

# Progetto Laboratorio Applicazioni Mobili

Giacomo Orsi

giacomo.orsi2@studio.unibo.it – matricola 875105

## Sommario

Questo documento descrive le funzionalità e le strategie di implementazione adottate nello sviluppo dell'applicazione iOS *MyPantry*, realizzata per il corso [Laboratorio Applicazioni Mobili](#) dell'Alma Mater Studiorum – Università di Bologna, anno accademico 2020/2021.

## Indice

<b>1 Funzionalità</b>	<b>3</b>
1.1 Inserimento dei prodotti . . . . .	3
1.2 Organizzazione della dispensa . . . . .	3
1.3 Informazioni sui prodotti . . . . .	4
1.4 Lista della spesa . . . . .	4
1.5 Notifiche e riepilogo . . . . .	4
<b>2 Strategie di implementazione</b>	<b>7</b>
2.1 Database . . . . .	7
2.1.1 Connessione al database remoto . . . . .	7
2.1.2 Database locale . . . . .	8
2.2 Scansione del <i>barcode</i> . . . . .	9
2.3 Notifiche . . . . .	9
2.4 Widget . . . . .	10
2.5 Apple Watch . . . . .	10
2.6 Documentazione . . . . .	11

## Introduzione

Il progetto realizzato è costituito da un'applicazione, chiamata *MyPantry*, per dispositivi iOS (iPhone, iPad, iPod Touch) che permette all'utilizzatore di monitorare il contenuto della sua dispensa.

Il sistema è stato realizzato utilizzando:

- Xcode 12.4 (build 12D4e)
- Swift 5



Figura 1: L'icona di *MyPantry*

# 1 Funzionalità

L'applicazione *MyPantry* permette all'utente di inserire i prodotti alimentari acquistati e tenere traccia del loro consumo. Nei paragrafi successivi vengono descritte le funzionalità dell'applicazione. Nella figura 2 è possibile vedere la home dell'applicazione su iPhone e su Apple Watch.



Figura 2: L'home di *MyPantry*

## 1.1 Inserimento dei prodotti

L'inserimento dei prodotti avviene attraverso la scansione o la digitazione del *barcode* del prodotto. L'applicazione accede ad un *database* remoto collaborativo per verificare se sia presente un prodotto associato a quell'identificativo. Qualora siano presenti più prodotti, all'utente viene data la possibilità di scegliere il prodotto corretto e la scelta viene registrata sul database condiviso, come descritto nel paragrafo 2.1.1.

Nel caso in cui nessuno dei prodotti inseriti sul database corrisponda al prodotto corretto o qualora non sia presente nessuna corrispondenza per il *barcode* ricercato, l'utente può aggiungere un nuovo prodotto, che viene messo a disposizione degli altri utenti e dunque caricato sul database condiviso. I prodotti contenuti nella dispensa dell'utente, invece, vengono memorizzati su un database locale, come descritto nel capitolo 2.1.2.

## 1.2 Organizzazione della dispensa

Quando l'utente inserisce un prodotto può specificare un'area della casa nella quale riporlo (ad esempio il frigo, il freezer, un determinato armadio,...) e una categoria a cui esso appartiene (pasta, snack, dolci,...). Sia le *categorie* che le aree della casa, denominate *dispense*, sono completamente personalizzabili.

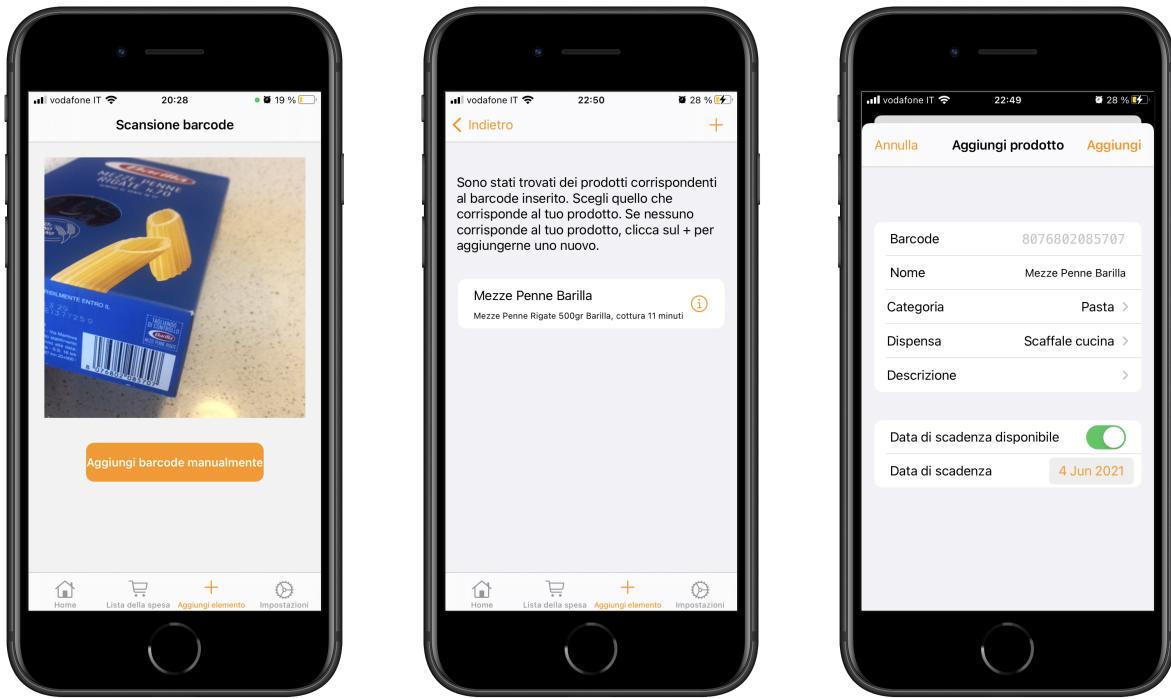


Figura 3: Fasi dell'inserimento di un prodotto

### 1.3 Informazioni sui prodotti

L'obiettivo dell'applicazione è rendere semplice e veloce la gestione dei prodotti in dispensa. Ad un prodotto può essere assegnata una data di scadenza, in modo da poter visualizzare velocemente quali siano i prodotti da consumare prima e ricevere una notifica quando la scadenza si sta avvicinando. L'applicazione tiene traccia anche dell'eventuale apertura dei prodotti, in modo da suggerire all'utente di consumare prima quelli che sono già stati aperti.

Dalla home di *MyPantry*, mostrata in figura 2, è possibile accedere agli elenchi dei prodotti nei quali è anche possibile effettuare una ricerca, come mostrato nella figura 4.

### 1.4 Lista della spesa

I prodotti consumati possono essere inseriti agevolmente nella *lista della spesa*, una sezione dedicata della app che permette di ricordare all'utente quali prodotti deve ri-acquistare. Un prodotto presente nella lista della spesa può essere reinserito nella dispensa in modo istantaneo, senza la necessità di interagire con il database collaborativo.

Si è scelto di consentire l'inserimento nella lista della spesa soltanto di prodotti precedentemente registrati all'interno dell'applicazione.

### 1.5 Notifiche e riepilogo

Al fine di semplificare al massimo la gestione della dispensa e evitare lo spreco di generi alimentari, l'applicazione *MyPantry* prevede un sistema di notifiche che avvisa gli utenti quando un prodotto presenta una data di scadenza ravvicinata. Inoltre, è stato realizzato un Widget che l'utente può inserire nella home o

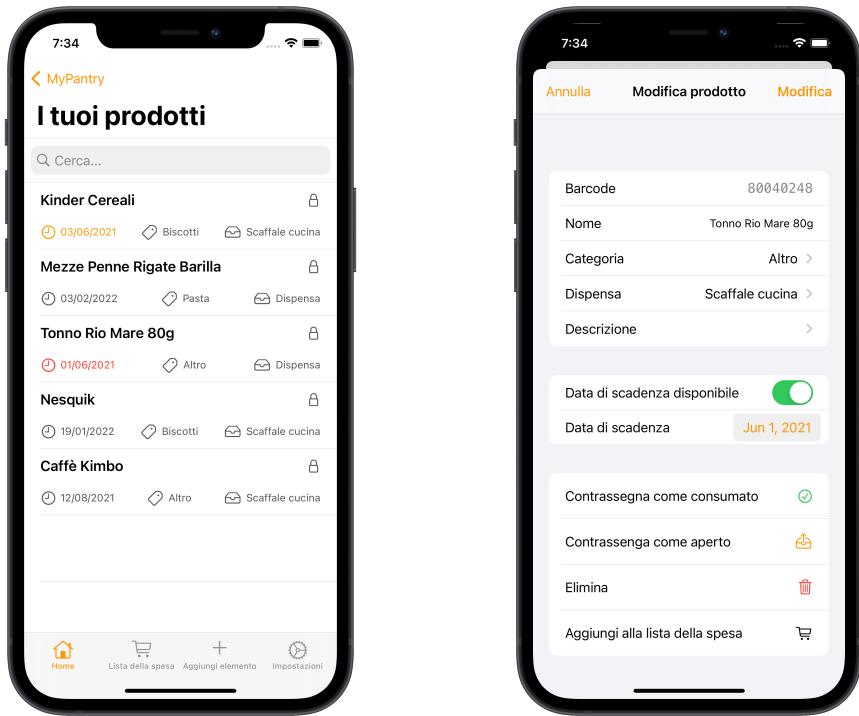


Figura 4: Gestione e modifica dei prodotti all'interno di *MyPantry*

nel centro notifiche del proprio dispositivo iOS per essere sempre aggiornato sulla situazione della dispensa, visualizzando il numero di prodotti presenti, il numero di prodotti in scadenza e il numero di quelli scaduti.

Il Widget realizzato è descritto nel capitolo 2.4. Infine, è stata realizzata una app watchOS che permette agli utenti proprietari di un *Apple Watch* di avere sempre a disposizione sul loro polso un riepilogo del contenuto della dispensa (vedi capitolo 2.5).

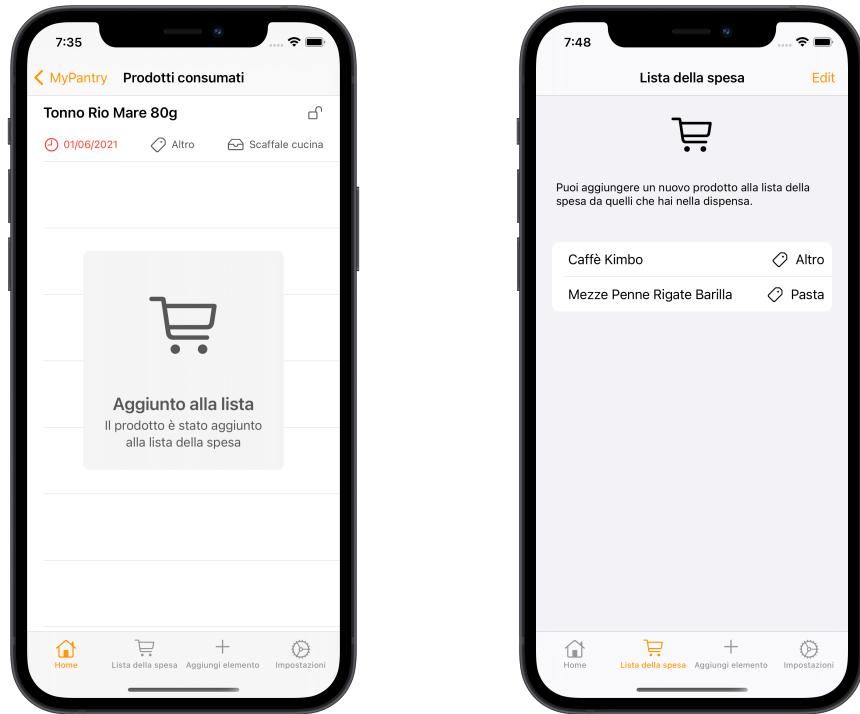


Figura 5: Lista della spesa

## 2 Strategie di implementazione

In questa sezione si analizzano le principali funzionalità di *MyPantry* e si discutono le strategie di implementazione adottate, sfruttando le librerie offerte da Swift 5.

Il progetto è stato realizzato seguendo il pattern *Model View Controller* e le [raccomandazioni fornite da Apple](#) per lo sviluppo di applicazioni in Swift. Inoltre, si è cercato di applicare il più possibile gli interessanti suggerimenti forniti dalle [Human Interface Guidelines](#) per sviluppare un'applicazione gradevole anche dal punto di vista della *user experience*.

### 2.1 Database

Come anticipato nel capitolo 1.1, l'applicazione sviluppata utilizza un database locale per memorizzare i prodotti inseriti dall'utente e interagisce con un database remoto collaborativo per associare le informazioni di un prodotto al *barcode* corrispondente.

#### 2.1.1 Connessione al database remoto

Per mantenere il codice del progetto ordinato e applicare i principi di ingegneria del software di *alta coesione e basso accoppiamento*, è stata creata una classe denominata `ServerModel` alla quale è stata assegnata la responsabilità di gestire le comunicazioni con il database remoto.

La comunicazione con il database remoto avviene sfruttando le `URLSession` del framework `foundation` di Swift.

Le risposte da parte del server condiviso sono in formato JSON. Sfruttando l'interfaccia `Codable` e `JSONDecoder`, è possibile convertire in modo rapido e efficiente la risposta del server nella `struct` mostrata nel listing 1.

```
1 struct Product : Codable {
2     let id : String
3     let name : String
4     let description : String
5     let barcode : String
6     let userId : String
7     let test : Bool
8     let createdAt : String
9     let updatedAt : String
10 }
```

Listing 1: Una `struct` per un prodotto disponibile sul server condiviso

Il listing 2, invece, mostra i metodi disponibili in `ServerModel`, che corrispondono alle funzionalità richieste dalle specifiche del progetto.

```
1 protocol ServerModelProtocol {
2     func login() throws -> Void
3     func searchProducts(withBarcode: Barcode) throws -> [Product]
4     func pushNewPantryItem(newProduct item: PantryItem) -> Void
5     func pushProductPreference(preference: Product) -> Void
6     func checkLoginDetails(email: String, password: String) -> Void
7     func register(email: String, password: String, username: String) -> Void
8 }
```

Listing 2: I metodi presenti in `ServerModel`

Se l'utente effettua la ricerca di un *barcode* ma il prodotto non è tra quelli restituiti dal database collaborativo, vi è la possibilità di aggiungere un nuovo prodotto. I dettagli di tale prodotto vengono caricati

sul database collaborativo per essere disponibili agli altri utenti. Se invece il prodotto cercato è tra quelli mostrati nell'elenco scaricato dal server, il prodotto scelto viene comunicato ad esso tramite il metodo `pushProductPreference`.

Al primo avvio dell'applicazione viene richiesto all'utente di registrarsi sul database collaborativo o di utilizzare delle credenziali esistenti. Le credenziali di accesso possono essere modificate in *impostazioni*.

**Nota** Qualora non sia disponibile una connessione a internet, si è deciso di non consentire all'utente di procedere con l'aggiunta di nuovi prodotti su *MyPantry* per non derogare alle specifiche del progetto che prevedono l'obbligatorietà dell'utilizzo del database collaborativo.

### 2.1.2 Database locale

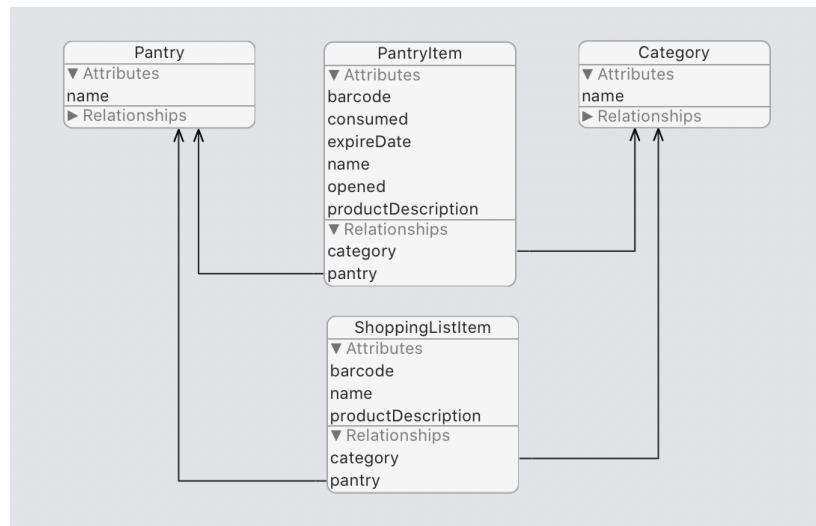


Figura 6: CoreData model

I prodotti disponibili nella dispensa dell'utente vengono memorizzati in un database locale realizzato utilizzando **CoreData**.

Il modello del database è mostrato nella figura 6 e prevede quattro entità:

- **Pantry**: rappresenta un'area della casa definita dall'utente nella quale vengono riposti i prodotti acquistati (può ad esempio essere il frigo, il freezer o un determinato scaffale)
- **Category**: rappresenta una categoria di prodotti definita dall'utente (può ad esempio essere pasta, snack, dolci,...)
- **PantryItem**: rappresenta un prodotto presente nella dispensa. Oltre al nome e al *barcode*, si tiene traccia della descrizione, della data di scadenza e dell'eventuale apertura del prodotto. Quando un prodotto viene consumato finisce nell'archivio della app e il valore `consumed` viene impostato a `true`
- **ShoppingListItem**: rappresenta un prodotto nella lista della spesa, del quale si memorizza il *barcode*, il nome, la descrizione, la categoria e la dispensa nella quale viene abitualmente inserito

L'inserimento, la modifica e la rimozione di entità non è mai affidata ai `ViewController` ma è una responsabilità assegnata a `MyPantryModel`.

L'eliminazione di una **Category** o di un **Pantry** segue la politica *cascade*. Di conseguenza, se l'utente decide di rimuovere una categoria o una dispensa da *MyPantry*, vengono eliminati anche tutti i prodotti associati ad essa.

Per memorizzare le preferenze dell'utente, come ad esempio la volontà di ricevere notifiche, sono state utilizzate le `UserDefault`s, come raccomandato da Apple.

## 2.2 Scansione del *barcode*

Per semplificare e agevolare l'inserimento di nuovi prodotti su *MyPantry*, è stata prevista la possibilità di scansionare il *barcode*. Per implementare questa funzionalità, è stato utilizzata la potente libreria [AVFoundation](#) di Swift. I formati di barcode riconosciuti da *MyPantry* sono quelli che rispettano gli standard EAN-8, EAN-13, PDF417<sup>1</sup>.

È comunque lasciata all'utente la possibilità di inserire il *barcode* manualmente.

## 2.3 Notifiche

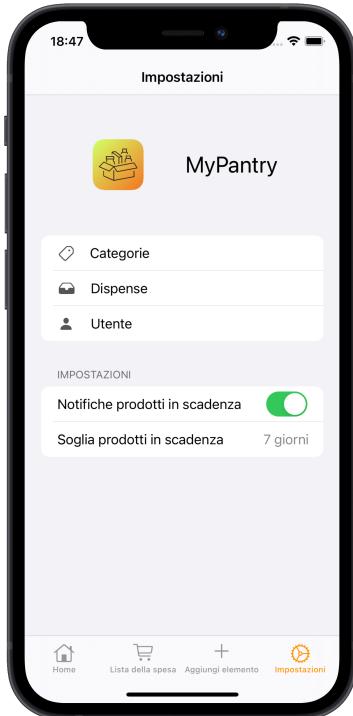


Figura 7: Pagina delle impostazioni di *MyPantry*

Dalla pagina delle *Impostazioni*, mostrata in figura 7, è possibile impostare il numero di giorni antecedenti la data di scadenza dei prodotti entro i quali un prodotto viene considerato *in scadenza*.

Se l'utente ha abilitato le notifiche per *MyPantry*, al momento dell'inserimento di un nuovo prodotto viene programmata una notifica per il giorno in cui il prodotto entrerà nello stato *in scadenza* e un'altra notifica per il giorno in cui il prodotto scadrà.

Qualora il prodotto venga consumato o eliminato prima della data di scadenza, le notifiche programmate vengono eliminate.

La programmazione delle notifiche viene effettuata dal `MyPantryModel` al momento dell'inserimento di un prodotto, sfruttando il framework di iOS `NotificationCenter`.

<sup>1</sup>In Europa gli standard adottati per i *barcode* dei prodotti alimentari sono EAN-8 e EAN-13

Nel caso in cui l'utente disabilitasse le notifiche dalle impostazioni di *MyPantry*, tutte le notifiche precedentemente programmate verrebbero eliminate.

## 2.4 Widget



Figura 8: Widget nella *home* di iOS

È stato realizzato un Widget per *MyPantry* che mostra il numero di prodotti presenti in dispensa, il numero di prodotti in scadenza e il numero di prodotti scaduti. In questo modo l'utente è sempre informato sullo stato della dispensa. Uno *screenshot* del Widget è mostrato in figura 8.

Per condividere informazioni tra il Widget, *MyPantry* e l'applicazione per Apple Watch descritta nel prossimo paragrafo, è stato creato un **AppGroup**, ovvero un identificativo che permette di raggruppare più applicazioni e consentire lo scambio di informazioni tra esse.

In particolare, lo scambio di informazioni tra *MyPantry* e il Widget avviene sfruttando le **UserDefault**s, già utilizzate per memorizzare le preferenze dell'utente. Ogni volta che viene effettuata una modifica a qualche prodotto in *MyPantry*, viene aggiornato il numero di prodotti presenti in dispensa, il numero di prodotti in scadenza e il numero di quelli scaduti all'interno delle **UserDefault**s. Il Widget accede a queste informazioni e le mostra all'utente nella home di iOS o nel centro notifiche.

Il Widget, a differenza di *MyPantry* che è una *storyboard-application*, è stato realizzato utilizzando interamente SwiftUI.

## 2.5 Apple Watch

È stata realizzata una app per Apple Watch che permette di mostrare le stesse informazioni mostrate dal Widget, descritto nel capitolo precedente.

Il sistema di scambio delle informazioni tra *MyPantry* e watchOS, tuttavia, non è lo stesso utilizzato con il Widget. A partire da watchOS 2, infatti, Apple ha rimosso la possibilità di creare `UserDefault`s o `CoreData` condivisi tra iOS e watchOS, per permettere agli Apple Watch di essere più autonomi da iOS e consentire ai loro utilizzatori di sfruttarne funzionalità anche quando non hanno a disposizione il proprio iPhone.

Di conseguenza, il sistema di comunicazione è più complesso e prevede lo scambio di messaggi in modo asincrono tra iOS e Apple Watch.

L'applicazione *MyPantry* per iOS, quando viene avviata o quando registra una modifica nel database dei prodotti, invia all'applicazione per Apple Watch un messaggio contenente il numero aggiornato di prodotti in dispensa, in scadenza e scaduti.

Il model `WatchConnectionModel` ha la responsabilità di stabilire la connessione con Apple Watch ed è utilizzato dal `MyPantryModel` per inviare i messaggi.

L'applicazione per Apple Watch, invece, rimane in ascolto di messaggi di questo tipo da parte dell'app per iOS. Ogni volta che riceve un messaggio con le informazioni aggiornate, memorizza il contenuto nelle proprie `UserDefault`s, in modo che i dati siano accessibili in futuro anche nel caso in cui iPhone non fosse raggiungibile.

Lo scambio di messaggi avviene utilizzando il framework `WatchConnectivity`.



Figura 9: MyPantry su Apple Watch

## 2.6 Documentazione

Il codice Swift realizzato è stato commentato (in inglese) seguendo le [linee guida](#) fornite da Apple per la realizzazione di documentazioni di progetti Xcode.

Utilizzando `swift-doc` sono state generate le pagine HTML della documentazione, che sono disponibili nella cartella `docs`.

# MyPantry Documentation

## Classes

**C** [AppDelegate](#)

**C** [MyPantryModel](#)

Handles all the main activities of MyPantry and the connection with the CoreData database

**C** [ServerModel](#)

Handles the connection between MyPantry and the collaborative database

**C** [WatchConnectionModel](#)

Handles the connection with the Apple Watch app. Everytime there is an update in the number of products in the database, `MyPantryModel` notifies the Apple Watch using the `WatchConnectivityModel`.

## ON THIS PAGE

[Classes](#)

[Structures](#)

[Enumerations](#)

[Protocols](#)

Figura 10: Un frammento della documentazione del progetto generata con `swift-doc`