

Report Progetto

Laboratorio di Applicazioni Mobili

Riccardo Cavallin
825624

Scopo dell'applicazione

Il progetto consiste nello sviluppo di un applicazione interattiva iOS per il tracciamento personale della salute attraverso dei report sanitari giornalieri.

I report sono consultabili attraverso un calendario e alcuni grafici.

Inoltre l'applicazione ricorda all'utente di inserire il report del giorno nel caso in cui se lo sia dimenticato e lo notifica di eventuali valori fuori norma.

Funzionalità previste

Il progetto è strutturato attorno a tre punti principali:

- **Report** : è possibile inserire, modificare ed eliminare un report. Questi report sono dei raccoglitori di informazioni riguardanti parametri vitali che l'utente decide di salvare nell'applicazione.
Ognuno di essi contiene almeno due rilevazioni di valori a scelta fra temperatura, pressione massima, pressione minima, glicemia e battito cardiaco in aggiunta ad una nota opzionale.
Ad ogni informazione è associato un indice di importanza che serve ad associare una priorità complessiva al report. Nel caso ci siano più di due report per un determinato giorno è possibile visualizzarne un riassunto.
Infine è possibile filtrare i report in base alla loro priorità.
- **Grafici** : l'applicazione ha una sezione dedicata per due grafici che mostrano l'andamento di due parametri sanitari nel corso dell'ultima settimana.
- **Notifiche** : l'utente è in grado di specificare un orario nel quale vuole che gli sia mandata una notifica che gli ricordi di inserire il report giornaliero.
Inoltre può decidere attivare una funzione di monitoraggio relativa ad uno specifico indice indicandone la durata e un valore soglia. Al termine del periodo indicato l'utente riceverà una notifica con l'esito del controllo.

Progettazione

L'applicazione è stata pensata e testata per funzionare su un iPad 9,7" 2018.

Per lo sviluppo si è utilizzato Xcode 11.4.1 e Swift 5.2 su macOS Catalina 10.15.4.

Il database utilizzato per la gestione dei report è Core Data.

Il pattern Model View Controller

Al fine di aderire alle linee guida Apple si è utilizzato il pattern Model View Controller, separando fisicamente e logicamente le parti dell'applicativo che si occupano del salvataggio e gestione dei dati dalla parte logica e grafica.

Il gruppo di file "View" contiene solamente *Main.storyboard*, cioè colui che si occupa esclusivamente dell'interfaccia grafica dell'applicativo che è stata costruita prevalentemente con l'ausilio dell'Interface Builder.

Nel gruppo di file "Controller" sono invece raggruppate tutte le classi che si occupano di gestire l'interfaccia grafica adattandola al contesto, schedulare le notifiche e ricevere dati dal "Model".

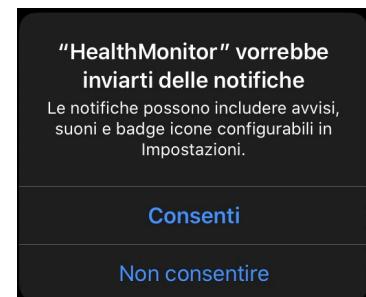
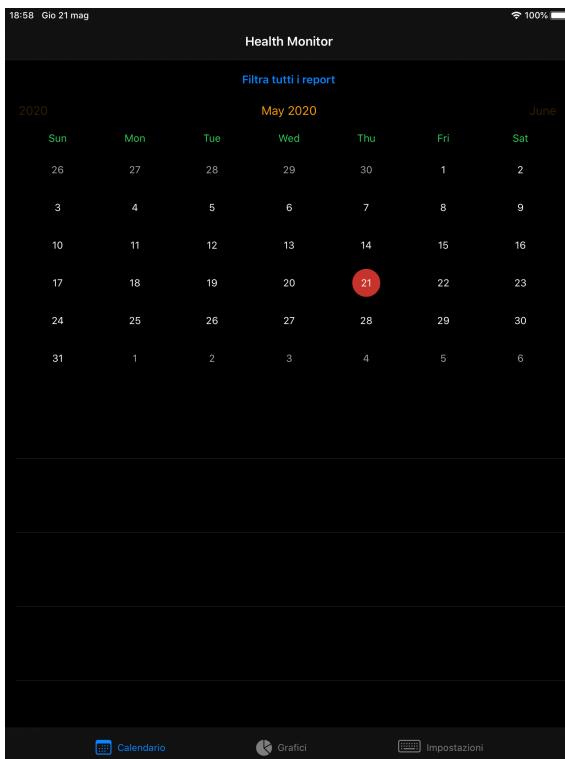
In quest'ultimo gruppo di file troviamo invece il database *HealthMonitor.xcoreddatamodeld* in accoppiata con la classe *Report*. È presente anche un ultima classe un po' particolare, *Retrieve*, che si occupa di modellare i dati e restituirli al Controller che li richiede.

In questo modo né l'interfaccia né la logica applicativa sono legate strettamente ai dati in memoria che risultano essere parametrizzati.

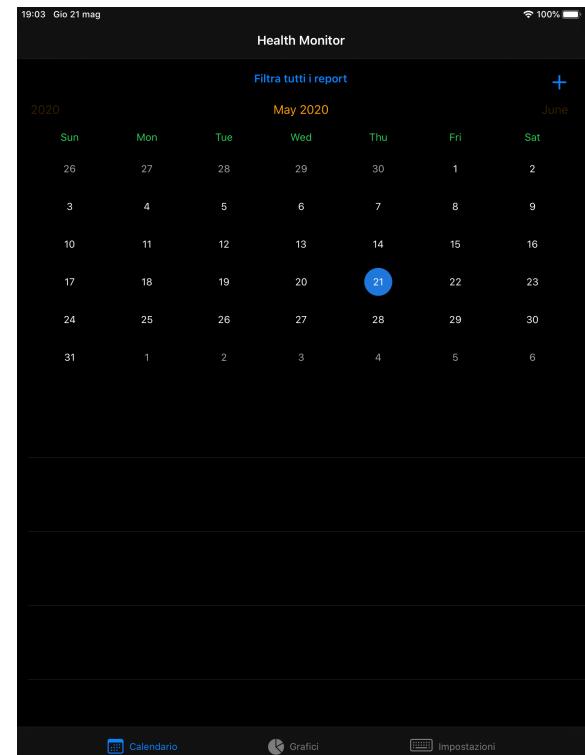
L'ultimo gruppo di file è chiamato "Supporting Files" e come dice il nome racchiude tutti i file vari rimanenti come file di configurazione e assets. A tal proposito degni di nota sono *AppDelegate* e *SceneDelegate* che permettono di eseguire operazioni in base al contesto corrente dell'applicazione.

Il primo per esempio è stato utilizzato per poter richiedere i permessi di invio notifiche mentre il secondo per specificare il comportamento d'interazione con le notifiche stesse.

Applicazione e interfaccia

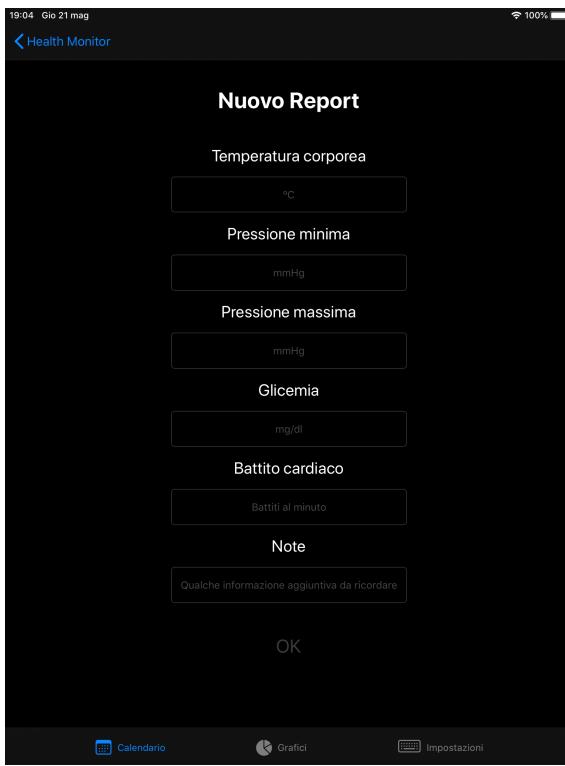


Alla prima apertura dopo l'installazione l'applicazione richiede il permesso di inviare delle notifiche e si presenta nel seguente modo.



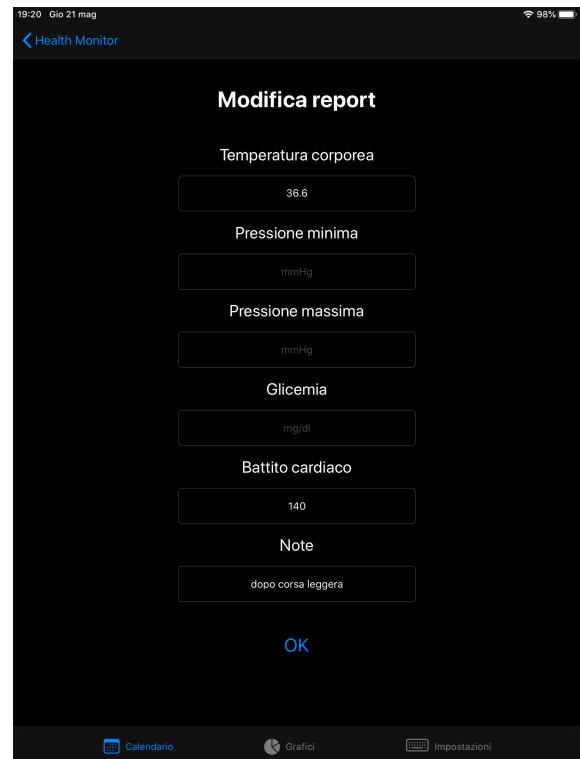
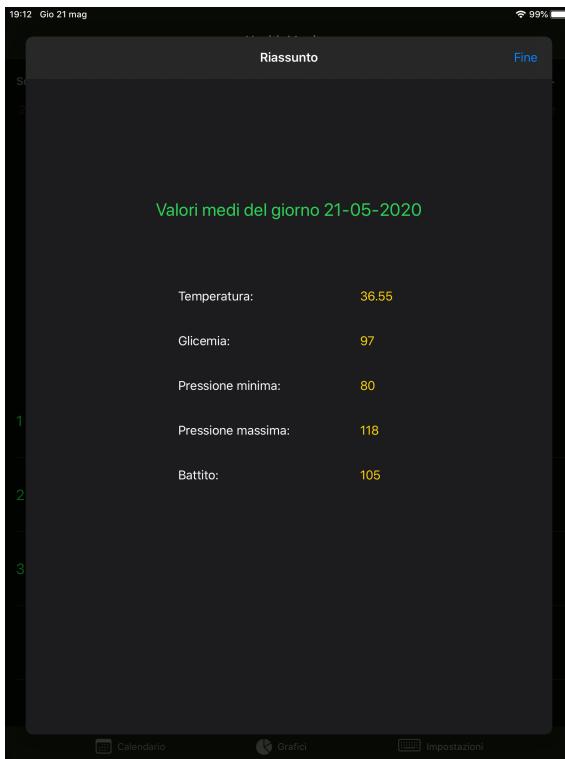
Cliccando sul numero di un giorno precedente o uguale ad oggi compare un bottone ("+") per poter inserire un nuovo report.

Per la realizzazione del calendario è stato utilizzato il Pod "FSCalendar".

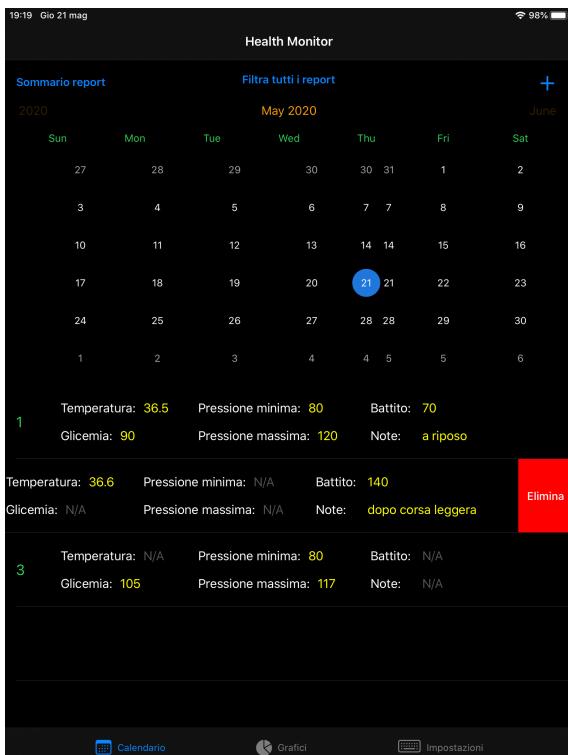


Sommaario report							Filtra tutti i report	+ June
May 2020								
Sun	Mon	Tue	Wed	Thu	Fri	Sat		
26	27	28	29	30	1	2		
3	4	5	6	7	8	9		
10	11	12	13	14	15	16		
17	18	19	20	21	22	23		
24	25	26	27	28	29	30		
31	1	2	3	4	5	6		
1	Temperatura: 36.5	Pressione minima: 80	Battito: 70					
	Glicemia: 90	Pressione massima: 120	Note: a riposo					
2	Temperatura: 36.6	Pressione minima: N/A	Battito: 140					
	Glicemia: N/A	Pressione massima: N/A	Note: dopo corsa leggera					
3	Temperatura: N/A	Pressione minima: 80	Battito: N/A					
	Glicemia: 105	Pressione massima: 117	Note: N/A					

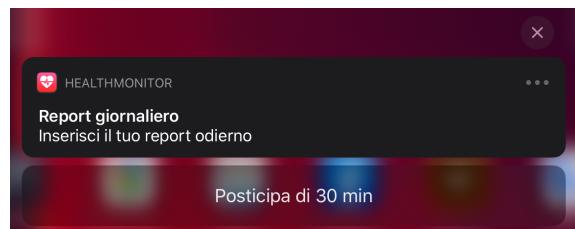
Nella schermata di inserimento è possibile specificare un massimo di 5 parametri. È necessario inserire i valori nei campi corrispondenti e poi premere "OK". Si noti che il bottone diventa cliccabile solamente quando due campi sono stati compilati. Inserendo alcuni report la Table View scrollabile inizia a popolarsi. I valori presenti vengono evidenziati in giallo e quelli non assegnati con la dicitura "N/A" in grigio.



Nel momento in cui vengono inseriti almeno due report per una giornata compare in alto un bottone "Sommario report". Cliccandolo si apre una schermata pop-up riportante la media per ogni parametro dei valori inseriti durante la giornata. Cliccando su di un report (cioè su di una cella della Table View) è possibile modificare i valori associati a quel report.

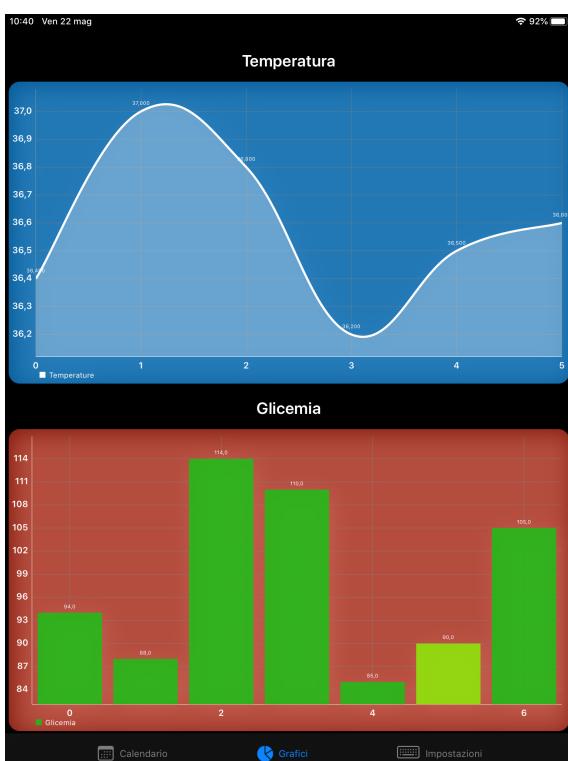
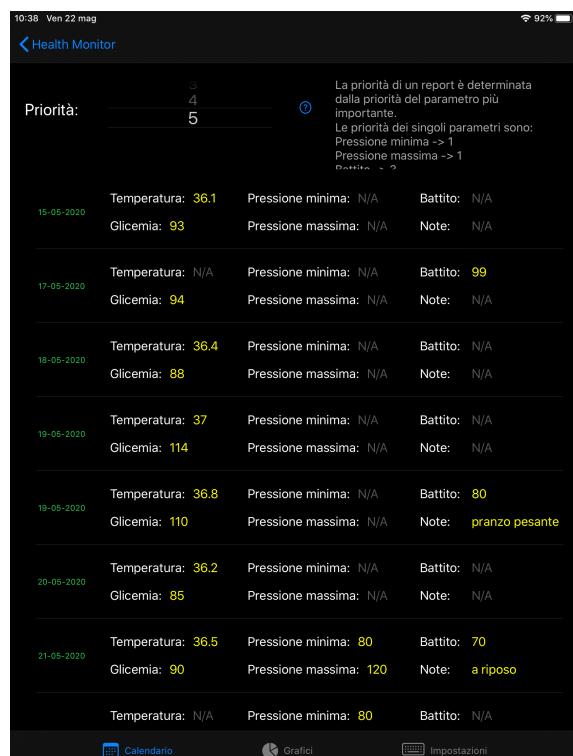


Cliccando invece su "Filtro tutti i report" si apre una schermata che permette di visualizzare tutti i report associati ad uno specifico livello di priorità.

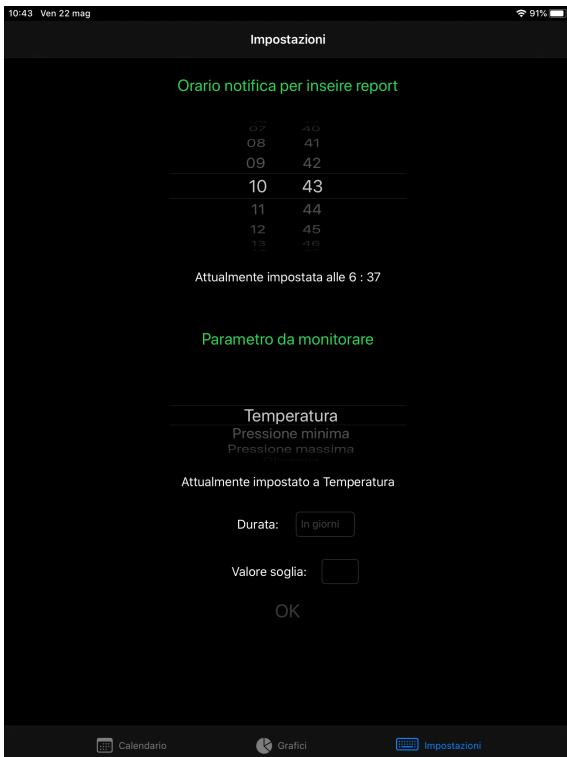


Effettuando uno swipe da destra a sinistra in corrispondenza di una cella è possibile eliminare un report.

Una notifica che arriverà all'orario impostato ricorda di inserire il report se non è già stato inserito. Cliccandola si verrà portati direttamente alla schermata di inserimento del report. È possibile inoltre posticiparla di 30 minuti.



Il secondo tab "Grafici" permette di visualizzare i grafici di andamento di temperatura e glicemia nel corso dell'ultima settimana. Nell'asse delle ascisse è riportato il numero di misurazioni mentre nell'asse delle ordinate il valore misurato corrispondente. Per la realizzazione dei grafici è stato utilizzato il pacchetto "Charts".



Nel terzo e ultimo tab “Impostazioni” è possibile personalizzare il comportamento dell’app specificando un orario nel quale si vuole ricevere la notifica che ricordi di inserire un report. Inoltre si può attivare la funzione di monitoraggio specificando un parametro da tenere sotto controllo, la durata dell’analisi e un valore soglia di riferimento.



Alla scadenza fissata si riceverà una notifica riassuntiva con l’esito del monitoraggio.

Realizzazione

Il database

La funzione del database è quella di memorizzare tutte le informazioni di ogni singolo report inserito dall’utente. La scelta è ricaduta su Core Data, framework messo a disposizione da Apple per salvare gli oggetti in modo permanente.

Nello specifico si è deciso di adottare questa soluzione perché, seppur più complessa, offre migliori garanzie a livello privacy rispetto all’utilizzo della classe UserDefaults.

Quest’ultima risulta infatti molto comoda per salvare piccole quantità di dati ma essi risultano essere accessibili in lettura anche da altre applicazioni. Essendo i report sanitari informazioni sensibili la soluzione Core Data è preferibile.

UserDefault è tuttavia stato usato per salvare piccole informazioni riguardo alle preferenze dell’utente su orari di notifica e valore da monitorare. In questo caso le informazioni infatti non sono critiche e un meccanismo chiave-valore risulta conveniente.

Attributes

Attribute	Type
N battito	Integer 16
D data	Date
N glicemia	Integer 16
S note	String
N pressioneMax	Integer 16
N pressioneMin	Integer 16
N priorita	Integer 16
N temperatura	Decimal

Gli attributi salvati per ogni report.

Retrieve

Retrieve è una classe che si occupa di recuperare ed elaborare dei dati dal database. Essa mette a disposizione dei metodi pubblici per recuperare tali informazioni.

```
// estraggo tutti i report dell'ultima settimana
func findLastWeekReports() -> [Report]? {
    let today = NSDate()
    let aWeekAgo = today.addingTimeInterval(-7*24*60*60)
    let request: NSFetchedResultsController<Report> = Report.fetchRequest()
    // estraggo solo i report dell'ultima settimana
    request.predicate = NSPredicate(format: "data > %@ AND data < %@", aWeekAgo, today)
    // ordinamento per data
    request.sortDescriptors = [NSSortDescriptor(key: "data", ascending: true)]
    // se fallisce restituisce nil
    return try? context.fetch(request)
}
```

All'interno di questa classe vengono effettuate la maggior parte delle query al database e un po' di computazione come ad esempio il calcolo dei valori medi.

Esempio di funzione che restituisce tutti i report dell'ultima settimana ordinando il risultato di una query che filtra per la data. È richiamata per ricavare i dati da plottare nei grafici.

```
func mediaBattito(reports: [Report]) -> Int? {
    let count = Int16(reports.filter({ $0.battito > 0 }).count)
    return (count > 0) ? Int((reports.reduce(0, {$0 + $1.battito})) / count) : nil
}
```

Esempio di funzione che computa la media del battito considerando solo i valori maggiori di 0.

FirstViewController

È il file del View Controller principale che gestisce la schermata iniziale contenente il calendario, la Table View dei reports e i vari bottoni.

```
func deleteAction(at indexPath: IndexPath) -> UIContextualAction {
    let action = UIContextualAction(style: .destructive, title: "Elimina") { (action, view, completion) in
        // eliminazione dal database
        self.context.delete(self.dailyReports![indexPath.row])
        // eliminazione dal vettore
        self.dailyReports?.remove(at: indexPath.row)
        // eliminazione dalla tabella
        self.tableView.deleteRows(at: [indexPath], with: .automatic)
        if self.dailyReports?.count == 0 {
            // se non ci sono più report devo rimettere la notifica per il giorno odierno
            let date = Date()
            let calendar = Calendar.current
            let dayCurrent = calendar.component(.day, from: date)
            let hour = self.defaults.integer(forKey: "oraNotificaReport")
            let minute = self.defaults.integer(forKey: "minutoNotificaReport")
            self.notificationPublisher.sendReportReminderNotification(title: "Report giornaliero", body: "Inserisci il tuo report odierno", badge: 1, sound: .default, day: dayCurrent, hour: hour, minute: minute, id: "reportReminder", idAction: "posticipa", idTitle: "Posticipa")
        }
        completion(true)
    }
    action.backgroundColor = .red
    return action
}
```

Un esempio delle funzioni di FirstViewController. La funzione qui sopra si occupa di eliminare il report corrispondente alla riga della tabella oggetto dell'azione di swipe.

Inoltre se il vettore dei report giornalieri è vuoto programma una notifica all'orario stabilito per ricordare all'utente di inserire un report.

All'interno di questo file Swift è presente anche una classe ReportTableViewCell che definisce il comportamento delle celle della Table View.

SecondViewController

È il View Controller della schermata dei grafici: ne specifica lo stile e converte i dati nel formato giusto perché possano essere visualizzati.

```
private func customizeTempChart() {
    temperatureChart.backgroundColor = .black
    temperatureChart.rightAxis.enabled = false
    let asseY = temperatureChart.leftAxis
    asseY.labelFont = .boldSystemFont(ofSize: 12)
    asseY.setLabelCount(10, force: false)
    asseY.labelTextColor = .white
    asseY.axisLineColor = .white
    asseY.labelPosition = .outsideChart
    temperatureChart.xAxis.labelFont = .boldSystemFont(ofSize: 12)
    temperatureChart.xAxis.labelPosition = .bottom
    temperatureChart.xAxis.setLabelCount(4, force: false)
    temperatureChart.xAxis.labelTextColor = .white
    temperatureChart.xAxis.axisLineColor = .white
    temperatureChart.animate(xAxisDuration: 0.5)
}

// converte i dati della glicemia nel formato giusto per plottarli
private func yValuesGlycem(glicemia: [Int16?] -> [BarChartDataEntry] {
    var values = [BarChartDataEntry]()
    for i in 0..
```

ThirdViewController

Questo View Controller gestisce il tab delle impostazioni attraverso alcune Labels, due Text Fields e due Pickers. Si occupa inoltre di salvare all'interno di UserDefaults le preferenze dell'utente e di programmare le notifiche all'orario desiderato.

```
@IBAction func valueChanged(_ sender: Any) {
    // rimuovo la notifica precedentemente programmata
    notificationPublisher.deleteNotification(id: "reportReminder")
    let date = hourPicker.date
    let components = Calendar.current.dateComponents([.day, .hour, .minute], from: date)
    let day = components.day!
    let hour = components.hour!
    let minute = components.minute!
    saveHourNotificationPreference(hour: hour, minute: minute)
    if minute < 10 {
        oraImpostata.text = "Impostato alle \u2022(\u2022:0\u2022(\u2022"
    } else {
        oraImpostata.text = "Impostato alle \u2022(\u2022:\u2022(\u2022"
    }
    let data = Calendar.current.date(from: Calendar.current.dateComponents([.year, .month, .day], from: Date()))
    let reportsOggi = model.findReportsByDay(matching: data!)
    // se ho già dei report oggi metto la notifica a domani
    if reportsOggi!.count > 0 {
        // imposto una nuova notifica per il nuovo orario
        notificationPublisher.sendReportReminderNotification(title: "Report giornaliero", body: "Inserisci il tuo report odierno", badge: 1,
            sound: .default, day: day+1, hour: hour, minute: minute, id: "reportReminder", idAction: "posticipa", idTitle: "Posticipa")
    } else { // altrimenti la fisso già a partire da oggi
        notificationPublisher.sendReportReminderNotification(title: "Report giornaliero", body: "Inserisci il tuo report odierno", badge: 1,
            sound: .default, day: day, hour: hour, minute: minute, id: "reportReminder", idAction: "posticipa", idTitle: "Posticipa")
    }
}
```

Quando cambia il valore selezionato nel Picker viene rimossa la notifica precedentemente programmata in favore di una nuova e si aggiorna la Label.

FormViewController

È il Controller del form di inserimento dati. Gestisce quindi tutte le Labels e campi di inserimento nonché il bottone di conferma. Inoltre salva nel database i dati inseriti e invia una notifica riassuntiva del monitoraggio nel caso in cui si sia giunti alla data di scadenza impostata.

Viene infine richiamato in seguito all'apertura della notifica che ricorda di inserire un report.

```

// salvataggio nel database
do {
    try context.save()
    // tolgo la notifica per oggi e la imposto per domani
    notificationPublisher.deleteNotification(id: "reportReminder")
    let date = Date()
    let calendar = Calendar.current
    let dayCurrent = calendar.component(.day, from: date)
    let hour = defaults.integer(forKey: "oraNotificaReport")
    let minute = defaults.integer(forKey: "minutoNotificaReport")
    // imposto una nuova notifica per domani
    notificationPublisher.sendReportReminderNotification(title: "Report giornaliero", body: "Inserisci il tuo report odierno", badge: 1,
        sound: .default, day: dayCurrent+1, hour: hour, minute: minute, id: "reportReminder", idAction: "posticipa", idTitle: "Posticipa")
} catch {
    fatalError("Errore nel salvataggio: \(error)")
}

checkForMonitor()

```

Dopo aver acquisito gli input dall'utente vengono salvati nel database.

Inoltre viene rimossa la notifica per il giorno corrente in favore di una per il giorno successivo.

EditFormViewController

Ha funzionalità analoghe rispetto al precedente con la differenza che permette di modificare dei dati già inseriti. Infatti viene richiamato in seguito ad un tap su di una cella della Table View nella schermata principale.

SummaryPopUpViewController

Recupera i dati medi giornalieri da Retrieve e li inserisce in una finestra pop-up riassuntiva per uno specifico giorno.

NotificationPublisher

È la classe che gestisce imposta e programma di fatto le notifiche. Viene richiamata da molte delle precedenti passando come parametri ai suoi metodi le stringhe da visualizzare come contenuto a schermo delle notifiche.

```

// notifica per ricordare di inserire un report
func sendReportReminderNotification(title: String, body: String, badge: Int, sound: UNNotificationSound, day: Int, hour: Int, minute: Int, id: String, idAction: String, idTitle: String) {

    // azione di posticipa notifica
    let postponeAction = UNNotificationAction(identifier: "posticipa", title: "Posticipa di 30 min" , options: [])

    // categorie
    let category = UNNotificationCategory(identifier: "reportCategory", actions: [postponeAction], intentIdentifiers: [], options: [])

    // aggiungo la categoria al framework delle notifiche
    UNUserNotificationCenter.current().setNotificationCategories([category])

    let content = UNMutableNotificationContent()
    content.title = title
    content.body = body
    // aggiungo 1 al badge di notifica
    content.badge = NSNumber(value: UIApplication.shared.applicationIconBadgeNumber + badge)
    content.sound = sound
    content.categoryIdentifier = "reportCategory"

    var dateComponents = DateComponents()
    dateComponents.calendar = Calendar.current
    dateComponents.day = day
    dateComponents.hour = hour
    dateComponents.minute = minute

    let trigger = UNCalendarNotificationTrigger(dateMatching: dateComponents, repeats: true)

    let id = id
    let request = UNNotificationRequest(identifier: id, content: content, trigger: trigger)
    UNUserNotificationCenter.current().add(request) { (error) in
        if error != nil {
            print(error!.localizedDescription)
        }
    }
}

```

Funzione che permette di schedulare una notifica personalizzata con un'azione "posticipa".

FilterViewController

Il file FilterViewController contiene due classi: la prima delle quali dai cui prende il nome è designata alla gestione del filtraggio dei report in base alla loro priorità. Gli elementi grafici usati sono una Table View, un Picker, un Button ed una Text Area.

La seconda classe ReportCellWithDate permette di visualizzare il correttamente il contenuto delle celle della Table View.

AppDelegate

Un'altra classe degna di nota è AppDelegate. Essa nasce di default con la creazione del progetto ed è stata modificata opportunamente per implementare dei comportamenti da attuare con le notifiche come l'apertura di una determinata View dopo il click oppure l'azione di "Posticipa".

```
let identifier = response.actionIdentifier
let idNotifica = response.notification.request.identifier

if idNotifica == "reportReminder" {
    switch identifier {

        case UNNotificationDismissActionIdentifier: // notifica cancellata
            // tolgo 1 al badge di notifica
            //print("The notification was dismissed")
            completionHandler()

        case UNNotificationDefaultActionIdentifier: // notifica viene aperta
            //print("The user opened the app from the notification")
            // tolgo 1 al badge di notifica
            UIApplication.shared.applicationIconBadgeNumber -= 1

            if let formVC = storyboard.instantiateViewController(withIdentifier: "Form") as? FormViewController,
                let tabBarController = rootViewController as? UITabBarController,
                let navController = tabBarController.selectedViewController as? UINavigationController {
                    navController.pushViewController(formVC, animated: true)
            }

            completionHandler()

        case "posticipa": // utente ha cliccato sull'action per posticipare
            // tolgo 1 al badge di notifica
            UIApplication.shared.applicationIconBadgeNumber -= 1
            let date = Date()
            let calendar = Calendar.current
            let dayCurrent = calendar.component(.day, from: date)
            let hour = defaults.integer(forKey: "oraNotificaReport")
            let minute = defaults.integer(forKey: "minutoNotificaReport")
            // imposto una nuova notifica fra 30 minuti
            notificationPublisher.sendReportReminderNotification(title: "Report giornaliero", body: "Inserisci il tuo report odierno", badge: 1, sound: .default, day: dayCurrent, hour: hour, minute: minute + 30, id: "reportReminder", idAction: "posticipa", idTitle: "Posticipa")
            completionHandler()

        default:
            print("Default case")
            completionHandler()
    }
}
```

Sezione per la gestione dell'interazione con le notifiche

Difficoltà e soluzioni

La difficoltà principale riscontrata è eseguire un processo in background che esegua un controllo sulla data corrente in modo da poter spedire la notifica "reminder".

Questa criticità nasce dal fatto che a lezione non è stato affrontato il problema e che con l'avvento di iOS 13 sono cambiate le norme Apple in materia di processi in background, di conseguenza le risorse disponibili online sono al momento limitate.

La soluzione adottata è quella di schedulare la notifica per l'orario impostato dall'utente all'interno delle impostazioni in modo che si ripeta ogni giorno. Nel caso in cui l'utente inserisca un report ad un orario precedente a quello di invio della notifica l'invio della stessa viene annullato e riprogrammato a partire dal giorno successivo.

Anche per la notifica relativa ai risultati del monitoraggio è stata adottata una soluzione simile, infatti la notifica viene inviata nel giorno della scadenza dopo l'inserimento del primo report giornaliero.