto le local notification, ho creato un oggetto core data, "Monitoring".

D	day	Date	◊
N	daysLeft	Integer 16	◊
III	id	UUID	◊
N	importance	Integer 16	◊
N	limit	Float	◊
N	numberOfDays	Integer 16	◊
D	startDay	Date	◊
S	value	String	◊

Oggetto Monitoring



Impostazioni

In questo oggetto ho salvato tutte le informazioni del monitoraggio che mi servono. Il giorno di inizio del monitoraggio, il numero di giorni che dura, il nome del valore che si vuole monitorare, l'importanza e il valore soglia. È possibile creare più monitoraggi contemporaneamente. La view HealthTab si occupa poi della gestione delle notifiche.

Quando viene eseguita la funzione `.onAppear()` di HealthTab vengono controllati tutti gli oggetti "Monitoring".

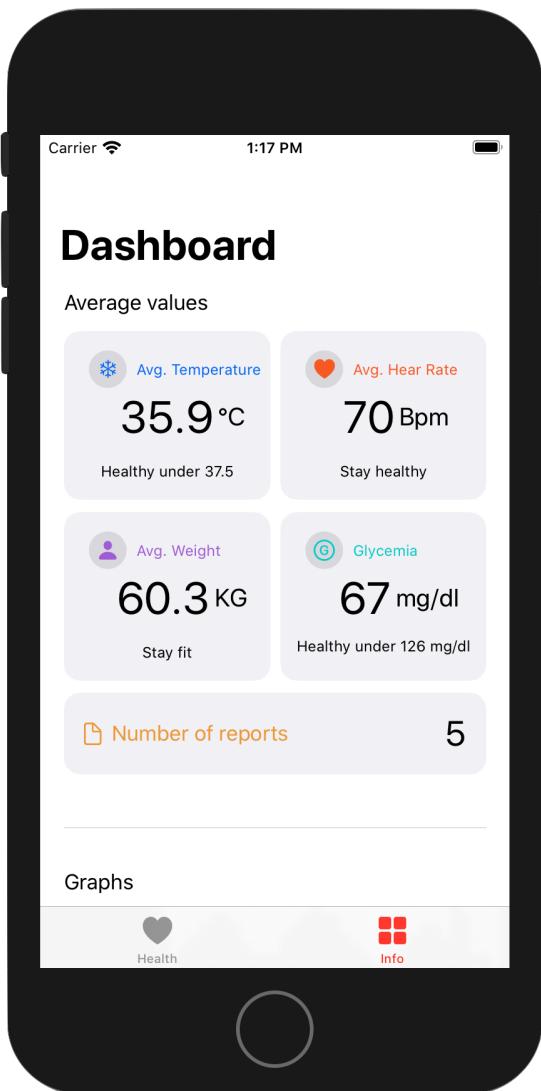
Ogni oggetto ha un valore "day" che indica il giorno dell'ultima modifica. Se questo valore è diverso da quello del giorno corrente Monitoring viene aggiornato. "daysLeft" viene decrementato del numero di giorni che intercorrono tra "day" e il giorno corrente e "day" diventa il giorno corrente. Facendo così mi assicuro che gli oggetti Monitoring non vengano aggiornati ogni volta che viene eseguita `.onAppear()` di HealthTab.

Se il valore di "daysLeft" è uguale a zero, viene chiamata una funzione che crea la notifica.

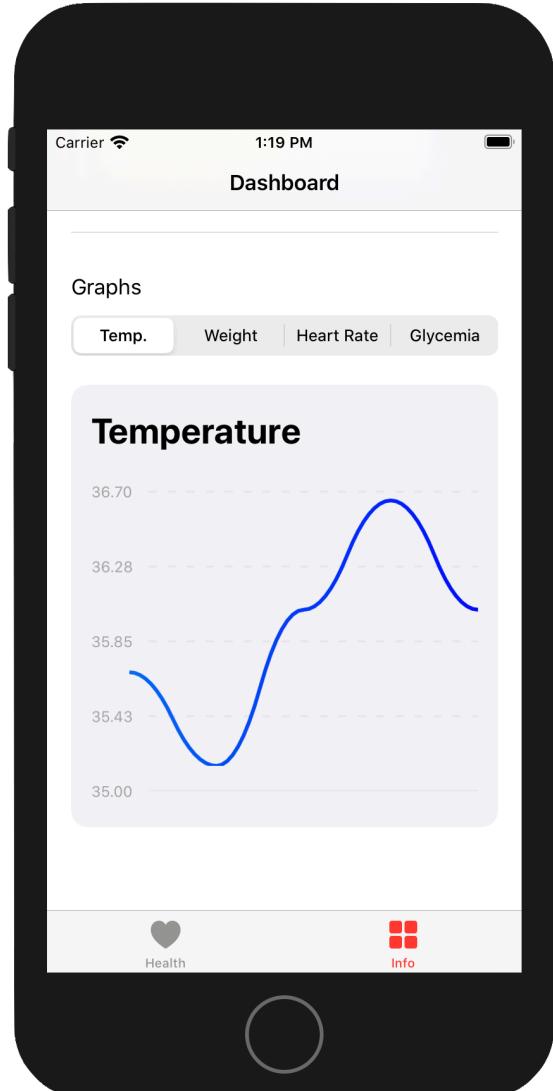
```
for m in monitoring{
    // check to see if the last modified date of the report is different from today
    if !compareDate(date1: m.day ?? Date(), date2: Date()){
        // Days between last update and today
        m.daysLeft -= Int16(daysBetween(firstDate: m.day!, secondDate: Date()))
        // Date updating
        m.day = Date()
        if (m.daysLeft < 1){
            let avgV = avgMonitoringValue(m: m, rs: rs)
            if (avgV == 0){
                return
            }
            // notification to send
            self.sendMonitoringNotification(value: m.value!, limit: m.limit, avg: avgV, start: m.startDay!)
        }
    }
}
```

Funzione che aggiorna gli oggetti Monitoring

Dashboard



Media dei valori



Grafici

Questa view permette di avere un riepilogo di varie informazioni sui report. In alto nella view sono visibili 4 card statiche che mostrano la media dei valori. Subito sotto c'è un piccolo pannello che mostra il numero di report salvati.

Scorrendo verso il basso c'è la sezione dei grafici. Per evitare di dover scorrere per vedere tutti i grafici, ho utilizzato un segmented picker che permette di cambiare il grafico visualizzato con un tocco.

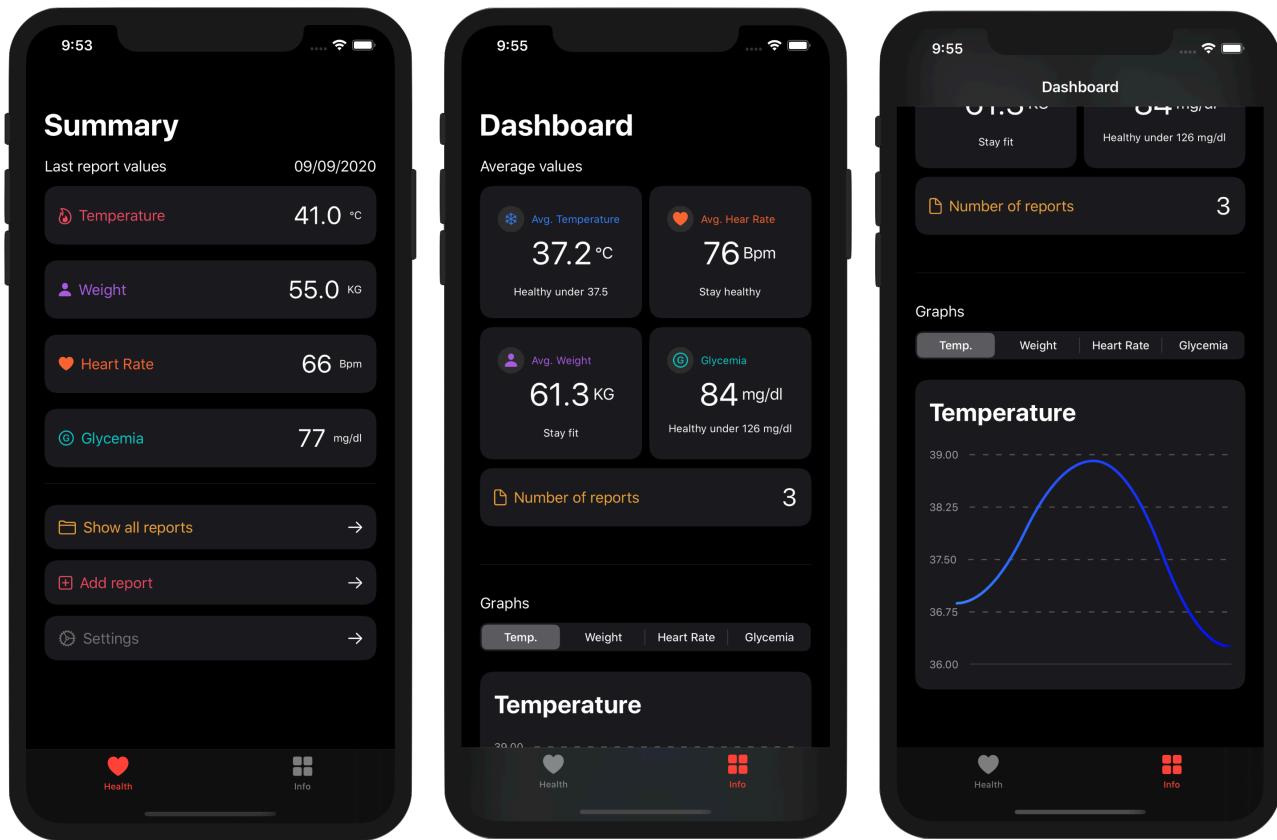
I grafici sono stati realizzati grazie al package [SwiftUICharts](#) (MIT License). Sono molto intuitivi da utilizzare e allo stesso tempo molto personalizzabili. Ho creato dei temi personalizzati per adattare al meglio lo stile del grafo alla mia applicazione.

```
boxView(content: AnyView{
    LineView(data: createWeightArray(reports : self.reports), title: "Weight", style: weightStyle()).background(Color("boxBackground")).padding(.horizontal)
}).frame(minHeight: graphHeight).padding(.vertical)
```

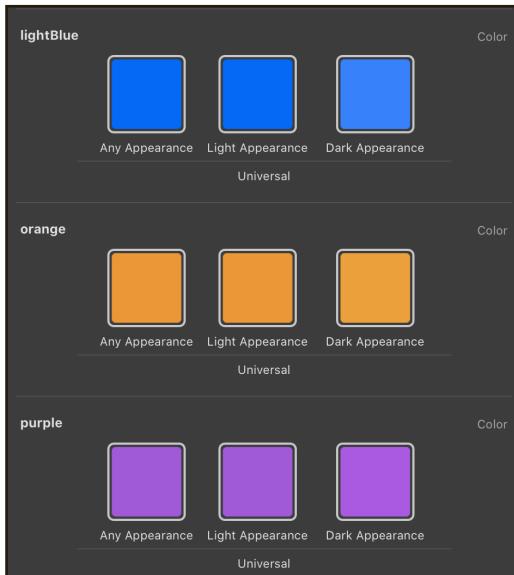
Chiamata di SwiftUICharts LineView

Tema scuro

Un vantaggio dell'utilizzo di SwiftUI è il supporto nativo al tema scuro, introdotto in iOS 13, senza dover scrivere nessuna riga di codice.



Tema scuro applicato a varie view



Colori personalizzati

Per migliora l'interfaccia ho creato dei colori personalizzati in versione per tema chiaro e tema scuro.

Il colore viene automaticamente adattato in base al tema che si sta utilizzando



I due tipi di notifiche