

Laboratorio di Applicazioni Mobili Informatica, Università di Bologna

Matteo Celani

2019-2020

Docente:	Luciano Bononi
Docente:	Federico Montori
Tutor didattico:	Luca Sciullo
Mail:	matteo.celani@studio.unibo.it
Matricola:	0000804303
GitHub:	www.github.com/matteocelani/PersonalHealthMonitor

Indice

1 Personal health monitor	3
1.1 Creare e modificare report	3
1.2 Notifiche	3
1.3 Grafici	3
2 HealtMonitor per iOS	4
2.1 I Report	4
2.2 NavigationView	5
3 Nuovo	6
3.1 AdaptsKeyboard()	7
3.2 endEditing()	7
3.3 Core Data	7
3.4 Aggiungi un Report	8
4 Calendario	9
4.1 The sheets	9
5 Riepilogo	11
5.1 Grafici	11
5.2 Tutti i Report	12
6 Notifiche	14

1 Personal health monitor

Nel seguente progetto lo studente è tenuto a implementare un'applicazione interattiva per tenere traccia delle informazioni personali sulla propria salute da salvare nei report giornalieri. In particolare, l'applicazione dovrebbe essere in grado di gestire i report all'interno di un calendario, inviare notifiche e tracciare i dati in base a filtri specifici.

1.1 Creare e modificare report

L'applicazione deve essere in grado di creare, modificare ed eliminare i rapporti sulla salute. I rapporti sulla salute sono riassunti delle informazioni sulla salute tracciati dall'utente che devono essere salvati ogni volta che l'utente ritiene che sia una buona idea e almeno una volta al giorno. Ogni rapporto deve includere un numero minimo di due informazioni relative alla salute dell'utente (ad es. Temperatura corporea, pressione sanguigna, indice glicemico, ecc.). Ogni informazione ha un'importanza, cioè un indice che specifica il livello di attenzione che richiede tale parametro (da 1 a 5) e ogni rapporto ha una nota opzionale che può essere riempita con informazioni ausiliarie. I report vengono archiviati dall'applicazione (si consiglia vivamente di utilizzare un database) e deve esserci la possibilità di mostrare report su base giornaliera, come ad esempio all'interno di un calendario. Nel caso di più report per lo stesso giorno, è necessario creare un report di riepilogo, in cui le informazioni sulla salute sono la media di tutti i dati raccolti di quel giorno. Ci deve essere anche la possibilità di visualizzare i report in base ad alcuni filtri (ad esempio, solo i report con importanza impostata su 5).

1.2 Notifiche

L'applicazione deve avvisare l'utente se non ha ancora inserito un rapporto per quel giorno. In questo caso, l'utente può eseguire le seguenti azioni: rinviare il promemoria (in tal caso all'utente verrà richiesta un'altra ora e data dello stesso giorno) o aprire direttamente dall'interno della notifica il modulo per la compilazione del rapporto. Il tempo in cui la notifica viene inviata dall'applicazione può essere impostato dall'utente da una pagina delle impostazioni. L'applicazione deve inoltre informare l'utente se la media dei dati raccolti per un'informazione - con importanza maggiore di 3 - in un determinato periodo di tempo ha superato una soglia predefinita. L'utente può personalizzare i parametri precedenti da una pagina delle impostazioni, ovvero può decidere quali informazioni devono essere monitorate, per quanto tempo e quale soglia non deve essere raggiunta.

1.3 Grafici

L'applicazione dovrebbe essere in grado di raccogliere statistiche sull'utilizzo che visualizzano almeno due grafici di qualsiasi tipo (grafico a torta, diagramma a riquadri, istogramma, grafico a linee, ecc.) Che mostrano dati utili (ad esempio la variazione di informazioni sanitarie nell'arco di una settimana, la variazione di il numero di rapporti raccolti ogni giorno, ecc.).

2 HealtMonitor per iOS

Questa versione di HealtMonitor è stata pensata e sviluppata per **iOS 13** usando **Swift 5** su **xCode 11.7**, il tutto su **MacOS Catalina 10.15.6**.

HealtMonitor per iOS è stata testata su **iPhone 11** aggiornato ad **iOS 13.6.1**. Inoltre è stata testata su diversi dispositivi iOS sul **Simulatore**, versione **11.7**, che offre xCode.

L'app è divisa in 3 sezioni principali richiamate nel **ContentView.swift** :

- **Summary()**
- **CalendarTab()**
- **AddReport()**

Ogni view viene invocata tramite una **TabView**, in "*Riepilogo*" abbiamo i grafici dei valori inseriti e la lista di tutti i report, in "*Calendario*" abbiamo un calendario organizzato per mesi dove si può controllare i giorni in cui si è inserito il report, infine possiamo aggiungere un nuovo report nell'ultima tab "*Nuovo*".



Figura 1: Schermate Iniziali

2.1 I Report

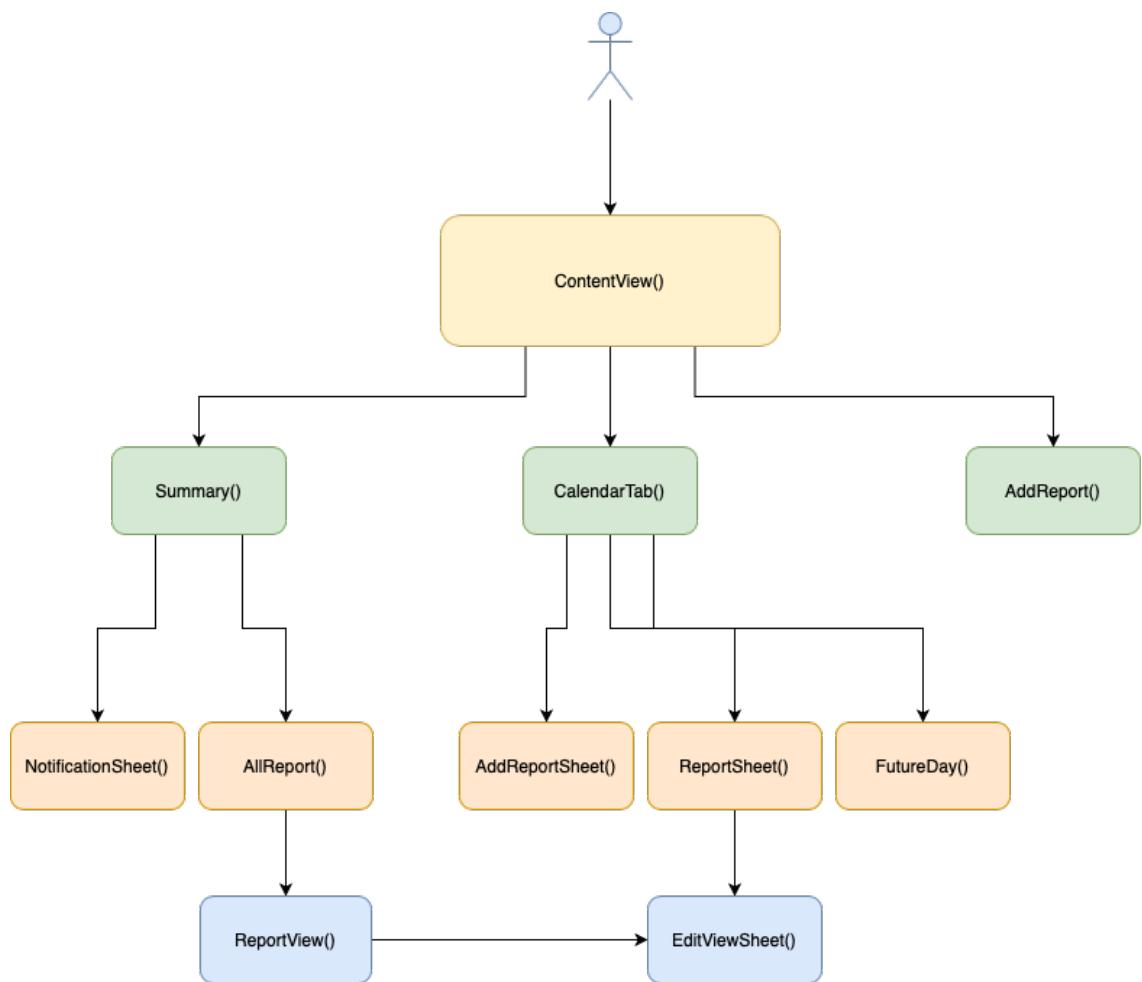
Nella nostra applicazione iOS è possibile aggiungere, modificare ed eliminare dei report giornalieri.

I report sono composti da 4 valori e ogni valore ha un importanza che va da 1 a 5. In particolare i valori da inserire possono essere:

- temperatura
- battito cardiaco
- glicemia
- frequenza respiratoria

Inoltre ogni report è caratterizzato da una data, un titolo (obbligatori) e una descrizione (facoltativa).

2.2 NavigationView



3 Nuovo

La Tab "Nuovo" è una delle più importanti per il funzionamento della nostra app. In HealthMonitor vogliamo aggiungere nuovi report quindi questa operazione è stata semplificata il più possibile con una tab specifica.

Purtroppo **iOS 13 e Swift 5** non ci sono venuti incontro quindi diverse funzioni di controllo si sono rese necessarie per la gestione della tastiera.

In questa tab si può aggiungere un nuovo report selezionando il giorno sul calendario, e compilando correttamente i campi obbligatori. C'è una funzione di controllo `validateForm()` che ritorna un Bool, essa ha il compito di sbloccare il tasto "Aggiungi Report" solo quando tutte le voci inserite rispettano i criteri specificati, quindi ritornerà true sse:

- è stato inserito un titolo
- è stata inserita una temperatura con un valore tra i 30 ed i 45 gradi
- sono stati inseriti il numero di battiti e la glicemia con valori superiori a 40
- è stata inserita una frequenza respiratoria con valore superiore a 10

false altrimenti.



Figura 2: Nuovo report

3.1 AdaptsKeyboard()

Per gestire le tastiere sono state scritte due funzioni fondamentali la prima è `AdaptsKeyboard()`. Essa, quando vengono toccati dei `TextField`, controlla se la tastiera "copre" il campo dove si dovrebbe scrivere ed in quel caso porta in alto la View appena sopra la tastiera non comprende con quest'ultima la `TextField`. Ad esempio quando andiamo ad aggiungere la frequenza respiratoria essendo un campo che si trova alla fine del foglio quando salirà la tastiera non sarà più visibile, qui entra in gioco `AdaptsKeyboard()` che richiamata alla fine della View controlla tutti campi.

3.2 endEditing()

Un'altra funzione fondamentale per facilitare l'utilizzo delle tastiere è `endEditing()`.

Questa funzione si occupa di chiudere la tastiera quando si è finito a scrivere. Essa richiamata nella classe `SceneDelegate` nel quale specifichiamo che se viene toccato due volte di fila lo schermo, la tastiera deve essere chiusa.

3.3 Core Data

I report vengono salvati e mantenuti in memoria attraverso i Core Data. E' stata creata una nuova entità dal nome `Report` che presenta in essa i 12 attributi necessari per il salvataggio dei dati.



Figura 3: `AdaptsKeyboard()`

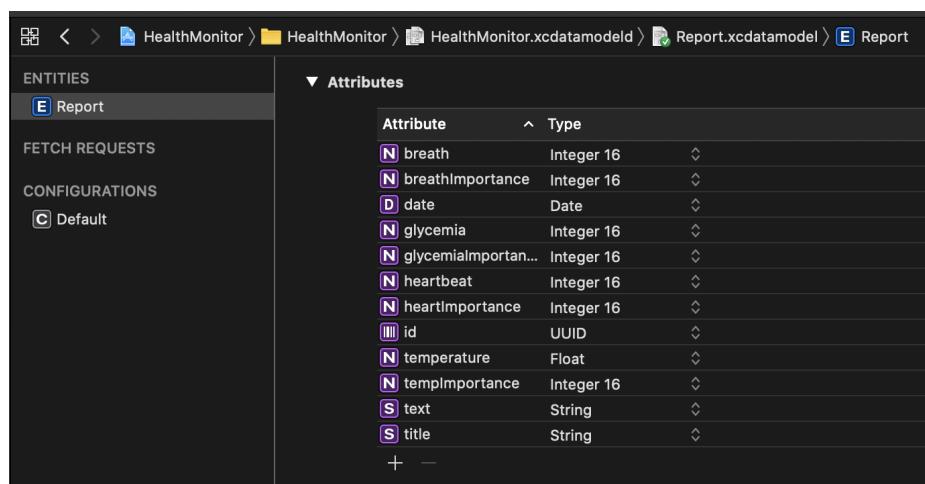


Figura 4: Core Data

3.4 Aggiungi un Report

```
Button(action: {
    self.controlReport()
    self.clearField()
}) {
    ButtonView()
}.disabled(!self.validateForm())
.alert(isPresented: $showingAlert) {
    Alert(title: Text("Il tuo report è stato aggiunto"), message: Text("Puoi
modificarlo nel calendario o in riepilogo"), dismissButton:
    default(Text("Capito")))
}
```

Quando viene spinto il pulsante "Aggiungi un report" vengono fatti alcuni controlli prima che il report venga aggiunto.

`controlReport()` si occupa di controllare se il report inserito per quella specifica data è già presente o meno. Se il report è presente verrà calcolata una media tra il report vecchio e quello nuovo, se il report non è presente verrà aggiunto un nuovo report.

`avgReport(report:Report)` è la funzione che, prendendo in input un report calcola la media tra i dati presenti all'interno del report passato come input e i dati che sono stati inseriti nel report, quindi aggiorna il report per quella data con i nuovi dati ed infine salva.

`newReport()` si occupa di creare un nuovo report, inserisce all'interno di ogni campo del Core Data i valori che sono stati messi nel report.

`clearField()` una volta salvato il report, `clearField()` si occupa di svuotare i `TextField` dai valori presenti.

`disabled(!self.validateForm())` il bottone risulta disabilitato finche i dati inseriti non sono rispettano i criteri imposti.

Infine viene stampato a video un alert che comunica che il report è stato inserito.

4 Calendario

Al centro delle tre `TabView` troviamo il Calendario che, stilisticamente parlando, è l'oggetto più complesso dell'app.

Tramite il calendario possiamo aggiungere eliminare e modificare i report. Inoltre la grafica colorata del calendario ci permette di comprendere i giorni che presentano report ed i giorni senza report.

La grafica e la gestione del calendario sono state implementate grazie ad **RKCalendar**, non trattandosi di una libreria il codice è stato ripulito ed adattato per lo sviluppo di HealthMonitor.

Il Calendario è invocato tramite

`CalendarController(reports: self.reports, CalendarManager: self.CalManager)` prendendo in input i report salvati e una classe nel quale sono state inseriti e la data di inizio, di fine calendario ed il tipo di calendario.

In `CalendarController` per ogni mese presente viene richiamata la view `CalendarMonth()` che è il cuore del nostro calendario, essa non solo si occupa di costruire graficamente il calendario, ma si occupa anche di controllare i giorni e quindi segnalare il giorno corrente, se è presente un report ed il giorno selezionato. I giorni verranno colorati in base a determinate proprietà:

- nessun colore → nessun report presente → è possibile aggiungere un nuovo report
- giallo → report presente → è possibile visualizzare, modificare ed eliminare il report
- rosso → giorno corrente
- verde → giorno selezionato

4.1 The sheets

Nella view "Calendario" non ci spostiamo verso nuove view, ma apriamo solo delle `sheet`. Come accennato prima il calendario è gestito tutto in `CalendarMonth()`. Qui troviamo dei controlli fondamentali:

```
.sheet(isPresented: self.$showSheet) {
    if self.isReport(date: self.CalendarManager.selectedDate ?? Date()) {
        ReportSheet(showSheet: self.$showSheet, reports: self.reports,
                    date: self.CalendarManager.selectedDate)
        .environment(\.managedObjectContext, self.managedObjectContext)
    }
    else if (self.CompareDate(date: Date(), referenceDate: self.
        CalendarManager.selectedDate!)) {
        AddReportSheet(showSheet: self.$showSheet, date: self.
        CalendarManager.selectedDate ?? Date(), reports: self.reports)
        .environment(\.managedObjectContext, self.managedObjectContext)
    } else {
        FutureDay(showSheet: self.$showSheet)
    }
}
```

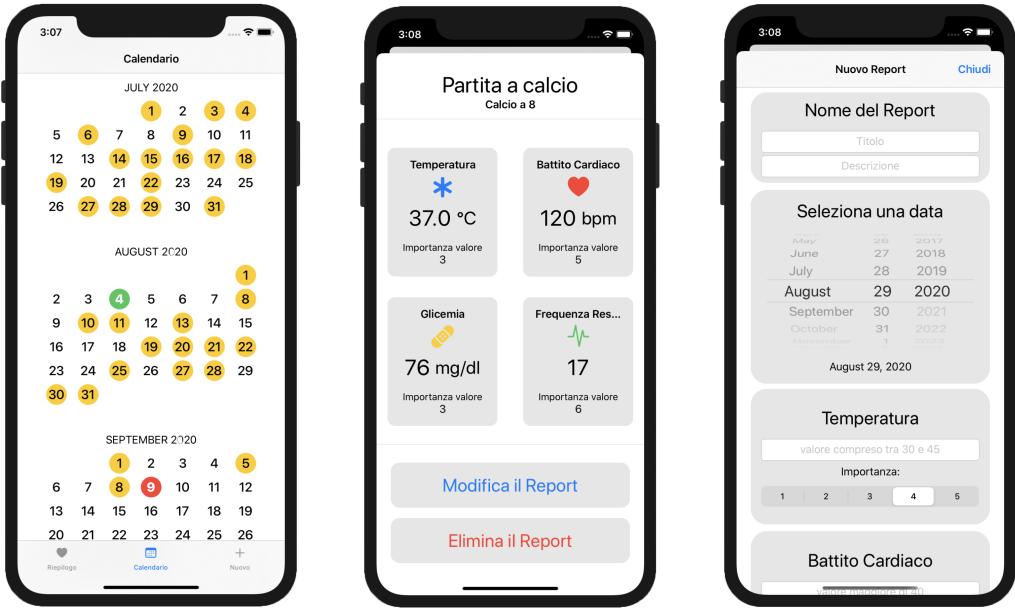


Figura 5: Calendario con report/Sheet con dettagli report/Sheet con aggiunta report

Quando una data viene toccata `showSheet` diventa true e quindi viene mostrata la sheet in base alle caratteristiche del giorno:

```
if self.isReport(date: self.CalendarManager.selectedDate ?? Date())
```

Se nel giorno toccato è presente il report allora verrà mostrato `ReportSheet()` con le informazioni del report presente per quel giorno.

In `ReportSheet()` possiamo, modificare o eliminare il report corrente. Per la modifica del report abbiamo `EditViewSheet()` che prendendo in input tutti valori del report, li inserisce nei vari `TextField` e quindi da la possibilità di cambiare i valori e di inserirne dei nuovi.

```
else if (self.CompareDate(date: Date(), referenceDate: self.CalendarManager.selectedDate!))}
```

Se il report per quel giorno non è presente allora ci sono due casi, è una data del calendario presente o passata o è una data del calendario futura.

Nel caso che è una data del calendario presente o passata si ha la possibilità di aggiungere un nuovo report grazie a `AddReportSheet()` che è molto simile a `AddReport()` ma è ripensata per una `sheet`.

Nel caso si tratta di una data futura allora verrà caricato `FutureDay()`, una sheet semivuota che avvisa che non si possono aggiungere report futuri.

5 Riepilogo

In riepilogo vengono generati i grafici relativi ai dati inseriti giorno per giorno inoltre è presente una lista dove si possono consultare tutti i Report, essi possono essere visualizzati tramite alcuni filtri, possono essere modificati ed eliminati.

Se non è presente nessun dato viene caricata una schermata iniziale di benvenuto.



Figura 6: Schermata di benvenuto

5.1 Grafici

Quando si inseriscono i dati, si iniziano a generare 4 grafici uno per ogni valore presente nel Report.

I grafici vengono generati usando una libreria Swift dal nome *SwiftUICharts*, esso ha diverse configurazioni, in particolare in HealthMonitor troviamo `LineChartView()` che prende in input array di dati.

I valori dei grafici sono ordinati per data, basta trascinare il dito sopra al grafico per visualizzare i valori.



Figura 7: Grafici con valori

5.2 Tutti i Report

Nella schermata iniziale è presente un `NavLink` che ti permette di spostarti tra le View, toccandoci sopra si caricherà `AllReport()` nel quale troviamo una lista completa dei Report.

La lista completa dei report in `AllReport()` è ordinata per data all'interno di una `List` nel quale ritroviamo i `NavLink` che ci permettono di accedere ai dettagli dei report, inoltre si possono cancellare i report trascinando l'elemento verso sinistra.

I report possono essere visualizzati anche tramite dei filtri, che calcolano la media dell'importanza dei 4 valori inseriti e restituisce una lista più corta. La funzione che si occupa del filtraggio dei report è definita nel seguente modo:

```
private func filterReport() -> [FetchedResults<Report>.Element] {
    return self.reports.filter{
        avgImp(tempImp: $0.tempImportance, heartImp: $0.heartImportance,
               glyImp: $0.glycemiaImportance, breImpo: $0.breathImportance) >= self.
        avgImportance
    }
}

func avgImp(tempImp: Int16, heartImp: Int16, glyImp: Int16, breImpo: Int16)
-> Int16 {
    let avgImp : Int16 = (tempImp + heartImp + glyImp + breImpo)/4
    return avgImp
}
```

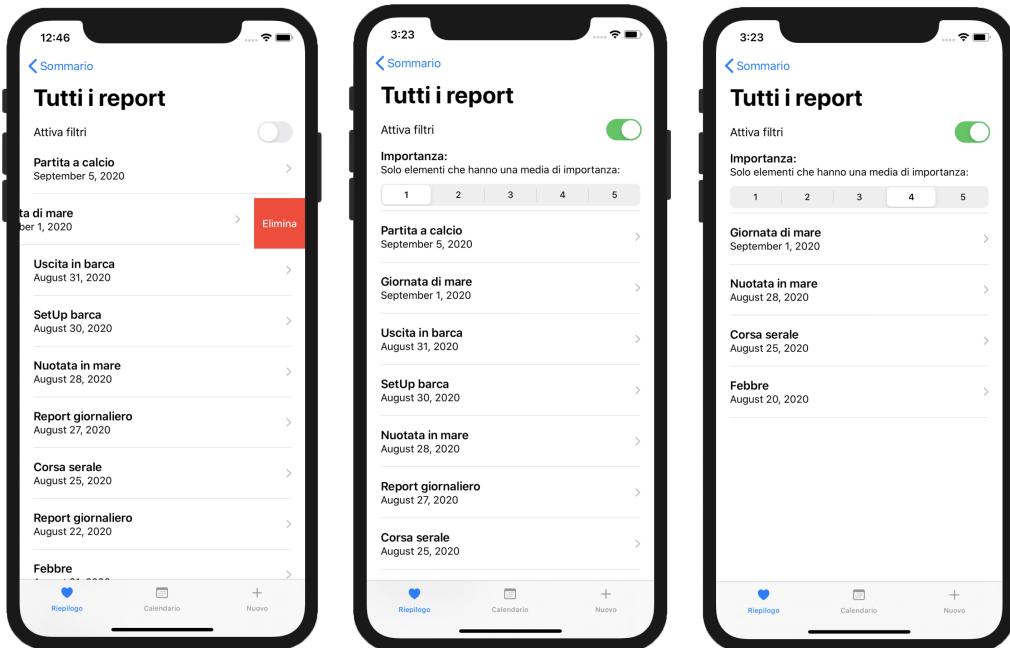


Figura 8: Lista Report

Toccando un oggetto della lista verrà caricata un nuova view `ReportView()` nel quale troviamo i dettagli del report selezionato. In `ReportView()` è possibile modificare il report premendo sul bottone "Modifica il Report", esso caricherà a schermo una nuova `sheet` e prendendo in input tutti i dati del report corrente verrà mostrato `EditViewSheet()`. Tramite questa `sheet` è possibile salvare le modifiche con un bottone a fine pagina, se invece si cambia idea basterà trascinare in basso la `sheet` o toccare il bottone "Chiudi".



Figura 9: Dettagli del report/Modifica del report/Report modificato

6 Notifiche

In HealthMonitor abbiamo la possibilità di ricevere delle notifiche.

Le notifiche vengono gestite in `NotificationSheet()`. Qui si viene avvisati se le notifiche sono attive o disattive. Nel caso sono attive tramite un `DatePicker` si ha la possibilità di selezionare l'orario in cui si vogliono ricevere le notifiche. Una volta attive le notifiche sono su base giornaliera. Ogni 24h l'app ricorderà all'utente di aggiungere un report.

```
func sendNotification(start: Date) ->(){
    let content = UNMutableNotificationContent()
    content.title = "Report Giornaliero"
    content.body = "Cosa aspetti? E' giunto il momento di aggiungere il report giornaliero!"
    content.sound = UNNotificationSound.default

    var hourStart = DateComponents()
    hourStart.hour = Calendar.current.component(.hour, from: start)
    hourStart.minute = Calendar.current.component(.minute, from: start)

    let trigger = UNCalendarNotificationTrigger(dateMatching: hourStart,
repeats: true)
    let request = UNNotificationRequest(identifier: "Notification", content:
content, trigger: trigger)
    UNUserNotificationCenter.current().add(request, withCompletionHandler:
nil)
}
```

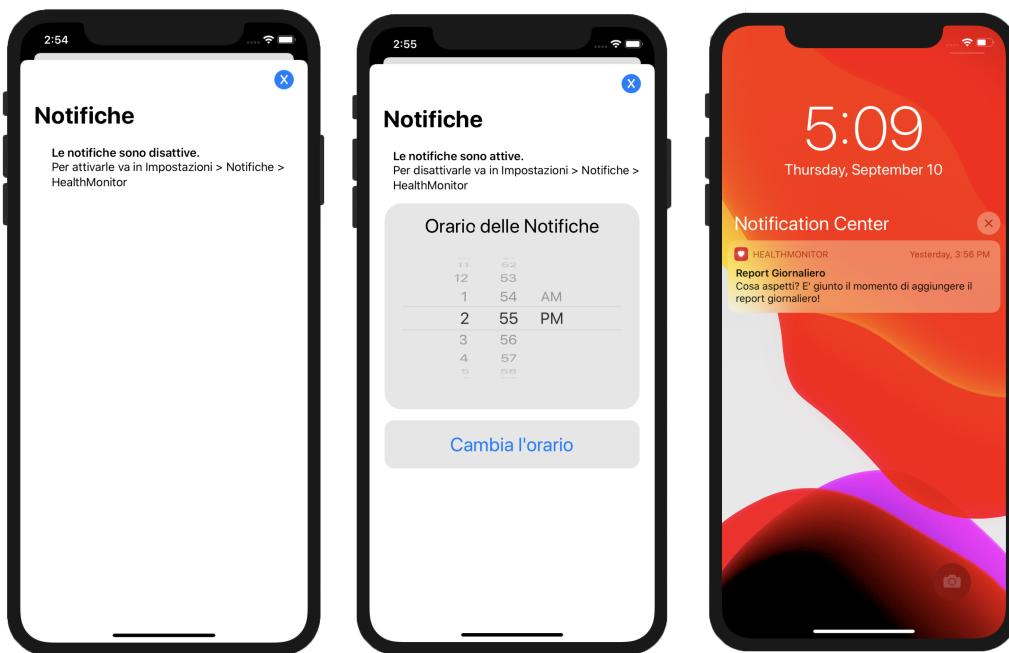


Figura 10: Notifiche disabilitate/Notifiche abilitate/Notifica