UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica e dell'Automazione

TESI DI LAUREA

Progettazione e sviluppo di un'applicazione iOS per il monitoraggio e la gestione del fabbisogno calorico quotidiano tramite Core ML

Design and development of an iOS application for monitoring and managing daily caloric intake using Core ML

Relatore

Dott. Enrico Corradini

Candidato

Alessandro Pettinaro

ANNO ACCADEMICO 2023-2024

"A due persone che, con la loro semplicità e il loro silenzio, sono state il pilastro della mia vita. Sempre sorridenti e instancabili lavoratori, non chiedevano nulla e non esprimevano facilmente le emozioni, ma il loro affetto era lì, sincero e profondo, anche senza parole. Non hanno avuto la possibilità di accompagnarmi fino a questo traguardo, ma so che nel loro modo , mi volevano un mondo di bene. Questa tesi è dedicata a loro, con la speranza che possano essere fieri di me, ovunque siano. G.S e V.P."

Sommario

L'obesità e il sovrappeso sono problematiche sempre più diffuse nella società contemporanea, con ripercussioni sulla salute e sulla qualità della vita. La gestione di queste condizioni è spesso complicata dalla carenza di strumenti pratici e affidabili che permettano un monitoraggio efficace e consapevole dell'alimentazione e dell'attività fisica. Per questo motivo, è fondamentale sviluppare una soluzione che supporti le persone nel migliorare il proprio stile di vita, facilitando il raggiungimento di un equilibrio tra salute e benessere. Questa tesi si focalizza sulla progettazione e realizzazione di un'applicazione iOS, HealthyLife, ideata per facilitare e ottimizzare il monitoraggio delle abitudini alimentari. Questo è stato possibile grazie all'integrazione di tecnologie avanzate, come HealthKit, un framework che permette di accedere e utilizzare i dati sanitari registrati nell'app Salute. Grazie a HealthKit, l'applicazione è in grado di raccogliere informazioni rilevanti, come attività fisica, peso corporeo e altri parametri biometrici, che vengono poi elaborati per calcolare con precisione il fabbisogno calorico giornaliero dell'utente. HelthyLife permette inoltre di monitorare facilmente le calorie assunte e semplifica l'aggiunta di alimenti consumati grazie a un modello di Machine Learning sviluppato con Create ML, che riconosce automaticamente gli alimenti tramite la fotocamera del dispositivo. In questo modo il monitoraggio dell'alimentazione diventa efficace, rapido e intuitivo.

Indice

Introduzione **1**

1 Descrizione e Analisi **3**
1.1 Problema da risolvere . . . 3
1.2 Descrizione . . . 4
1.3 Analisi dei requisiti . . . 6
1.3.1 Requisiti funzionali . . . 7
1.3.2 Requisiti non funzionali . . . 8
1.4 Diagrammi casi d'uso . . . 9
1.4.1 Gestione degli utenti . . . 9
1.4.2 Gestione pasti . . . 12
1.5 Diagramma delle classi . . . 14

2 Progettazione **18**
2.1 Architettura dell'applicazione . . . 18
2.2 Classi di progettazione . . . 19
2.3 Mockup . . . 21
2.4 Soluzione per l'identificazione degli alimenti . . . 23
2.4.1 Identificazione tramite Machine Learning . . . 23
2.4.2 Identificazione tramite codice a barre . . . 26
2.5 Soluzione per il Recupero dei Dati Nutrizionali . . . 26
2.5.1 USDA . . . 26
2.5.2 Open food facts . . . 26
2.6 Soluzione fabbisogno giornaliero . . . 26

3 Implementazione **28**
3.1 Tecnologie utilizzate . . . 28
3.1.1 Swift-UI . . . 28
3.1.2 HealtKit . . . 29
3.1.3 Firebase . . . 29
3.1.4 Core ML . . . 30
3.2 Implementazione classe HealtKit . . . 31
3.2.1 Implementazione della funzione Steptoday . . . 31
3.2.2 Implementazione della funzione Height . . . 32
3.3 Implementazione della classe FoodBarcodeAPI . . . 32

3.4 Implementazione di HomeView 33
3.5 Addestramento e integrazione del modello Core ML 34
3.5.1 Implementzione della funzione classify 35

4 Test e User Experience **36**
4.1 Test 36
4.2 User experience 37
4.2.1 Auth 38
4.2.2 Fase di registrazione 39
4.2.3 Home 40
4.2.4 Food 41

5 Conclusione **43**
5.1 Discussione 43
5.2 Sviluppi futuri 43

Bibliografia **45**

Ringraziamenti **46**

Elenco delle figure

1.1 Suddivisione dei requisiti . . . 6
1.2 Suddivisione dei requisiti funzionali . . . 7
1.3 Suddivisione dei requisiti non funzionali . . . 9
1.4 Attori . . . 9
1.5 Diagramma dei casi d'uso: gestione utente . . . 10
1.6 Diagramma dei casi d'uso : Gestione pasti . . . 12
1.7 Diagramma delle classi analisi del package "Model" . . . 15
1.8 Diagramma delle classi analisi del package "Repository" . . . 16
1.9 Diagramma delle classi analisi del package " Food Recognition e API Integration " . . . 16

2.1 Schema rappresentativo dell'architettura Model-View-ViewModel . . . 19
2.2 Diagramma delle classi di progettazione del package "Repository" . . . 19
2.3 Diagramma delle classi di progettazione del package "Model" . . . 20
2.4 Diagramma delle classi di progettazione del package "API" . . . 21
2.5 Mockup del login e del sign-up . . . 21
2.6 Mockup della fase di registarzione dell'utente . . . 22
2.7 Mockup della home e della sezione riguardante l'account utente . . . 22
2.8 Mockup delle schermate inerenti al DiaryFood . . . 23
2.9 Machine Learning . . . 24
2.10 Addestarmento modello . . . 25
2.11 Processo di identificazione degli alimenti tramite machine learning . . . 25
2.12 Logo USDA . . . 26
2.13 Logo Open food facts . . . 27

3.1 Logo di SwiftUI . . . 28
3.2 Logo HealthKit . . . 29
3.3 Logo Firebase . . . 29
3.4 Schema di funzionamento Firestore . . . 30
3.5 Flusso di lavoro di Core ML . . . 30
3.6 Implementazione della funzione Steptoday . . . 31
3.7 Implementazione della funzione Height . . . 32
3.8 Implementazione della classe FoodBarcodeAPI . . . 32
3.9 Implementazione di HomeView . . . 33
3.10 Implementazione della classe FoodClassifier . . . 34

4.1 Test del riperimento del valore dell'altezza da Salute 37
4.2 Test del metodo di calcolo del fabbisogno giornaliero 37
4.3 Interfacce di autenticazione 38
4.4 Interfacce per la registarzione 39
4.5 Viste relative alla sezione "Home" 40
4.6 Viste relative alla gestione del cibo nella sezione "Food" 41

Elenco delle tabelle

1.1 Descrizione caso d'uso Registrazione 11
1.2 Descrizione caso d'uso Inserimento Alimento CB 13
1.3 Descrizione caso d'uso Visualizzazione Dashboard 14
1.4 Descrizione delle classi del package "Model" 15
1.5 Descrizione delle classi del package "Food Recognition e API Integration " . . 17

2.1 Descrizione delle classi di progettazione aggiuntive del package "Repository" 20

Introduzione

L'organizzazione mondiale della sanità ha definito il concetto di salute come "una condizione di completo benessere fisico, mentale e sociale e non esclusivamente l'assenza di malattia o infermità". La salute fisica è quindi un aspetto centrale per il benessere di ogni individuo, influenzando direttamente la qualità della vita e la prevenzione di numerose patologie croniche. Adottare uno stile di vita sano, caratterizzato da un'alimentazione equilibrata e da un'attività fisica regolare, aiuta a garantire il corretto funzionamento dell'organismo e migliorare la resilienza fisica e mentale. Una dieta bilanciata fornisce i nutrienti indispensabili per sostenere i processi vitali, mantenere un peso corporeo adeguato e rafforzare il sistema immunitario. L'attività fisica, invece, contribuisce alla salute cardiovascolare, migliora la forza muscolare e apporta benefici tangibili al benessere psicologico, riducendo lo stress e favorendo un miglior equilibrio emotivo. Nonostante si conoscano questi benefici, il sovrappeso e l'obesità sono diventati problematiche sempre più comuni nella società contemporanea, con gravi conseguenze sulla salute della colletività. Secondo i dati dell'Organizzazione Mondiale della Sanità, queste condizioni sono strettamente legate all'insorgenza di malattie come il diabete di tipo 2, le patologie cardiovascolari e alcuni tipi di tumore. Per affrontare queste problematiche e offrire un concreto supporto agli utenti, si è deciso di progettare e sviluppare un'applicazione iOS, denominata HelthyLife. L'obiettivo principale dell'applicazione è semplificare il monitoraggio dell'alimentazione e dell'attività fisica, rispondendo alle difficoltà più comuni legate alla gestione del proprio benessere. Grazie all'integrazione di tecnologie avanzate, come il framework HealthKit e un modello di Machine Learning, HelthyLife mira a semplificare e rendere più intuitivo il percorso verso uno stile di vita sano, fornendo agli utenti uno strumento pratico per comprendere, analizzare e migliorare le proprie abitudini quotidiane. Questa tesi nasce dall'esperienza maturata durante un tirocinio presso il Dipartimento di Ingegneria dell'Informazione (DII) dell'Università Politecnica delle Marche. La tesi è strutturata in cinque capitoli, ognuno dei quali affronta un aspetto specifico dello sviluppo dell'applicazione. Nel Capitolo 1 viene analizzato il procedimento per reperire le informazioni necessarie al calcolo del fabbisogno calorico giornaliero per uomini e donne di diverse fasce di età. Inoltre, viene fornita una panoramica delle principali funzionalità dell'applicazione e un'analisi dettagliata dei requisiti funzionali e non funzionali utili alla progettazione. Il Capitolo 2 approfondisce le scelte progettuali e le soluzioni adottate per sviluppare un sistema modulare, manutenibile e scalabile, descrivendo nel dettaglio l'architettura e il processo di progettazione. Nel Capitolo 3 si entra nel dettaglio delle tecnologie utilizzate, includendo esempi di codice. Il Capitolo 4 è dedicato alla qualità dell'applicazione, analizzando le strategie di testing adottate e l'esperienza utente, con l'obiettivo di garantire un sistema affidabile e user-friendly. Infine, il Capitolo 5 si conclude con una riflessione sui

risultati raggiunti e un'analisi delle prospettive di sviluppo futuro di HealthyLife, come l'introduzione di nuove funzionalità per rendere l'applicazione ancora più completa ed efficace.

CAPITOLO 1

Descrizione e Analisi

Questo progetto parte dall'analisi del problema del sovrappeso e dell'obesità nella società moderna mettendo alla luce le difficoltà che molte persone incontrano nel migliorare il proprio stile di vita, specialmente nel monitorare l'apporto calorico giornaliero. In questo capitolo viene quindi proposta una soluzione innovativa al problema attraverso la progettazione di un'applicazione iOS che sfrutta il framwork Healtkit per monitorare e migliorare lo stato di salute dell'utente. L'applicazione semplifica il monitoraggio dell'alimentazione grazie all'identificazione degli alimenti tramite barcode e identificazione tramite machine learning.

1.1 Problema da risolvere

Nella società moderna il peso corporeo rappresenta un aspetto sempre piu rilevante, non solo per un fatto di estetica, ma sopratutto per la salute dell'individuo. Studi effettuati dall'OMS (Organizzazione mondiale della società) [1] affermano che il 59% degli adulti europei e 1 bambino su 3 è in sovrappeso o è affetto da obesità. Sovrappeso e obesità sono tra le principali cause di morte e disabilità nell'Europa: stime recenti suggeriscono che causano più di 1,2 milioni di decessi all'anno, corrispondenti a oltre il 13% della mortalità totale nel contenente europeo. Una delle maggiori difficoltà che incontrano gli individui quando tentano di migliorare il proprio stile di vita è la disinformazione: è sopratutto sui social che le persone ottengono informazioni su metodi e strategie per raggiungere obiettivi di salute. Ma sui social capita sempre più spesso di trovare informazioni provenienti da fonti non ufficiali o non autorevoli. Un'altra diffocoltà che l'individuo incontra, inoltre, è monitorare l'apporto calorico di ogni pasto, poiché questa operazione richiede molto tempo. Infatti, per ogni alimento assunto, è necessario cercare le informazioni nutrizionali su internet e successivamente calcolare i macronutrienti (ovvero principi alimentari che devono essere introdotti in grandi quantità, poichè rappresentano la più importante fonte energetica dell'organismo, come ad esempio corboedrati, grassi e proteine). Tutto questo processo rende molto complicato per una persona tener traccia degli alimenti consumati e quindi avere un quadro generale dei valori nutrizionali assunti in una giornata. C'è bisogno, quindi, di uno strumento che semplifichi questo processo, permettendo in modo facile e intuitivo all'individuo di:

- comprendere se è in buona salute o meno;
- registrare gli alimenti assunti durante la giornata e informare l'utente nel caso in cui un determinato alimento possa risultare dannoso per la sua salute;

[1] https://www.epicentro.iss.it/obesita/report-obesita-oms-2022

- tener traccia dei macronutrienti assunti durante la giornata;
- monitorare in modo automatico l'attività fisica e le calorie consumate tramite il framework HealthKit (framework messo a disposizione da Apple che ci consente di accedere ai dati memorizzati in Salute [2]).

L'applicazione sviluppata risolve queste problematiche fornendo all'utente un sistema efficace, in grado di elaborare un percorso calorico personalizzato basato sui dati e sugli obiettivi dell'utente.

1.2 Descrizione

Il progetto proposto consiste nella realizzazione di un'applicazione iOS che si occupa del monitoraggio e del miglioramento dello stile di vita di un'individuo. L'applicazione progettata infatti, grazie al reperimento dei dati salutari dell'utente (tramite il framework HealthKit), è in grado di analizzare lo stato di salute e creare un percorso calorico personalizzato. Per valutare lo stato di salute di una persona si utilizza l'indice di massa corporea (BMI), un parametro che misura la condizione fisica basandosi sul rapporto tra il peso e l'altezza. Il BMI si calcola con la seguente formula:

$$\textbf{BMI} = \frac{\text{weight}}{\text{height}^2}$$

Il BMI[3] è una stima approssimativa della percentuale di grasso corporeo ed è comunemente utilizzato per classificare gli individui in diverse categorie in base alla loro condizione fisica:

1. **Sottopeso**: BMI inferiore a 18,5;
2. **Normopeso**: BMI compreso tra 18,5 e 24,9;
3. **Sovrappeso**: BMI compreso tra 25 e 29,9;
4. **Obesità di Classe 1** (Obesità lieve): BMI compreso tra 30 e 34,9;
5. **Obesità di Classe 2** (Obesità moderata): BMI compreso tra 35 e 39,9.

Una volta calcolato il BMI, l'applicazione comunica all'utente la categoria di appartenenza e stima il suo range di peso ideale. Successivamente viene determinato il fabbisogno calorico, partendo principalmente da due fattori denominati il *BMR* e il *TDEE*.

Il **BMR** (Metabolismo basale)[4] è un valore che indica il dispendio energetico di un individuo a riposo. Questo comprende l'energia necessaria per le funzioni metaboliche vitali (respirazione, circolazione sanguigna, digestione, attività del sistema nervoso, ecc.). Il calcolo del BMR varia in base a genere, altezza, peso e età. Le formule per calcolarlo sono le seguenti:

1. Nel caso in cui l'individuo è di **sesso maschile**:

$$\textbf{BMR} = 66.47 + (13.75 \times \text{weight}) + (5.00 \times (\text{height} \times 100)) - (6.75 \times \text{age})$$

2. Nel caso in cui l'individuo è di **sesso femminile**:

$$\textbf{BMR} = 655.09 + (9.56 \times \text{weight}) + (1.84 \times (\text{height} \times 100)) - (4.67 \times \text{age})$$

weight è il peso in chilogrammi, *height* è l'altezza in metri e *age* è l'età in anni.

[2] L'applicazione Salute è un'app nativa su dispositivi Apple, che consente agli utenti di monitorare e gestire vari aspetti della loro salute e benessere. La sua funzione principale è quella di raccogliere, visualizzare e analizzare i dati relativi alla salute e all'attività fisica dell'utente.

[3] https://www.my-personaltrainer.it/bmi.htm

[4] https://www.myprotein.it/thezone/alimentazione/metabolismo-basale-cos-e-come-si-calcola/

Il **TDEE** (Total Daily Energy Expenditure)[5]rappresenta la quantità di calorie bruciate dal corpo umano, considerando il livello di attività fisica dell'utente. Per calcolare il TDEE si utilizza la seguente formula:

$$\textbf{TDEE} = \text{BMR} \times \text{fattore di attività}$$

Il fattore di attività è un valore che rappresenta il livello di attività fisica. Di seguito vengono elencati i diversi fattori di attività:

1. **Sedentaria**: pochissima o nessuna attività fisica, con un fattore di attività pari **1.2**;
2. **Leggermente attivo**: leggera attività fisica 1 a 3 volte, con un fattore di attività pari **1.37**;
3. **Moderatamente attivo**: moderata attività fisica 3 a 5 volte a settimana , con un fattore di attività pari **1.55**;
4. **Molto attivo**: attività fisica intensa quasi ogni giorno, con un fattore di attività pari **1.725**;
5. **Estremamente attivo**: attività fisica 2 volte al giorno, con un fattore di attività pari **1.9**;

Oltre a considerare i fattori appena descritti l'applicazione, da l'opportunità all'utente di inserire il suo obiettivo di peso. Una volta indicato quest'ultimo dato, viene calcolato il fabbisogno calorico giornaliero e l'applicazione mette a disposizione 2 tecniche di dimagrimento denominate:

- **Normal**, che prevede un deficit calorico pari a *500 kcal*;
- **Speed**, che prevede un deficit calorico pari a *800 kcal*.

Per ogni tipo di tecnica vengono anche riportati i giorni approssimativi per il raggiungimento dell'obiettivo di peso prefissato dall'utente.

Oltre alle funzionalità appena descritte l'applicazione presenta altre servizi che riguardano principalmente la gestione e il tracciamento dei pasti e delle calorie:

1. *Individuare le chilocalorie massime da assumenre ad ogni pasto*;
2. *Individuare i macronutrienti per ogni giornata*;
3. *Individuare il valori nutrizionale delle pietanze*;
4. *Verificare la presenza di eventuali allergeni*;
5. *Registrare giornalmente le chilocalorie assunte*;
6. *Calcolare i passi effettuati in ogni giornata e le chilocalorie assunte*.

Grazie al fabbisogno giornaliero calcolato precedentemente, l'app è in grado di determinare le chilocalorie che l'utente deve assumere nei singoli pasti e i macronutrienti che deve assumere durante la giornata. Per fare questo rendendo l'uso dell'applicazione facile e intuitivo, si è deciso di implementare due modalità di identificazione dei valori nutrizionali degli alimenti, uno per gli alimenti confezionati e uno per gli alimenti non confezionati:

- per conoscere i volori nutrizionali degli ***alimenti confezionati***è possibile utilizzare il codice a barre presente in ogni involucro.

[5]https://www.my-personaltrainer.it/nutrizione/tdee-o-dispendio-calorico-cosa-e-da-cosa-e-composto.html

- per gli ***alimenti non confezionati***, l'applicazione offre la possibilità di utilizzare un sistema di riconoscimento del cibo basato su un modello di machine learning sviluppato con Create ML. Questo modello consente di identificare in modo rapido gli alimenti attraverso l'uso della fotocamera anteriore e di conseguenza il loro valore nutrizionale.

L'applicazione offre anche una gestione avanzata delle allergie alimentari. Durante la fase di registrazione, l'utente ha la possibilità di specificare le eventuali allergie da cui è affetto. Successivamente, quando un alimento viene scansionato, l'app confronta automaticamente le informazioni raccolte con la lista delle allergie registrate dall'utente, verificando la presenza di potenziali allergeni. In caso di rilevamento, l'app provvede a notificare immediatamente l'utente, avvisandolo della possibile presenza di ingredienti che potrebbero causare reazioni allergiche.

Dopo l'identificazione delle caratteristiche degli alimenti, l'utente immetterà la quantità assunta di ogni pietanza e questa verrà memorizzata. Grazie a una sezione dell'applicazione chiamata Food, potrà gestire facilmente i pasti giornalieri e per ogni pasto consumato, avere un riepilogo delle calorie assunte. Inoltre, grazie l'utilizzo di un "date picker" è possibile selezionare una data specifica per visualizzare o inserire i pasti registrati in quel giorno. L'app include anche una dashboard che offre una panoramica dettagliata delle calorie e dei macronutrienti assunti durante la giornata. Oltre a questi dati, la dashboard visualizza anche il numero di passi effettuati e le calorie bruciate, fornendo un quadro completo dell'attività fisica dell'utente.

1.3 Analisi dei requisiti

Mediante l'analisi approfondita della descrizione dall'applicazione, sono stati identificati una serie di requisiti chiave, che hanno permesso di derivare una serie di casi d'uso volti a soddisfarli. Un caso d'uso rappresenta un'azione o un insieme di azioni che un attore, ovvero un utente o un sistema esterno, può compiere all'interno dell'applicazione. I requisiti del sistema sono stati suddivisi in requisiti funzionali e requisiti non funzionali (come mostrato in Figura 1.1). I requisiti funzionali rappresentano quelle necessità relative a concrete funzionalità richieste per il software. I requisiti non funzionali, invece, rappresentano vincoli particolari e non propriamente concreti su alcuni funzionamenti del software.

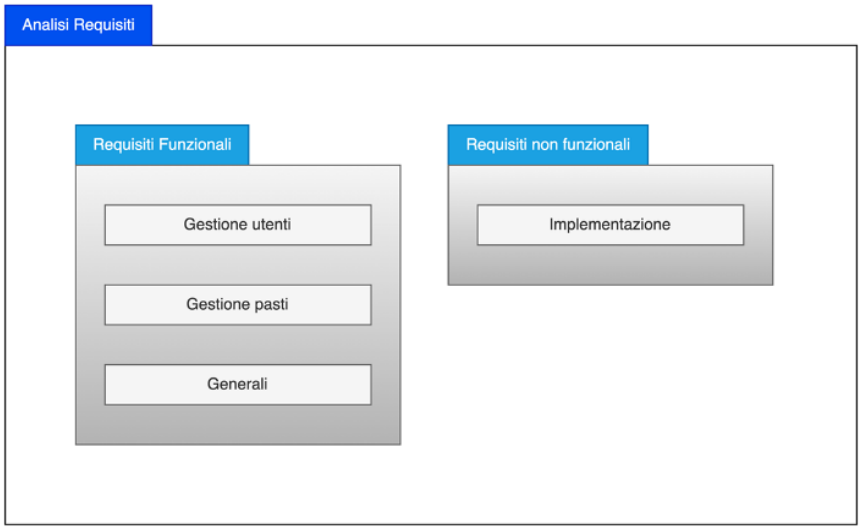

Figura 1.1: Suddivisione dei requisiti

1.3.1 Requisiti funzionali

Sono organizzati in 3 macrocategorie:

- **Gestione utenti**;
- **Gestione pasti**;
- **Generali**.

Ogni macrocategoria è rappresentata da un package che contiene i corrispondenti requisiti funzionali, come mostrato in Figura1.2

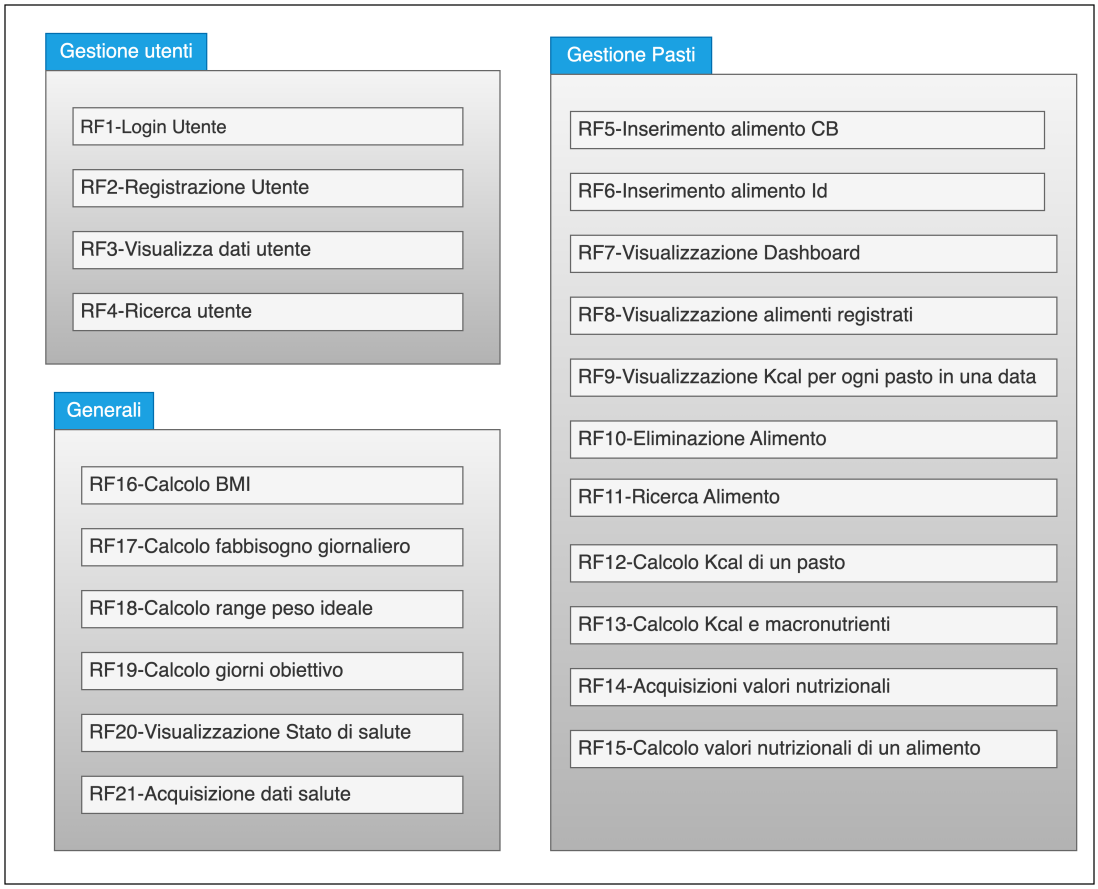

Figura 1.2: Suddivisione dei requisiti funzionali

Segue una descrizione di ogni requisito funzionale del sistema:

- RF1-**Login Utente**: il sistema si occuperà di consentire l'accesso all'utente all'interno della piattaforma.
- RF2-**Registrazione Utente**: il sistema consentirà la registrazione all'utente.
- RF3-**Visualizza dati utente:** il sistema permetterà la visualizzazione dei propri dati all'utente.
- RF4-**Ricerca utente**: il sistema si occuperà di ricercare l'utente.
- RF5-**Inserimento alimento CB**: il sistema permetterà all'utente di registrare un alimento tramite il codice a barre.

- RF6-**Inserimento alimento Id**: il sistema permetterà all'utente di registrare un alimento tramite identificazione dell'alimento.
- RF7-**Visualizzazione Dashboard**: il sistema permetterà la visualizzazione di una schermata riassuntiva della giornata dove al suo interno sono riportate tutte le kcal e i macronutrienti assunti durante la giornata.
- RF8-**Visualizzazione alimenti registrati**: il sistema permetterà la visualizzazione di alimenti registrati in un pasto per una determinata data.
- RF9-**Visualizzazione Kcal per ogni pasto in una data**: il sistema permetterà la visualizzazione delle kcal assunte dall'utente in una data suddivise nei pasti.
- RF10-**Eliminazione Alimento**: il sistema permetterà l'eliminazione di un alimento registrato.
- RF11-**Ricerca Alimento**: il sistema permetterà la ricerca di alimenti assunti dall'utente.
- RF12**Calcolo Kcal di un pasto**: il sistema si occuperà del calcolo di kcal di ogni pasti.
- RF13-**Calcolo Kcal e macronutrienti**: il sistema si occuperà del calcolo delle kcal e dei macronutrienti di una determinata data .
- RF14-**Acquisizioni valori nutrizionali**: il sistema sarà in grado di acquisire i valori nutrizionali di un alimento tramite un'API esterna (Application Protocol Intarface).
- RF15-**Calcolo valori nutrizionali di un alimento**: il sistema si occuperà del calcolo delle kcal e dei macronutrienti di un alimento assunto da un utente.
- RF16-**Calcolo BMI**: il sistema si occuperà del calcolo del BMI dell'utente.
- RF17-**Calcolo fabbisogno giornaliero**: il sistema si occuperà del calcolo del fabbisogno giornaliero in base a un obiettivo di peso dell'utente.
- RF18-**Calcolo range peso ideale**: il sistema si occuperà di calcolare il range di peso ideale dell'utente.
- RF19-**Calcolo giorni obiettivo**: il sistema si occuperà del calcolo dei giorni di dieta per il raggiungimento dell'obiettivo di peso dell'utente.
- RF20-**Visualizzazione Stato di salute**: il sistema si occuperà della visualizzazione dello stato di salute dell'utente.
- RF21-**Acquisizione dati salute**: il sistema si occuperà dell'acquisizione dei dati dell'utente grazie a HealthKit.

1.3.2 Requisiti non funzionali

I requisiti non funzionali sono stati raggrupati in una singola categoria di implementazione,come mostrato in Figura1.3 Di seguito è presente una descrizione di ogni requisito non funzionale del sistema.

- RNF1-**Implementazione Swift**: il sistema sarà implementato in Swift.
- RNF2-**Interfaccia grafica**: il sistema avrà un'interfaccia grafica realizzata con Swift UI.
- RNF3-**Persistenza dati online**: tutti i dati saranno memorizzati su Firebase.

Figura 1.3: Suddivisione dei requisiti non funzionali

1.4 Diagrammi casi d'uso

A seguito dell'analisi dei requisiti, sono stati sviluppati i diagrammi dei casi d'uso in modo da rappresentare le principali interazioni tra gli attori e il sistema. Il diagramma dei casi d'uso rappresenta un diagramma UML (Unified Modeling Language), utile per lo più per visualizzare i diversi tipi di ruoli in un sistema e come tali ruoli interagiscono con il sistema stesso. I casi d'uso del sistema sono stati divisi in 2 macrocategorie, ognuna concretizzata in un diagramma dei casi d'uso. Queste categorie rappresentano le due aree funzionali chiave del sistema: la **"gestione utenti"** e la **"gestione pasti"**. Gli attori che interagiscono con il sistema sono quelli riportati in Figura 1.4.

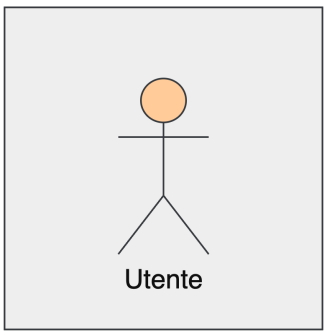

Figura 1.4: Attori

1.4.1 Gestione degli utenti

La macrocategoria "**gestione utenti**" racchiude tutte le funzionalità relative alla creazione e alla gestione dei profili utente all'interno del sistema.

Il diagramma dei casi d'uso in Figura 1.5 descrive le azioni che l'utente, ossia l'attore principale, può compiere per gestire il proprio profilo all'interno del sistema. In particolare, l'utente può interagire con l'app attraverso diversi casi d'uso, che possono essere:

1. *Login*
2. *Registrazione*
3. *Visualizza dati*

4. *Acquisizione dati salute*
5. *Calcolo BMI*
6. *Calcolo fabbisogno giornaliero*
7. *Calcolo giorni obiettivo*
8. *Calcolo range peso ideale*
9. *Ricerca utente*

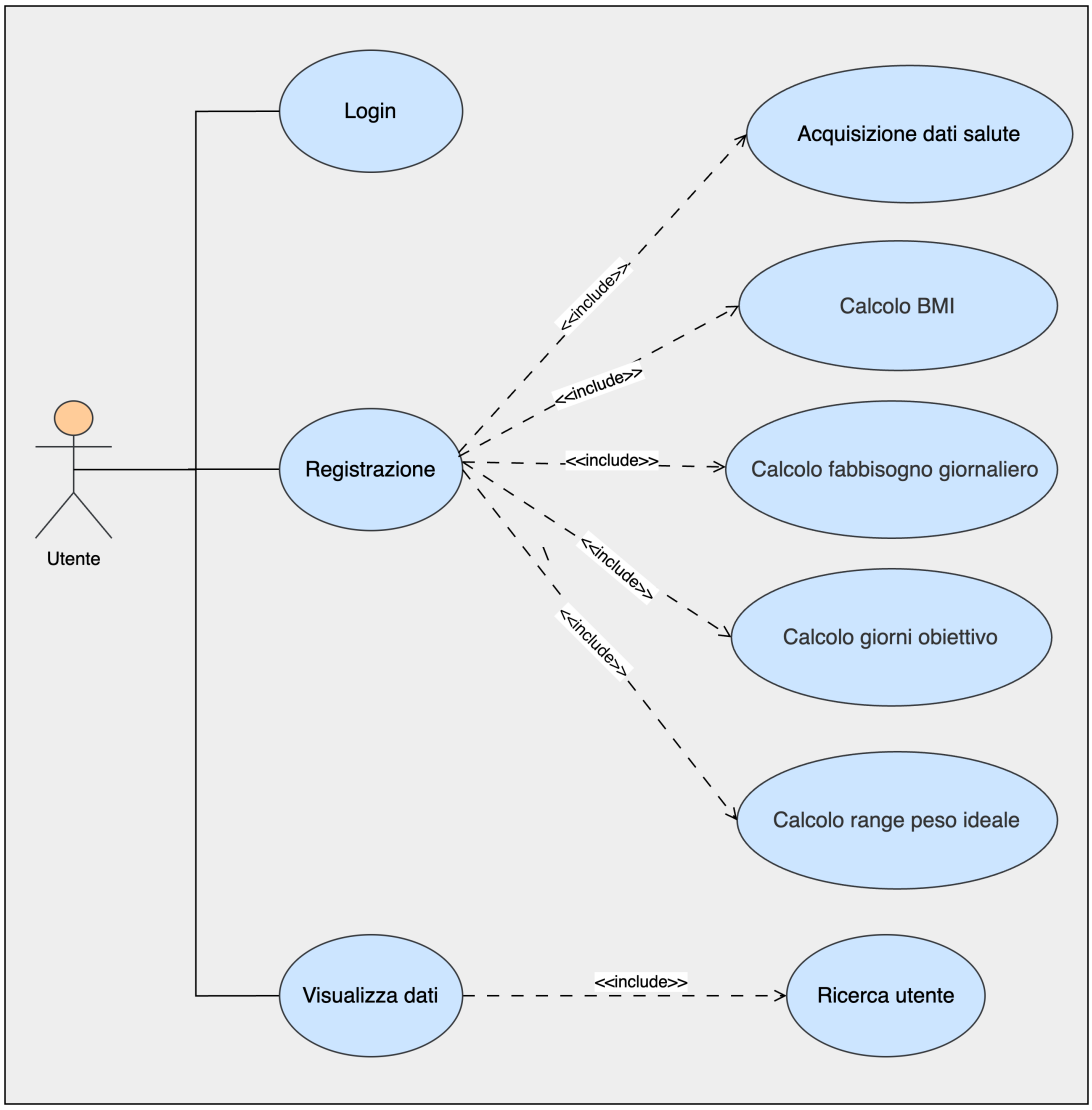

Figura 1.5: Diagramma dei casi d'uso: gestione utente

Di seguito, nella Tabella 1.1 viene riportata una descrizioni dettagliata del casi d'uso *Registrazione*.

Registrazione	Questo caso d'uso consente all'utente di creare un nuovo account nell'applicazione
Attori primari	Utente
Attori secondari	Nessuno
Precondizioni	L'utente non è registrato
Postcondizioni	L'utente è registrato
Sequenza eventi principali	1. Il caso d'uso inizia quando l'utente vuole registrarsi 2. L'utente immette l'indirizzo email e la password 3. *if* le credenziali sono già in uso (a) Il sistema visualizza un opportuno messaggio di errore 4. *else* (a) L'utente inserisce il proprio nome, cognome e le informazioni relative alle allergie da cui è affetto (b) *include* (Acquisizione dati salute) (c) Il sistema mostra i dati dell'utente reperiti da salute (d) *include* (Calcolo BMI) (e) Il sistema mostra lo stato di salute dell'utente (f) *include* (Calcolo range peso ideale) (g) L'utente inserisce il suo obiettivo di peso (h) *if* obiettivo di peso>= peso attuale utente i. Il sistema visualizza un opportuno messaggio di errore; (i) *else* i. L'utente inserisce il suo livello di attività fisica ii. L'utente inserisce la modalità di dimagrimento iii. *include*(Calcolo fabbisogno giornaliero) iv. *include*(Calcolo giorni obiettivo) v. Il sistema mostra il fabbisogno giornaliero e i giorni necessari per il ragiungimento del peso ideale vi. Il sistema infine salva le informazioni su firebase
Sequenza degli eventi alternativa	Nessuna

Tabella 1.1: Descrizione caso d'uso Registrazione

1.4.2 Gestione pasti

La macrocategoria "**gestione pasti**" comprende tutte le funzionalità dell'applicazione dedicate alla registrazione e alla gestione degli alimenti consumati dagli utenti. L'obiettivo principale di questa sezione è facilitare il monitoraggio quotidiano dell'alimentazione, permettendo agli utenti di tenere traccia in modo dettagliato di ciò che mangiano.

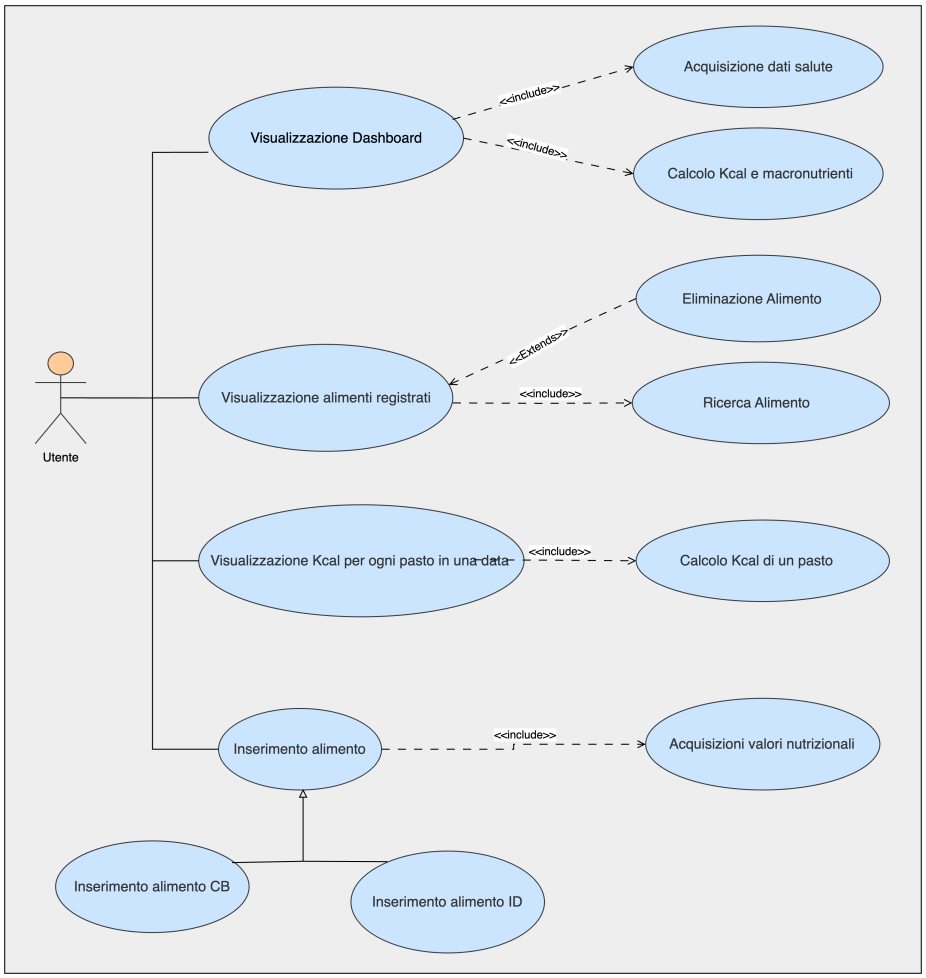

Figura 1.6: Diagramma dei casi d'uso : Gestione pasti

Il diagramma dei casi d'uso illustrato in Figura 1.6 rappresenta le diverse azioni che l'utente, ovvero l'attore principale, può svolgere per gestire gli alimenti. Il sistema fornisce una serie di funzionalità che consentono all'utente di monitorare in modo dettagliato le proprie abitudini alimentari. In particolare, l'utente può interagire con il sistema attraverso diversi casi d'uso, che possono essere:

1. *Visualizzazione Dashboard*
2. *Acquisizione dati salute*
3. *Calcolo Kcal e macronutrienti*
4. *Visualizzazione alimenti registrati*
5. *Eliminazione Alimento*
6. *Ricerca alimento*

7. *Visualizzazione Kcal per ogni pasto in una data*
8. *Calcolo Kcal di un pasto*
9. *Inserimento Alimento CB*
10. *Inserimento alimento Id*
11. *Acquisizione valori nutrizionali*
12. *Inserimento alimento CB*

Di seguito vengono riportate le descrizioni dettagliate dei casi d'uso relativi al *Inserimento Alimento CB* (Tabella 1.2) e alla *Visualizzazione Dashboard* (Tabella 1.3).

Inserimento Alimento CB	Questo caso d'uso consente all'utente di registrare all'interno di un pasto un alimento tramite codice a barre
Attori primari	Utente
Attori secondari	Nessuno
Precondizioni	L'attore primario deve disporre di un account di utilizzo con appropriati diritti d'accesso
Postcondizioni	Nessuno
Sequenza eventi principali	1. Il caso d'uso inizia quando attore primario vuole registrare all'interno di un pasto un alimento tramite il codice a barre 2. Il sistema accede alla telecamera del telefono e scansiona il codice a barre 3. *if* il codice a barre non è valido (a) Il sistema visualizza un opportuno messaggio di errore 4. *else* (a) *include* (Acquisizione valori nutrizionali) (b) *if* alimento ha ingredienti di cui l'attore primario è allergico i. Il sistema informerà l'attore primario con un messaggio (c) Il sistema mostra i valori nutrizionali dell'alimento (d) L'attore primario inserirà la grammatura che ha assunto dell'alimento (e) Il sistema infine registrerà l'alimento su Firebase
Sequenza degli eventi alternativa	Nessuna

Tabella 1.2: Descrizione caso d'uso Inserimento Alimento CB

Visualizzazione Dashboard	Questo caso d'uso consente all'utente di visualizzare una dashboard che fornisce un riepilogo dell'andamento giornaliero, con informazioni come le calorie consumate, i passi effettuati e i macronutrienti assunti
Attori primari	Utente
Attori secondari	Nessuno
Precondizioni	L'attore primario deve disporre di un account di utilizzo con appropriati diritti d'accesso
Postcondizioni	Nessuno
Sequenza eventi principali	1. Il caso d'uso inizia quando attore primario vuole visualizzare la dashboard 2. *include* (Acquisizione dati salute) 3. *include* (Calcolo Kcal e macronutrienti) 4. *if* i dati non sono disponibili (a) Il sistema visualizza un opportuno messaggio di errore 5. *else* (a) Il sistema visualizza un grafico che indica quante chilocalorie l'utente deve ancora assumere nel corso della giornata, insieme alla quantità rimanente di macronutrienti (grassi, proteine e carboidrati) espressi in grammi (b) Il sistema visualizza il numero di passi compiuti e le calorie bruciate dall'attore principale durante la giornata
Sequenza degli eventi alternativa	Nessuna

Tabella 1.3: Descrizione caso d'uso Visualizzazione Dashboard

1.5 Diagramma delle classi

Il diagramma delle classi, svolge un ruolo fondamentale nel rappresentare in modo accurato la realtà che il sistema intende gestire. Le classi possono essere considerate come entità che rappresentano persone, oggetti fisici o concetti astratti, e il loro scopo è modellare il sistema evidenziando le relazioni, gli attributi e le azioni che ciascuna di esse può compiere. Ogni componente del sistema è stato organizzato in modo da svolgere un compito specifico, riducendo così, le interdipendenze e favorendo una gestione più semplice e ordinata del codice. Per questo motivo, le classi sono state suddivise in tre package principali: **Repository**, **Food Recognition e API Integration** e **Model**. In Figura1.7 viene riportato il diagramma delle classi del package "Model". Queste classi definiscono le entita fondamentali e le loro interazioni all'interno dell'applicazione.

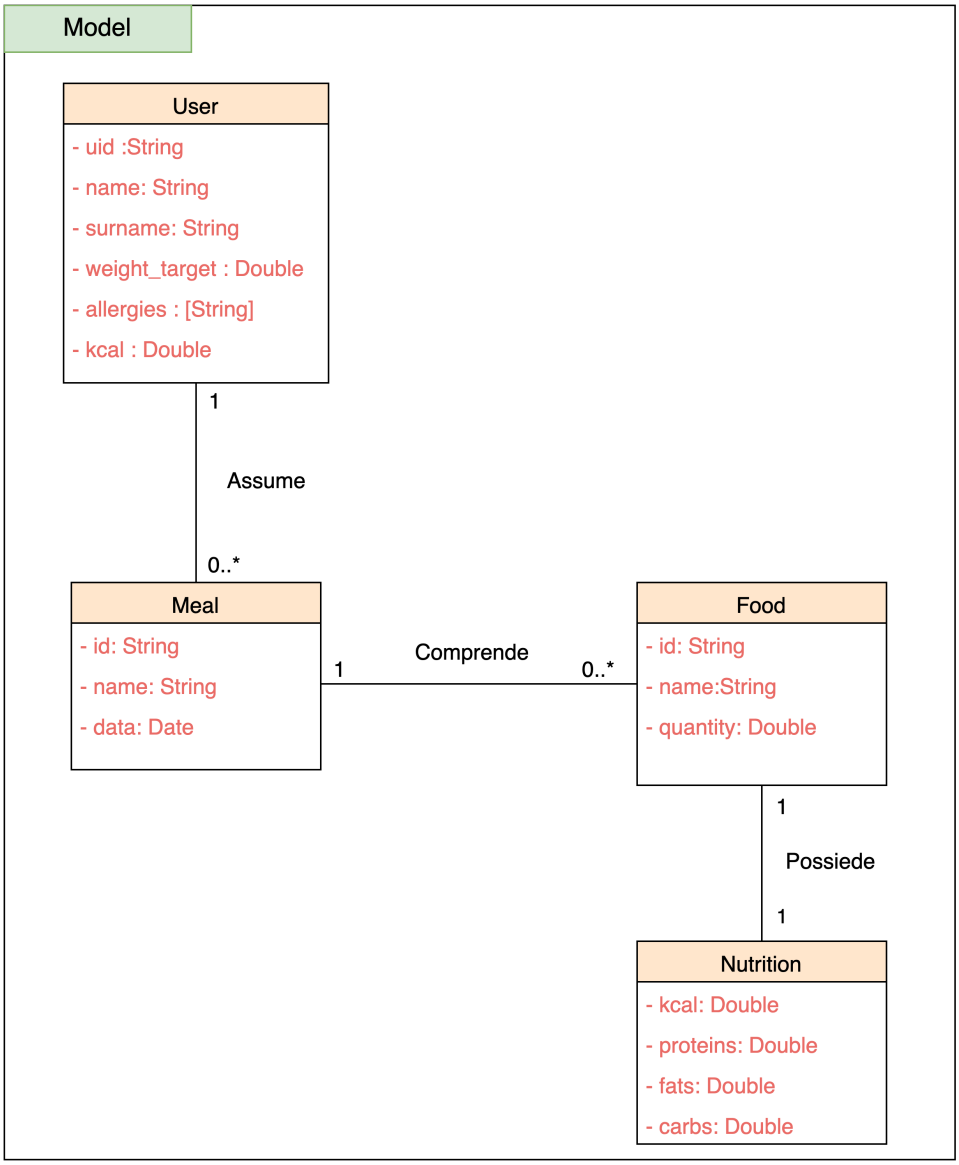

Figura 1.7: Diagramma delle classi analisi del package "Model"

Le classi fondamentali del package "**Model**"sono riportate nella tabella 1.4:

Classe	Descrzione
Utente	Rappresenta l'entità principale del sistema, ossia l'utilizzatore dell'applicazione. Gli attributi associati a questa classe sono le informazioni personali dell'utente e un identificatore univoco chiamato UID (generato automaticamente da firebase al momento della registarzione)
Meal	Rappresenta un pasto associato all'utente
Food	Rappresenta un alimento assunto dall'utente in un pasto
Nutrition	Rappresenta i macronutrienti di un alimento

Tabella 1.4: Descrizione delle classi del package "Model"

In Figura 1.8 viene riportato il diagramma delle classi del package "**Repository**". Il package Repository contiene una classe denominata FirestoreRepository, che si occupa della persistenza dei dati e della comunicazione con il database.

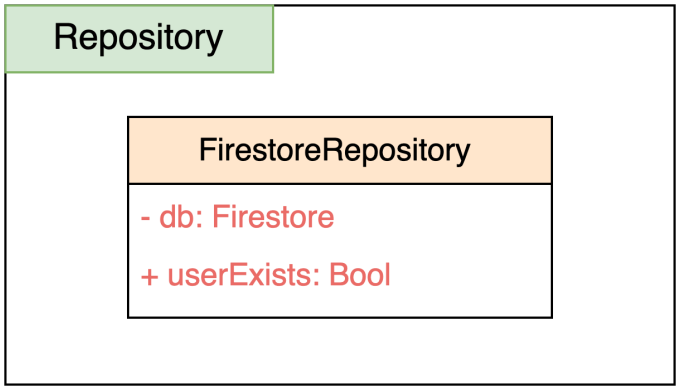

Figura 1.8: Diagramma delle classi analisi del package "Repository"

In Figura 1.9 viene riportato il diagramma delle classi del package " **Food Recognition e API Integration** ". Questo package contiene 3 classi dedicate alla gestione delle funzionalità di riconoscimento e ricerca degli alimenti. Queste classi sono descritte nella Tabella 1.5.

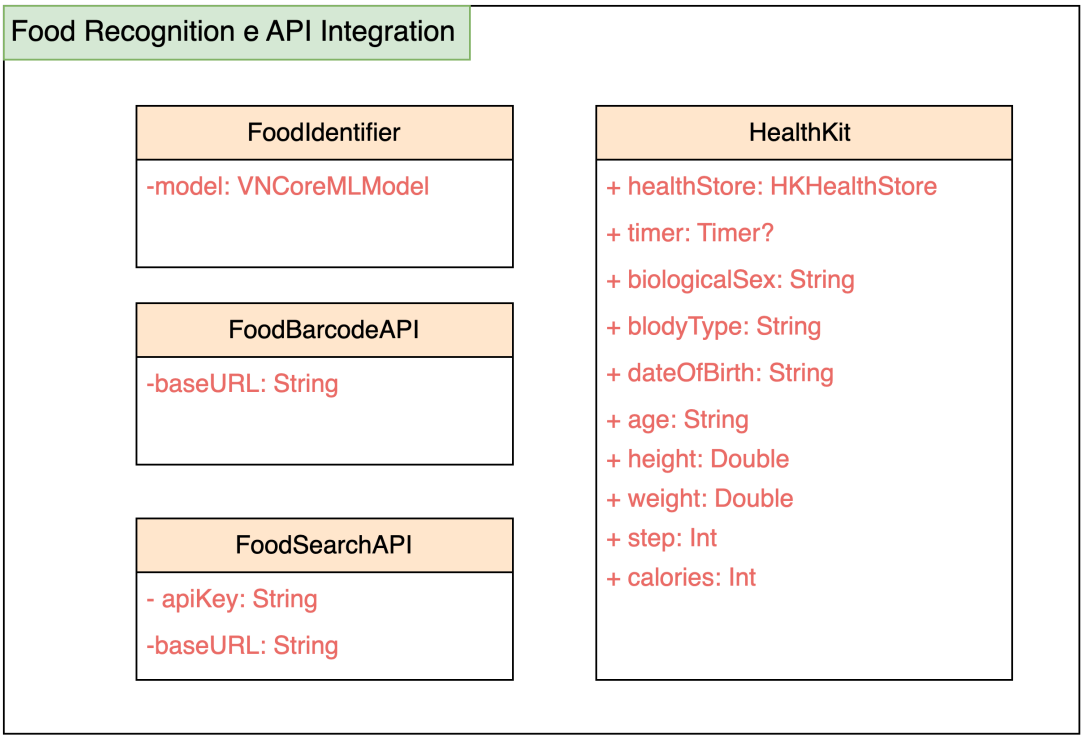

Figura 1.9: Diagramma delle classi analisi del package " Food Recognition e API Integration "

Classe	Descrzione
FoodIdentifier	Gestisce la classificazione di immagini utilizzando un modello di machine learning creato con Create ML
FoodBarcodeAPI	Questa classe, permette di interagire con l'API di Open Food Facts (database collaborativo di prodotti alimentari, che raccoglie informazioni su milioni di alimenti provenienti da tutto il mondo.) per ottenere informazioni sui prodotti tramite il codice a barre.
FoodSearchAPI	Questa classe, permette di interagire con l'API d USDA FoodData Central, utilizzando una query di testo , per ottenere informazioni sui prodotti.
Healthkit	Si occupa di monitorare i dati sanitari dell'utente tramite l'integrazione con l'HealthKit di iOS.

Tabella 1.5: Descrizione delle classi del package "Food Recognition e API Integration "

CAPITOLO 2

Progettazione

In questo capitolo vengono illustrate le scelte progettuali adottate per lo sviluppo dell'applicazione, a partire dall'analisi del pattern architetturale. Questo approccio è stato scelto per soddisfare le specifiche esigenze dell'applicazione, garantendo modularità, manutenibilità e scalabilità del sistema. Successivamente, viene descritta l'implementazione delle classi di progettazione, derivate dalle classi di analisi. Il capitolo include anche la presentazione dei mockup, strumenti essenziali per la progettazione e il testing dell'interfaccia utente prima della fase di sviluppo. Infine, vengono illustrate le soluzioni tecniche adottate per affrontare due aspetti fondamentali dell'applicazione: il calcolo del fabbisogno calorico giornaliero, basato sui dati acquisiti tramite HealthKit e l'identificazione degli alimenti.

2.1 Architettura dell'applicazione

Oggi, lo sviluppo di un'applicazione mobile non si limita solo alla creazione di funzionalità efficienti, ma richiede anche l'implementazione di un'architettura robusta e scalabile così da garantire al sistema prestazioni elevate e una gestione ottimale delle risorse. L'architettura di un'applicazione descrive la struttura e l'organizzazione dei vari componenti, definendo come interagiscono tra loro. Dopo, un'attenta analisi dei requisiti, si è deciso di utilizzare il pattern architetturale MVVM (Model-View-ViewModel). Questa decisione consente di strutturare l'applicazione in modo tale da separare l'interfaccia grafica dalla logica di business. Il pattern MVVM è composto da tre componenti fondamentali (Figura 2.1):

- **Model**: rappresenta dati e logica di buisiness dell'applicazione. Gestisce la persistenza dei dati e può includere chiamate a database, API remote o altre fonti di dati. Questo componente comunica esclusivamente con il ViewModel attraverso eventi o data binding.
- **View**: rappresenta l'interfaccia grafica dell'applicazione, responsabile della presentazione dei dati forniti dal ViewModel.
- **ViewModel**: agisce come intermediario tra il Model e la View, gestendo la logica di presentazione e l'interazione con i dati. Il suo compito principale è recuperare i dati dal Model, elaborarli e trasformarli in una forma adeguata per la presentazione nella View. Oltre a fornire i dati alla View, il ViewModel si occupa anche di gestire gli input provenienti dall'interfaccia utente. Quando l'utente interagisce con la View, il ViewModel risponde a questi input eseguendo le azioni appropriate sul Model.

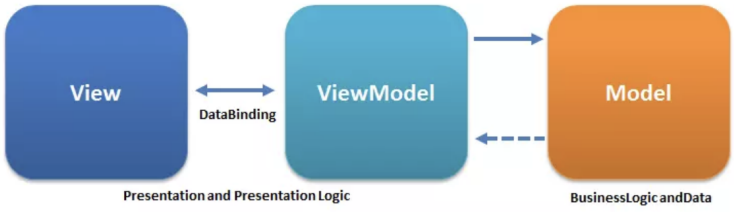

Figura 2.1: Schema rappresentativo dell'architettura Model-View-ViewModel

L'uso di questo pattern è molto diffuso nella programmazione mobile, poichè consente di separare le responsabilità tra i vari componenti dell'applicazione. Quest'ultima facilita la manutenzione e l'aggiornamento del codice.

2.2 Classi di progettazione

Dopo aver discusso l'architettura dell'applicazione e descritto le classi di analisi, è il momento di procedere con l'implementazione delle classi di progettazione. Le classi di progettazione sono delle classi le cui specifiche sono talmente complete che possono essere effettivamente implementate e derivano da due fonti principali: il dominio del problema e il dominio della soluzione. Il dominio del problema descrive il processo di raffinamento delle classi di analisi, trasformandole in classi di progettazione più dettagliate e concrete. Il dominio della soluzione fornisce gli strumenti tecnologici (framework, pattern, librerie) necessari per implementare tali classi, traducendo il design in un sistema funzionante. Le classi di analisi del sistema, una volta modellate e raffinate, sono state convertite in classi di progettazione rispettando la suddivisione in package **Model**, **Repository** e **API**. Nella fase di progettazione il package repository è stato suddiviso in classi specifiche come **UserRepository**, **MealRepository** e **FoodRepository**, applicando il principio di separazione delle responsabilità; così facendo si mantiene il codice ben strutturato e modulare, dato che ogni repository è specializzato nella gestione di un insieme diverso di dati. Inoltre, ogni classe è stata arricchita con dei metodi in grado di implementare le funzionalità richieste dal sistema, come illustrato nella Figura 2.2

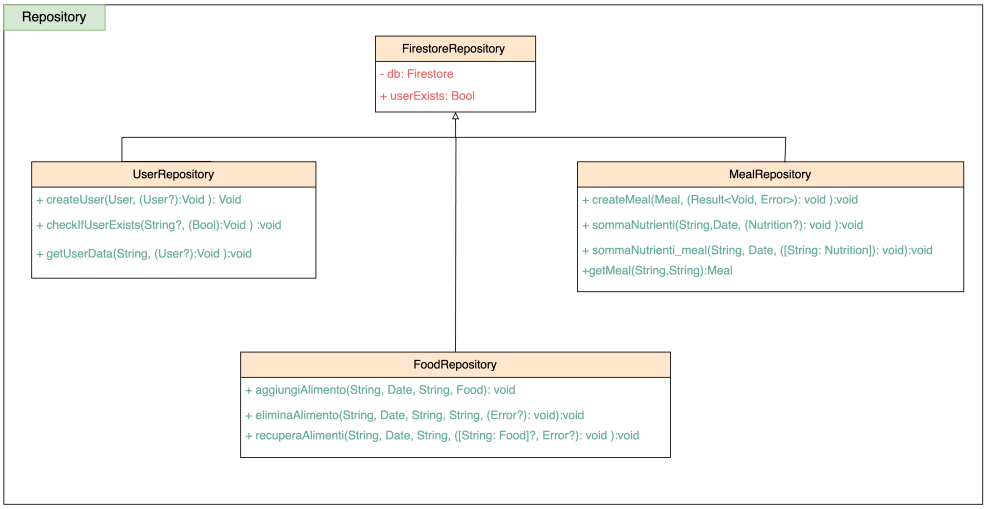

Figura 2.2: Diagramma delle classi di progettazione del package "Repository"

Di seguito, la Tabella 2.1 fornisce una descrizione dettagliata di ciascuna classe del package "Repository".

Classe	Descrzione
UserRepository	si occupa della gestione dei dati relativi agli utenti, come la registrazione, l'autenticazione e il recupero delle informazioni personali
FoodRepository	si occupa della gestione dei dati degli alimenti
MealRepository	si occupa della gestione i dati relativi ai pasti

Tabella 2.1: Descrizione delle classi di progettazione aggiuntive del package "Repository"

In Figura 2.3 è riportato il diagramma delle classi di progettazione del package "Model".

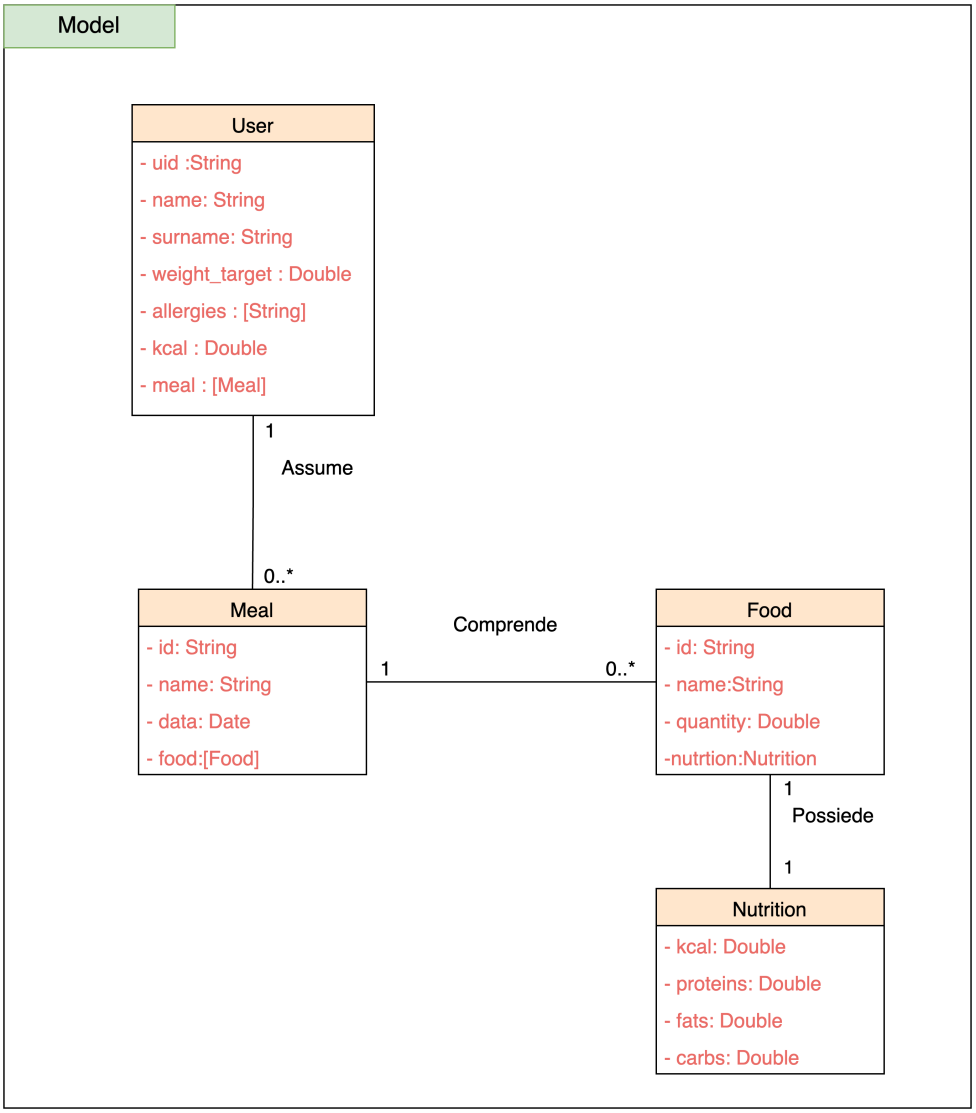

Figura 2.3: Diagramma delle classi di progettazione del package "Model"

Infine, in Figura 3.8 viene riportato l'ultimo package chiamato "API".

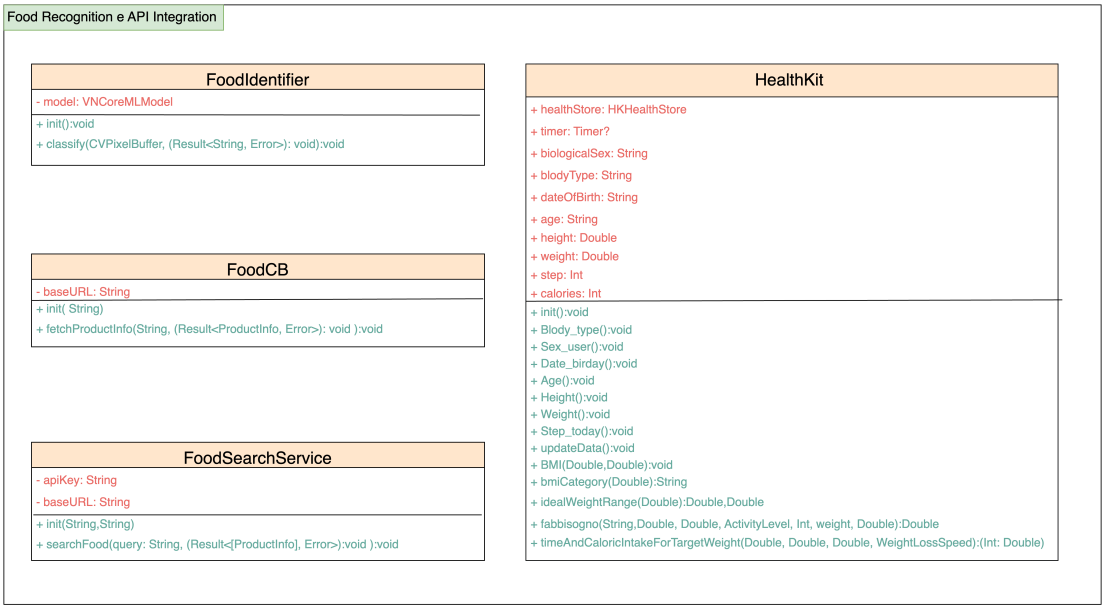

Figura 2.4: Diagramma delle classi di progettazione del package "API"

2.3 Mockup

Nello sviluppo di un'applicazione, l'uso dei mockup rappresenta una pratica fondamentale per la progettazione dell'interfaccia utente. I mockup consentono di visualizzare l'organizzazione e il posizionamento degli elementi grafici, fornendo un'anteprima chiara di come l'interfaccia apparirà agli utenti finali. Per la realizzazione di questi mockup, è stato utilizzato Figma, un software avanzato per il design grafico e la prototipazione, che ha permesso di delineare in modo preciso ogni dettaglio visivo e funzionale. I mockup sono stati suddivisi in base alle principali aree funzionali dell'applicazione, identificate come: **Auth**, **Registrazione**, **Home** e **Food**. Questa suddivisione ha facilitato il processo di progettazione, permettendo di concentrare l'attenzione su ciascuna sezione. In Figura 2.5, sono riportati i mockup relativi alla sezione Auth, che illustrano il flusso di autenticazione dell'utente nel sistema.

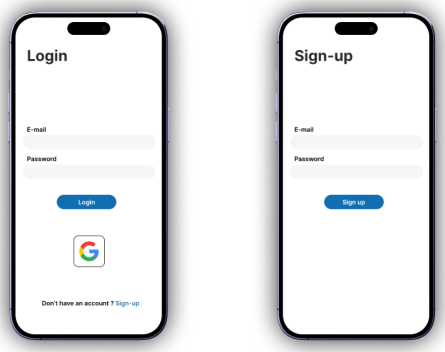

Figura 2.5: Mockup del login e del sign-up

In Figura 2.6 sono illustrati i mockup della sezione di Registrazione, che mostrano il processo di registarzione dell'utente. Durante questa fase, il sistema raccoglie le informazioni personali necessarie per creare un piano calorico personalizzato. Questi dati permettono all'app di adattare il programma di dimagrimento e gli obiettivi calorici in base alle esigenze specifiche dell'utente. In Figura 2.7 sono illustrati i mockup della sezione Home, che raf-

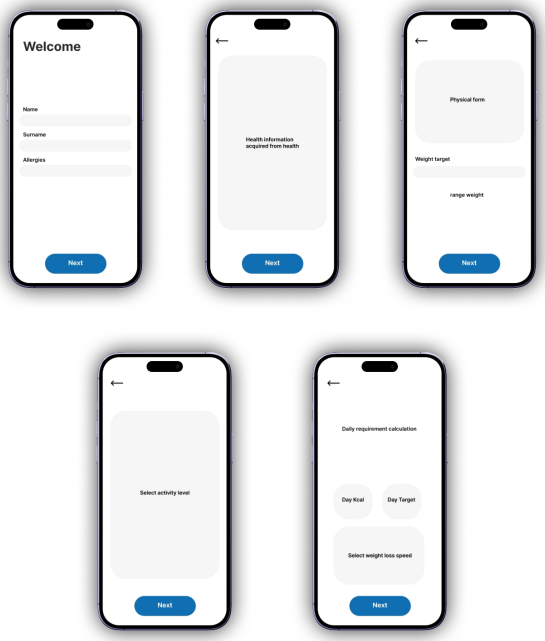

Figura 2.6: Mockup della fase di registarzione dell'utente

figurano due schermate principali. La prima, denominata "Home", presenta la dashboard dell'applicazione, dove vengono visualizzati i dati principali dell'utente, tra cui le calorie assunte (rappresentate graficamente), i macronutrienti consumati, i passi effettuati e le calorie bruciate, offrendo così una panoramica completa dei progressi giornalieri. La seconda schermata, denominata "Account", mostra le informazioni relative all'account dell'utente, insieme ai dati acquisiti da HealthKit, fornendo un riepilogo dei parametri personali e delle attività monitorate.

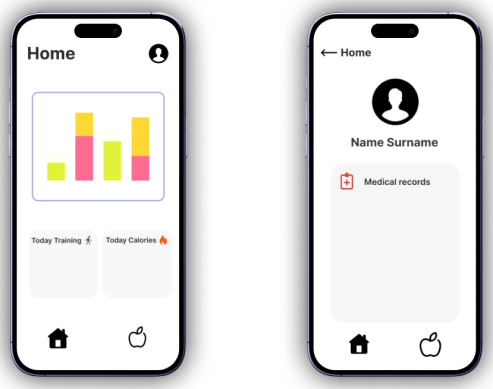

Figura 2.7: Mockup della home e della sezione riguardante l'account utente

In **Figura 2.13** sono mostrati i mockup della sezione **Food**, che presentano le funzionalità principali per monitorare e gestire l'alimentazione quotidiana. La schermata principale offre una panoramica delle calorie consumate nei quattro pasti principali (colazione, pranzo, snack e cena), permettendo all'utente di controllare facilmente l'assunzione calorica giornaliera. L'app consente di identificare gli alimenti sia tramite scansione del codice a barre per i prodotti confezionati, sia attraverso un modello di intelligenza artificiale che riconosce i cibi dalle immagini. Inoltre, è presente una schermata che offre un riepilogo dettagliato degli alimenti consumati durante un pasto, inclusi valori come chilocalorie e macronutrienti.

Figura 2.8: Mockup delle schermate inerenti al DiaryFood

2.4 Soluzione per l'identificazione degli alimenti

L'identificazione degli alimenti è una delle funzionalità più rilevanti dell'applicazione. Quindi, per rendere questo processo intuitivo per l'utente, sono stati implementati due metodi di identificazione: uno basato su Machine Learning, che consente di identificare gli alimenti attraverso una foto e l'altro tramite la scansione del codice a barre. Di seguito, verranno illustrati in dettaglio i due approcci adottati per risolvere queste problematiche.

2.4.1 Identificazione tramite Machine Learning

Il **Machine Learning** è una sottobranca dell'Intelligenza Artificiale che permette ai sistemi di apprendere compiti specifici attraverso l'analisi di grandi quantità di dati, senza essere esplicitamente programmati. Questo processo, noto come apprendimento per generalizzazione, consente al sistema di estrapolare informazioni dai dati osservati. Un sistema di Machine Learning si compone di cinque elementi essenziali:

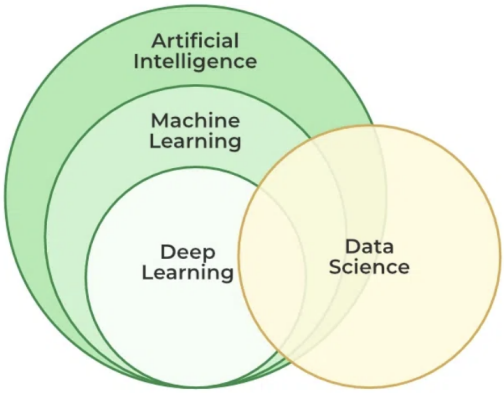

Figura 2.9: Machine Learning

- **Dati** : rappresentano l'esperienza disponibile per l'apprendimento e sono organizzati in un insieme di istanze x_p(dette anche samples o instances) ciascuna caratterizzata da n attributi. (features)
- **Tasks** : definiscono l'obiettivo finale dell'apprendimento, ovvero ciò che il sistema deve imparare a fare per risolvere un determinato problema. Esistono diverse tipologie di tasks, ma le principali sono:
  1. **Supervised Learning** : vengono forniti al modello dei sample etichettati sotto forma di coppie ⟨input,output⟩, con l'obiettivo di stimare una funzione f sconosciuta che collega gli input agli output. Gli input, o variabili indipendenti, rappresentano le caratteristiche osservabili, mentre gli output, o variabili dipendenti (note anche come risposte), rappresentano i risultati attesi. In questo processo, si distinguono due situazioni principali:
     (a) Se l'output è un insieme di categorie, parliamo di classificazione: il modello deve imparare a distinguere tra diversi gruppi discreti;
     (b) Se invece l'output è un valore numerico continuo, siamo di fronte a un problema di regressione: il modello dovrà prevedere un valore preciso, come il prezzo di una casa o la temperatura prevista per domani.
  2. **Unsupervised learning**: i dati forniti al modello non includono input-output, ma sono composti esclusivamente da dati non etichettati.
- **Modello**: descrive la relazione tra i dati utilizzando una rappresentazione adeguata. Le ipotesi sono le funzioni h_w proposte dal modello per approssimare la "vera" funzione f. Queste ipotesi sono parametrizzate da w e costituiscono lo spazio delle ipotesi H, che raccoglie tutte le possibili funzioni candidate per rappresentare accuratamente la relazione tra input e output nei dati.
- **Algoritmo di apprendimento**: ha il compito di esplorare lo spazio delle ipotesi H di un determinato modello, al fine di trovare la migliore approssimazione possibile della funzione f.
- **Validazione**: valuta la capacità di generalizzazione di una determinata ipotesi, misurandone l'accuratezza.

Dopo varie ricerche su come sviluppare un modello efficace per l'identificazione di immagini, si è deciso di utilizzare Create ML. Create ML[1]è uno strumento di Apple in grado di

[1]https://developer.apple.com/documentation/createml

creare e addestrare modelli di Machine Learning, in modo facile e intuitivo. Infatti, Create ML offre diversi modelli predefiniti che possono essere addestrati per vari tipi di applicazioni, tra cui il classificatore di immagini, che è stato fondamentale per implementare la funzionalità di identificazione degli alimenti. Un classificatore di immagini è un modello che, una volta addestrato, è in grado di riconoscere e categorizzare un'immagine in base a ciò che rappresenta. Il processo di addestramento avviene mostrando al modello un insieme di immagini etichettate, cioè immagini già associate a una categoria specifica. Ad esempio, per addestrare un classificatore a riconoscere animali, gli vengono mostrate immagini di elefanti, giraffe e leoni, ciascuna con la propria etichetta (es. Elefante, Giraffa, Leone) come rappresentato in Figura 3.3. Durante l'addestramento, il modello analizza queste immagini e impara a

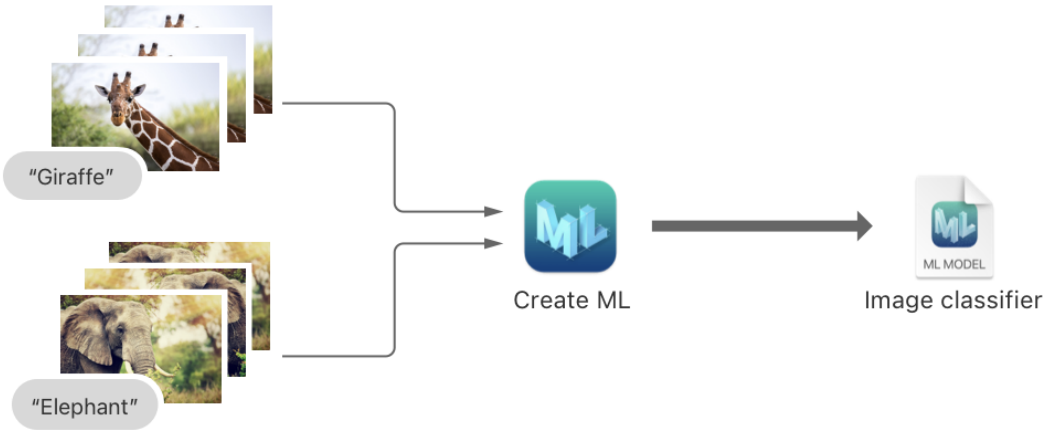

Figura 2.10: Addestarmento modello

riconoscere le caratteristiche comuni di ciascuna categoria. Il modello, grazie alle conoscenze acquisite durante la fase di addestramento, è in grado di identificare automaticamente nuove immagini. Quando analizza un'immagine, il modello genera una previsione ovvero l'etichetta che ritiene più probabile, ad esempio "Giraffa" come mostrato in Figura 2.11.

Figura 2.11: Processo di identificazione degli alimenti tramite machine learning

Oltre all'etichetta è sempre presente un punteggio di confidenza, che esprime il grado di certezza con il quale il modello ha formulato la sua previsione, permettendo di valutare l'affidabilità del risultato. Dopo l'addestramento, il modello viene valutato su un set di dati non utilizzato in precedenza, al fine di verificare la sua capacità di generare previsioni accurate su nuove immagini. Se le prestazioni risultano soddisfacenti, il modello viene esportato come un file Core ML (formato ottimizzato per l'integrazione in applicazioni iOS) che inseguito verrà integrato nell'applicazione.

2.4.2 Identificazione tramite codice a barre

Per identificazione degli alimenti tramite il codice a barre, è stato scelto di utilizzare il framework di acquisizione video di Apple, AVCaptureSession[2]. Questa tecnologia permette all'applicazione di sfruttare la fotocamera del dispositivo per leggere e interpretare i codici a barre in tempo reale. L'uso di AVCaptureSession consente di configurare una sessione di cattura video che acquisisce le immagini direttamente dalla fotocamera. Sono supportati vari formati di codici a barre comuni, come EAN e PDF417, garantendo la compatibilità con una vasta gamma di prodotti alimentari confezionati.

2.5 Soluzione per il Recupero dei Dati Nutrizionali

Per fornire informazioni precise e aggiornate sugli alimenti l'applicazione utilizza due approcci distinti in base alla modalità di identificazione della pietanza.

2.5.1 USDA

Quando un alimento viene identificato tramite Core ML, l'app utilizza l'etichetta restituita dal modello per effettuare una ricerca attraverso un'API fornita dall'USDA[3] (United States Department of Agriculture). Questa API consente di accedere a un ampio database di informazioni nutrizionali, recuperando i valori dei macronutrienti (calorie, proteine, grassi e carboidrati) associati all'alimento riconosciuto.

Figura 2.12: Logo USDA

2.5.2 Open food facts

Quando un alimento viene identificato tramite codice a barre, l'applicazione sfrutta l'API di Open Food Facts[4] per ottenere le informazioni. Nel momento in cui l'utente scansiona il codice a barre di un prodotto confezionato, l'app invia una richiesta al database esterno utilizzando il codice come identificatore unico. In questo modo, è possibile recuperare automaticamente i dati nutrizionali completi, inclusi i valori di macronutrienti, la foto del prodotto e ulteriori dettagli rilevanti. Questo metodo semplifica il processo di aggiunta dei cibi al diario alimentare, consentendo agli utenti di registrare rapidamente e con precisione i prodotti confezionati senza dover inserire manualmente le informazioni.

2.6 Soluzione fabbisogno giornaliero

Il calcolo del fabbisogno calorico giornaliero è un aspetto fondamentale per mantenere uno stile di vita sano, poiché consente di stabilire un corretto bilancio energetico adattato

[2]https://developer.apple.com/documentation/avfoundation/avcapturesession
[3]https://fdc.nal.usda.gov
[4]https://world.openfoodfacts.org

Figura 2.13: Logo Open food facts

alle esigenze e agli obiettivi personali. Per semplificare questo processo e offrire un'esperienza utente intuitiva, l'applicazione ha integrato il framework HealthKit, che consente di raccogliere e gestire i dati relativi alla salute e al fitness direttamente dai dispositivi Apple. Per garantire un'elevata accuratezza nella determinazione del fabbisogno calorico giornaliero, è stata fondamentale la collaborazione con il nutrizionista Dott. Christian Ciulla, il cui contributo ha svolto un ruolo fondamentale nella definizione delle formule e dei parametri utilizzati. Grazie alla sua esperienza professionale, sono stati identificati e calibrati i parametri fondamentali per il calcolo del fabbisogno energetico giornaliero, con particolare attenzione al metabolismo basale (BMR) e al il dispendio energetico totale (TDEE). Tale approccio, fondato sulle migliori pratiche in ambito nutrizionale, permette all'applicazione di fornire all'utente una stima personalizzata e attendibile del proprio fabbisogno energetico, favorendo il mantenimento di un bilancio calorico equilibrato. Le formule per il calcolo del BMR e del TDEE e delle uleriori metriche studiate sono state prensentate nel Capitolo 1.

CAPITOLO 3

Implementazione

In questo capitolo verranno esaminate le tecnologie e le metodologie utilizzate per l'implementazione dell'applicazione, con un'analisi approfondita delle scelte tecniche e delle soluzioni adottate per ciascuna funzionalità. Saranno presentate in dettaglio le modalità di costruzione dell'interfaccia utente, la gestione dei dati sanitari e alimentari e l'integrazione del modello di machine learning per il riconoscimento automatico degli alimenti. Inoltre, verrà descritto il processo di addestramento del modello di machine learning, illustrando le fasi di preparazione, addestramento e validazione dei dati.

3.1 Tecnologie utilizzate

In questa sezione verranno illustrate le tecnologie adottate per l'implementazione dell'applicazione, evidenziando le loro caratteristiche principali e il ruolo significativo che hanno avuto nel processo di sviluppo.

3.1.1 Swift-UI

L'applicazione è stata sviluppata utilizzando SwiftUI[1], un framework moderno introdotto da Apple durante la WWDC 2019. Basato sul linguaggio di programmazione Swift, SwiftUI offre un approccio innovativo e semplificato per costruire interfacce utente su più piattaforme, tra cui iOS, macOS, watchOS e tvOS. A differenza dell'approccio imperativo tradizionalmente adottato da UIKit, SwiftUI segue un paradigma di programmazione dichiarativo, consentendo agli sviluppatori di definire l'interfaccia utente in modo più intuitivo e conciso.

Figura 3.1: Logo di SwiftUI

[1] https://developer.apple.com/documentation/swiftui

3.1.2 HealtKit

HealthKit[2] è il framework di Apple dedicato alla gestione dei dati sanitari degli utenti, offrendo uno spazio centralizzato per archiviare informazioni relative alla salute e al fitness, accessibile su iPhone e Apple Watch. Con il consenso dell'utente, le applicazioni possono interagire con l'HealthKit store per accedere e condividere questi dati in modo sicuro e controllato. L'integrazione di HealthKit consente all'applicazione di monitorare parametri come il conteggio dei passi, l'altezza e altre metriche rilevanti, offrendo così agli utenti un'esperienza personalizzata.

Figura 3.2: Logo HealthKit

3.1.3 Firebase

Firebase[3] è una piattaforma di sviluppo mobile offerta da Google che fornisce una serie di servizi cloud, tra cui autenticazione degli utenti, database in tempo reale, storage, hosting, notifiche push e analisi delle applicazioni. Nel contesto dell'applicazione sviluppata, Firebase è stato utilizzato principalmente per gestire l'autenticazione degli utenti e per l'archiviazione e gestione dei dati attraverso Cloud Firestore.

Figura 3.3: Logo Firebase

Cloud Firestore è un database NoSQL offerto da Firebase e Google Cloud, noto per la sua flessibilità e scalabilità. È progettato per archiviare e sincronizzare i dati tra dispositivi mobili, web e server in tempo reale. Firestore organizza i dati in documenti, che a loro volta sono contenuti all'interno di collezioni. I documenti possono memorizzare una vasta gamma di tipi di dati, dai numeri semplici agli oggetti più complessi, e possono contenere sottocollezioni, rendendo possibile la creazione di strutture di dati ben organizzate. I dati dell'applicazione sono strutturati in Firestore in modo logico e scalabile, facilitando la gestione e consentendo un accesso efficiente alle informazioni. La struttura parte dalla collezione principale "Users",

[2]https://developer.apple.com/documentation/healthkit
[3]http://firebase.google.com

in cui ogni documento rappresenta un singolo utente, identificato univocamente tramite il proprio UID. All'interno di ogni documento utente sono presenti sottocollezioni, ciascuna delle quali rappresenta una data specifica in cui l'utente ha registrato degli alimenti. Ogni sottocollezione giornaliera contiene fino a quattro documenti, ognuno dedicato a un pasto principale: colazione, pranzo, cena e spuntino. In questi documenti vengono archiviati i dettagli degli alimenti consumati per ogni pasto, fornendo una struttura organizzata e facilmente accessibile.

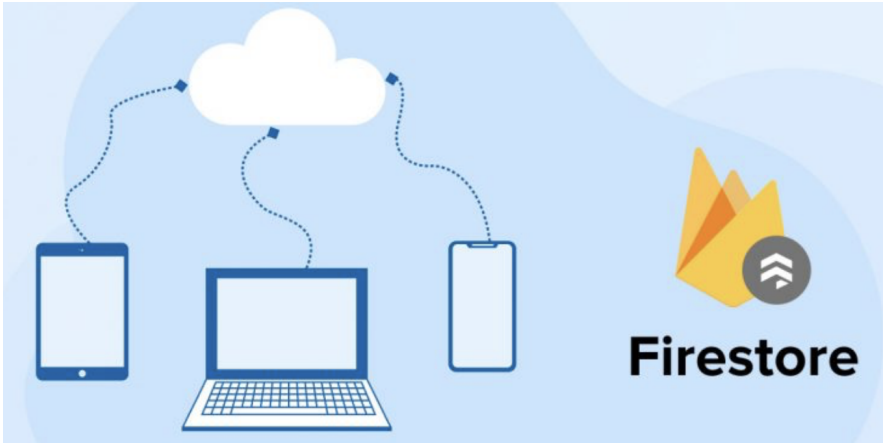

Figura 3.4: Schema di funzionamento Firestore

3.1.4 Core ML

Core ML[4] è un framework fornito da Apple, utilizzato per integrare modelli di machine learning nelle applicazioni iOS, offrendo una rappresentazione unificata per tutti i modelli. Attraverso le API di Core ML, è possibile utilizzare i dati dell'utente per effettuare previsioni, allenare o perfezionare modelli direttamente sul dispositivo, garantendo al contempo sicurezza e privacy dei dati, in quanto tutto il processo avviene localmente. In Figura 3.5 viene rappresentato il flusso di lavoro di CoreML che può essere descritto come segue:

Figura 3.5: Flusso di lavoro di Core ML

1. **Modello Core ML**: un file contenente il modello addestrato, risultato di un algoritmo di machine learning applicato a un set di dati. Questo modello può essere creato utilizzando Create ML, un'app integrata in Xcode che semplifica l'addestramento e la creazione di modelli nel formato Core ML.

2. **Core ML Framework**: il framework responsabile dell'esecuzione, ottimizzazione e gestione del modello.

[4]https://developer.apple.com/documentation/coreml/

3. **App**: l'applicazione utilizza il modello per fare previsioni o analisi in tempo reale, direttamente sui dati forniti dall'utente.

Nell'applicazione è stato utilizzato Core ML per integrare un modello di machine learning creato con Create ML.

3.2 Implementazione classe HealtKit

La classe HealthKit funge da ViewModel, gestendo l'acquisizione e la gestione dei dati provenienti dal framework HealthKit. In Swift, per implementare un ViewModel, si utilizza il protocollo ObservableObject, che consente di notificare automaticamente alle View ogni volta che le proprietà dell'oggetto cambiano. Questa notifica automatica avviene grazie al property wrapper @Published, applicato alle proprietà del ViewModel. Quando una proprietà contrassegnata con @Published viene modificata, tutte le View che osservano l'oggetto vengono aggiornate automaticamente. Per consentire a una View di osservare il ViewModel e reagire ai suoi cambiamenti, SwiftUI utilizza il property wrapper ObservedObject. Questo permette alla View di rimanere sincronizzata con le modifiche del ViewModel e aggiornare la propria interfaccia in tempo reale. Di seguito, verranno illustrate alcune delle funzioni implementate all'interno della classe HealthKit.

3.2.1 Implementazione della funzione Steptoday

La funzione *Steptoday()* è stata sviluppata per raccogliere il numero totale di passi effettuati dall'inizio della giornata fino al momento attuale, utilizzando il framework HealthKit (riportato in Figura 3.6). Per ottenere questo risultato, la funzione utilizza una query di tipo **HKStatisticsQuery**, che consente di eseguire calcoli statistici su un insieme di campioni quantitativi corrispondenti a un determinato tipo di dato. In questo caso, la query calcola la somma dei passi, basandosi su un predicato che filtra i dati a partire dall'inizio del giorno fino all'ora corrente. Per identificare il tipo di dato specifico relativo al conteggio dei passi, la funzione utilizza **HKQuantityType**, una sottoclasse concreta di HKObjectType. **HKObjectType** è una classe astratta che rappresenta i diversi tipi di dati gestiti da HealthKit, mentre **HKQuantityType** rappresenta dati quantitativi, ovvero valori numerici misurabili. In questo caso, viene utilizzato l'identificatore **HKQuantityTypeIdentifier.stepCount** per specificare il tipo di dato relativo al conteggio dei passi.

```
245     func Step_today() {
246           guard let stepCount = HKObjectType.quantityType(forIdentifier: .stepCount) else {
247               print("Step Count Type is no longer available in HealthKit")
248               return
249           }
250
251           let calendar = Calendar.current
252         _ = calendar.startOfDay(for: Date())
253
254           let predicate = HKQuery.predicateForSamples(withStart:.Start_Day, end: Date(), options: .strictStartDate)
255           let query = HKStatisticsQuery(quantityType: stepCount, quantitySamplePredicate: predicate, options: .cumulativeSum) { _, result, error in
256               if let error = error {
257                   print("Error: \(error.localizedDescription)")
258                   return
259               }
260               guard let result = result, let quantity = result.sumQuantity() else {
261                   print("No data available")
262                   return
263               }
264               let steps = Int(quantity.doubleValue(for: HKUnit.count()))
265               DispatchQueue.main.async {
266                   self.step = steps
267               }
268           }
269           healthStore.execute(query)
270       }
```

Figura 3.6: Implementazione della funzione Steptoday

3.2.2 Implementazione della funzione Height

La funzione *Height*() è progettata per recuperare l'altezza dell'utente registrata in HealthKit. Per ottenere questo dato, viene utilizzata una query di tipo **HKSampleQuery**, che permette di estrarre campioni di dati sanitari dall'HealthKit store. La query è configurata per recuperare tutti i campioni relativi all'altezza (**HKQuantityTypeIdentifier.height**), senza restrizioni sul numero di risultati e senza applicare filtri specifici. Al completamento della query, se sono presenti campioni disponibili, viene selezionato l'ultimo campione registrato, dal quale si estrae il valore dell'altezza in metri. Questo valore viene quindi assegnato alla proprietà height della classe. Se non ci sono dati disponibili o si verifica un errore durante l'esecuzione, viene visualizzato un messaggio informativo che segnala il problema.

```
195     func Height() {
196             let height_ = HKQuantityType(.height)
197             let query = HKSampleQuery(sampleType: height_, predicate: nil, limit: HKObjectQueryNoLimit, sortDescriptors: nil) { (query, results, error) in
198                 guard let samples = results as? [HKQuantitySample], error == nil else {
199                     print("Errore durante l'esecuzione della query: \(error?.localizedDescription ?? "Errore sconosciuto")")
200                     return
201                 }
202                 if let heightSample = samples.last {
203                     let heightValue = heightSample.quantity.doubleValue(for: HKUnit.meter())
204                     self.height = heightValue
205                 } else {
206                     print("Non sono disponibili dati sull'altezza.")
207                 }
208             }
209             healthStore.execute(query)
210         }
```

Figura 3.7: Implementazione della funzione Height

3.3 Implementazione della classe FoodBarcodeAPI

In questa sezione viene motrata l'implementazione della classe FoodBarcodeAPI, responsabile del reperimento dei valori nutrizionali di un alimento confezionato. Di seguito viene riportato il codice della classe (Figura 3.8)

```
11  class FoodBarcodeAPI {
12      private let baseURL: String
13      private let session: URLSession
14
15      init(baseURL: String = "https://world.openfoodfacts.org/api/v0", session: URLSession = .shared) {
16          self.baseURL = baseURL
17          self.session = session
18      }
19
20      func fetchProductInfo(forCode code: String, completion: @escaping (Result<ProductInfo, Error>) -> Void) {
21          let urlString = "\(baseURL)/product/\(code).json"
22
23          guard let url = URL(string: urlString) else {
24              completion(.failure(NSError(domain: "InvalidURL", code: -1, userInfo: [NSLocalizedDescriptionKey: "Invalid URL"])))
25              return
26          }
27
28          let task = session.dataTask(with: url) { data, response, error in
29              if let error = error {
30                  completion(.failure(error))
31                  return
32              }
33
34              guard let data = data else {
35                  completion(.failure(NSError(domain: "NoData", code: -1, userInfo: [NSLocalizedDescriptionKey: "No data received from server"])))
36                  return
37              }
38
39              do {
40                  let decoder = JSONDecoder()
41                  let productResponse = try decoder.decode(ProductResponse.self, from: data)
42                  completion(.success(productResponse.product))
43              } catch {
44                  completion(.failure(error))
45              }
46          }
47          task.resume()
48      }
49  }
```

Figura 3.8: Implementazione della classe FoodBarcodeAPI

3.4 Implementazione di HomeView

La HomeView dell'applicazione, riporta una dashboard, progettata per monitorare lo stile di vita e le abitudini alimentari dell'utente. Questa schermata integra e visualizza i dati provenienti da HealthKit e Firestore, fornendo un quadro completo e aggiornato della salute quotidiana dell'utente. Grazie a una serie di indicatori visivi e barre di progresso, l'utente può facilmente tenere sotto controllo il proprio fabbisogno calorico, il consumo di macronutrienti e il livello di attività fisica. In Figura 3.9 è mostrato il codice della HomeView.

```swift
struct HomeView: View {
    @ObservedObject var homeViewModel: HomeViewModel
    var body: some View {
        NavigationStack {
            if homeViewModel.isLoading {
                ProgressView("Loading...")
                    .progressViewStyle(CircularProgressViewStyle())
                    .frame(maxWidth: .infinity, maxHeight: .infinity)
                    .edgesIgnoringSafeArea(.all)
                    .onAppear {
                        homeViewModel.loadData()
                    }
            } else {
                VStack {
                    if let user = homeViewModel.User.user {
                        let needs = calculateMacronutrientNeeds(totalCalories: user.kcal, proteinPercentage: 15, carbohydratePercentage: 50, fatPercentage: 35)
                        VStack {
                            CircularProgressBar(progress: homeViewModel.NutrientsDay!.kcal, maxCalories: user.kcal)
                                .transition(.opacity)
                                .animation(.easeInOut(duration: 1.5), value: homeViewModel.isDataLoaded)
                            Divider()
                            HStack(spacing: 0) {
                                MacronutrientProgressView(macronutrientName: "Carbo", icon: "laurel.leading", currentAmount: homeViewModel.NutrientsDay!.carbs, maxAmount: needs.carbohydrates, color: Color.yellow)
                                    .transition(.slide)
                                    .animation(.easeInOut(duration: 1.5), value: homeViewModel.isDataLoaded)
                                MacronutrientProgressView(macronutrientName: "Proteine", icon: "fish.fill", currentAmount: homeViewModel.NutrientsDay!.proteins, maxAmount: needs.proteins, color: Color.red)
                                    .transition(.slide)
                                    .animation(.easeInOut(duration: 1.5), value: homeViewModel.isDataLoaded)
                                MacronutrientProgressView(macronutrientName: "Fats", icon: "drop.fill", currentAmount: homeViewModel.NutrientsDay!.fats, maxAmount: needs.fats, color: Color.brown)
                                    .transition(.slide)
                                    .animation(.easeInOut(duration: 1.5), value: homeViewModel.isDataLoaded)
                            }
                        }
                        .background(Color(.secondarySystemBackground))
                        .cornerRadius(10)
                        .padding()
                    } else {
                        Text("Nessun dato disponibile")
                            .transition(.opacity)
                            .animation(.easeInOut(duration: 1.5), value: homeViewModel.isDataLoaded)
                    }
                    Spacer()
                    LazyVGrid(columns: Array(repeating: GridItem(spacing: 8), count: 2)) {
                        Icon_Home(Name: "Activity today", NameI1: "Step", NameI2: "Run", Image_Sting: "figure.walk", Image_Sting2: "figure.run", ColorImage: .green, today1: "\(homeViewModel.healt.step)")

                        Icon_Home(Name: "Calorie today", NameI1: "Consumed", NameI2: "Hired", Image_Sting: "flame.fill", Image_Sting2: "flame.fill", ColorImage: Color(uiColor: .orange), today1: "\(homeViewModel.healt.calories)")
                    }
                    .padding()
                    .navigationTitle("Home")
                    .toolbar {
                        NavigationLink(destination: Account(healt: homeViewModel.healt, authViewModel: homeViewModel.authViewModel, firestoreManager: homeViewModel.firestoreManager, userModel: homeViewModel.User)) {
                            Image(systemName: "person.crop.circle")
                                .resizable()
                        }
                    }
                }
                .onAppear {
                    homeViewModel.loadData()
                }
            }
        }
    }
}
```

Figura 3.9: Implementazione di HomeView

Struttura della View: la HomeView è gestita tramite un oggetto @ObservedObject chiamato homeViewModel, che rappresenta il ViewModel della Home. Questo componente si occupa di raccogliere, elaborare e gestire i dati provenienti da HealthKit e Firebase, nonché di implementare la logica necessaria per l'aggiornamento dell'interfaccia utente. La HomeView si divide in due sezioni principali:

1. **Stato di caricamento**: se *homeViewModel.isLoading* è impostato su true, viene visualizzato un indicatore di progresso (ProgressView), che occupa l'intero schermo e avvisa l'utente che i dati sono in fase di caricamento.

2. **Visualizzazione dei dati utente**:

   (a) una volta caricati i dati, se l'utente è presente, viene visualizzato un grafico circolare che rappresenta il progresso delle calorie rispetto al fabbisogno giornaliero. Inoltre, all'interno di un HStack, sono mostrati tre indicatori di progresso per carboidrati, proteine e grassi, che illustrano il consumo attuale rispetto agli obiettivi giornalieri, utilizzando la funzione MacronutrientProgressView.

   (b) se i dati dell'utente non sono disponibili, viene visualizzato un messaggio di testo che informa l'utente con "Nessun dato disponibile".

3.5 Addestramento e integrazione del modello Core ML

Il processo di addestramento del modello si è diviso in tre fasi principali:

1. **Preparazione del dataset**: la fase iniziale ha riguardato la preparazione del dataset FOOD101, scaricato dalla piattaforma Kaggle. Questo dataset è composto da immagini di alimenti, accuratamente etichettate e organizzate in 101 categorie diverse, ciascuna rappresentante un tipo specifico di cibo. Ogni categoria comprende circa 1.000 immagini, suddivise in 750 per l'addestramento e 250 per il test, offrendo una vasta gamma di esempi visivi per addestrare efficacemente modelli di machine learning per il riconoscimento degli alimenti.

2. **Addestramento del modello**: una volta completata la preparazione del dataset, è stato avviato il processo di addestramento del modello utilizzando Create ML. L'addestramento ha compreso 75 iterazioni, mirate a ottimizzare il processo di apprendimento e migliorare la capacità del modello di riconoscere correttamente le immagini.

3. **Test e validazione**: dopo l'addestramento, il modello è stato sottoposto a una fase di test e validazione utilizzando un set di immagini non impiegate durante l'addestramento. Questo approccio ha permesso di valutare la capacità del modello di generalizzare su nuove immagini.

Dopo il completamento di queste fasi, il modello è stato esportato e caricato all'interno dell'applicazione. Per interagire con il modello, è stata sviluppata una classe denominata FoodClassifier, progettata per facilitare la comunicazione tra l'applicazione e il modello di Machine Learning. Di seguito è riportato il codice della classe implementata (Figura 3.10)

```swift
class FoodClassifier {
    private let model: VNCoreMLModel

    init?() {
        // Carica il modello ML creato con Create ML
        guard let coreMLModel = try? VNCoreMLModel(for: Food_classificator().model) else {
            return nil
        }
        self.model = coreMLModel
    }

    func classify(pixelBuffer: CVPixelBuffer, completion: @escaping (Result<String, Error>) -> Void) {
        let request = VNCoreMLRequest(model: model) { (request, error) in
            if let error = error {
                completion(.failure(error))
                return
            }

            guard let results = request.results as? [VNClassificationObservation],
                  let firstResult = results.first else {
                completion(.failure(NSError(domain: "com.example", code: -1, userInfo: [NSLocalizedDescriptionKey: "Could not classify"])))
                return
            }
            if firstResult.confidence > 0.75 {
                completion(.success(firstResult.identifier))
            } else {
                completion(.failure(NSError(domain: "com.example", code: -1, userInfo: [NSLocalizedDescriptionKey: "Low confidence"])))
            }
        }

        let handler = VNImageRequestHandler(cvPixelBuffer: pixelBuffer, options: [:])
        DispatchQueue.global(qos: .userInitiated).async {
            do {
                try handler.perform([request])
            } catch {
                completion(.failure(error))
            }
        }
    }
}
```

Figura 3.10: Implementazione della classe FoodClassifier

3.5.1 Implementzione della funzione classify

La funzione classify è progettata per eseguire la classificazione di immagini utilizzando i framework Core ML e Vision di Apple. La funzione accetta come input un'immagine rappresentata da un oggetto di tipo CVPixelBuffer e restituisce, come output, l'etichetta della classe identificata con successo oppure un errore dettagliato in caso di problemi durante l'elaborazione.

1. **Richiesta Core ML**: La funzione inizia creando un oggetto VNCoreMLRequest, che consente di utilizzare modelli Core ML per analizzare immagini attraverso il framework Vision. Questo oggetto è associato a una closure di completamento che gestisce i risultati forniti dal modello.

2. **Gestione dei risultati**: All'interno della closure di completamento, la funzione gestisce due scenari principali:

    - **Errore durante l'elaborazione**: Se si verifica un errore, la funzione invoca la closure di completamento passando un risultato di tipo .failure contenente l'errore.
    - **Elaborazione riuscita**: Se l'elaborazione ha successo, la funzione estrae i risultati come array di VNClassificationObservation, che rappresentano le possibili classificazioni con i relativi livelli di confidenza. Tra questi, viene selezionata la classificazione con il livello di confidenza più alto. Se tale valore supera la soglia del 75%, la funzione restituisce l'etichetta della classe come risultato.

CAPITOLO 4

Test e User Experience

In questo capitolo vengono illustrate le strategie adottate per garantire la qualità dell'applicazione, concentrandosi sui test e sull'esperienza utente. I test hanno avuto un ruolo fondamentale nel verificare la correttezza delle funzionalità e la stabilità del sistema, permettendo di individuare e risolvere eventuali problemi sin dalle prime fasi di sviluppo. Parallelamente, l'esperienza utente è stata progettata per rendere l'interazione con l'app semplice, intuitiva e piacevole, accompagnando gli utenti attraverso le diverse funzionalità dell'applicazione.

4.1 Test

I test sono stati fondamentali per garantire la funzionalità e la stabilità dell'applicazione. La loro implementazione ha permesso di identificare e correggere eventuali problemi nelle prime fasi di sviluppo, riducendo così il rischio di malfunzionamenti durante l'uso finale. Sono stati condotti principalmente due tipi di test unitari: uno per il calcolo del fabbisogno calorico giornaliero e uno per il recupero dell'altezza dell'utente tramite HealthKit. I test sono stati eseguiti utilizzando il framework XCTest di Xcode[1], uno strumento integrato che facilita la creazione e gestione dei test automatizzati in iOS, macOS, watchOS e tvOS. XCTest supporta diversi tipi di test, tra cui:

- **Test Unitari**, verificano il corretto funzionamento di singole funzioni o metodi, assicurando che ogni componente dell'applicazione produca i risultati attesi.
- **Test di Integrazione:** Valutano l'interazione tra diversi moduli dell'applicazione, garantendo che collaborino efficacemente per fornire le funzionalità desiderate.
- **Test di Interfaccia Utente (UI Testing)**, simulano le azioni dell'utente sull'interfaccia dell'applicazione, verificando che i flussi di interazione rispondano correttamente e che l'esperienza utente sia fluida e coerente.
- **Test delle Prestazioni**, consentono di misurare le performance di specifici blocchi di codice, rilevando eventuali regressioni nelle prestazioni e aiutando a prevenire rallentamenti futuri.

Di seguito viene riportato il codice relativo ai test implementati. Nella figura 4.1 è illustrato il codice del test relativo al recupero dell'altezza dell'utente, che verifica il corretto funzionamento della funzionalità di lettura dei dati da HealthKit.

[1] https://developer.apple.com/documentation/xctest/

```swift
class HealthKitTests: XCTestCase {

    func testHeight() {

        let expectedHeight = 1.75

        let mockHealthKit = HealthKit()
        mockHealthKit.Height { height in
            XCTAssertEqual(height!, expectedHeight, accuracy: 0.01, "Expected height value to be \(expectedHeight) meters")
        }
    }


}
```

Figura 4.1: Test del riperimento del valore dell'altezza da Salute

In Figura 4.2 è illustrato il codice del test relativo al calcolo del fabbisogno calorico giornaliero, che verifica la correttezza della funzione fabbisogno.

```swift
final class HealtyLifeTests: XCTestCase {

    func testFabbisognoMale() {
            let sex = "male"
            let weight = 70.0
            let height = 1.75
            let age = 25
            let activity = ActivityLevel.sedentary
            let weight_target = 70.0


            let expectedBMR = 66.47 + (13.75 * weight) + (5.00 * (height * 100)) - (6.75 * Double(age))
            let expectedTDEE = expectedBMR * activity.Tde
            let result = fabbisogno(sex: sex, weight: weight, height: height, activity: activity, age: age, weight_target: weight_target)

            XCTAssertEqual(result.BMR, expectedBMR, accuracy: 0.1, "Il calcolo del BMR per l'uomo non è corretto")
            XCTAssertEqual(result.TDEE, expectedTDEE, accuracy: 0.1, "Il calcolo del TDEE per l'uomo non è corretto")
        }

        func testFabbisognoFemale() {
            let sex = "female"
            let weight = 55.0
            let height = 1.65
            let age = 30
            let activity = ActivityLevel.veryActive
            let weight_target = 55.0


            let expectedBMR = 655.09 + (9.56 * weight) + (1.84 * (height * 100)) - (4.67 * Double(age))
            let expectedTDEE = expectedBMR * activity.Tde
            let result = fabbisogno(sex: sex, weight: weight, height: height, activity: activity, age: age, weight_target: weight_target)


            XCTAssertEqual(result.BMR, expectedBMR, accuracy: 0.1, "Il calcolo del BMR per la donna non è corretto")
            XCTAssertEqual(result.TDEE, expectedTDEE, accuracy: 0.1, "Il calcolo del TDEE per la donna non è corretto")
        }

}
```

Figura 4.2: Test del metodo di calcolo del fabbisogno giornaliero

4.2 User experience

L'applicazione è stata progettata per garantire un utilizzo semplice e immediatamente comprensibile. Ogni elemento dell'interfaccia è stato studiato per consentire agli utenti di navigare agevolmente e accedere rapidamente alle funzionalità desiderate. Il design si ispira agli standard delle applicazioni native di Apple, adottando uno stile moderno e una palette cromatica accurata, mirata a migliorare la leggibilità e a offrire un'esperienza visiva piacevole. L'approccio minimalista dell'applicazione elimina gli elementi non essenziali, focalizzandosi esclusivamente sulle funzionalità fondamentali. Questo consente di garantire un utilizzo intuitivo e accessibile anche a utenti con minore esperienza tecnologica, senza compromettere la qualità dell'esperienza offerta. Di seguito vengono presentate le schermate finali dell'applicazione, che riflettono i principi di progettazione descritti in precedenza.

4.2.1 Auth

L'applicazione inizia con una fase di autenticazione, che consente agli utenti di accedere al proprio account o crearne uno nuovo. In Figura 4.3 sono illustrate le schermate relative alla fase di autenticazione dell'applicazione.

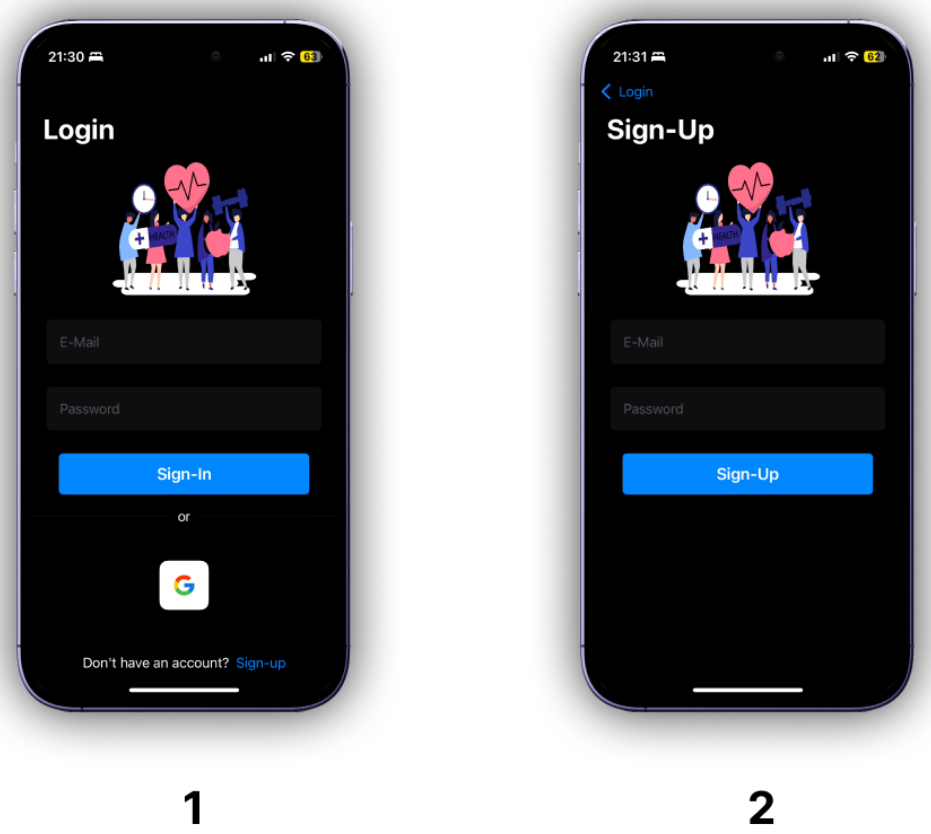

Figura 4.3: Interfacce di autenticazione

1. **Figura 4.3 (1)**: mostra la schermata che consente all'utente di accedere inserendo le proprie credenziali personali oppure utilizzando l'opzione di login tramite Google.
2. **Figura 4.3 (2)**: mostra la schermata di registarzione che guida i nuovi utenti nel processo di creazione di un account, fornendo un'interfaccia chiara e intuitiva per completare la registrazione.

4.2.2 Fase di registrazione

Nel caso di un nuovo utente, l'applicazione offre un processo di registrazione strutturato in più schermate. Questa fase raccoglie le informazioni essenziali per personalizzare l'esperienza dell'utente. Di seguito sono illustrate le schermate relative alla fase di autenticazione dell'applicazione:

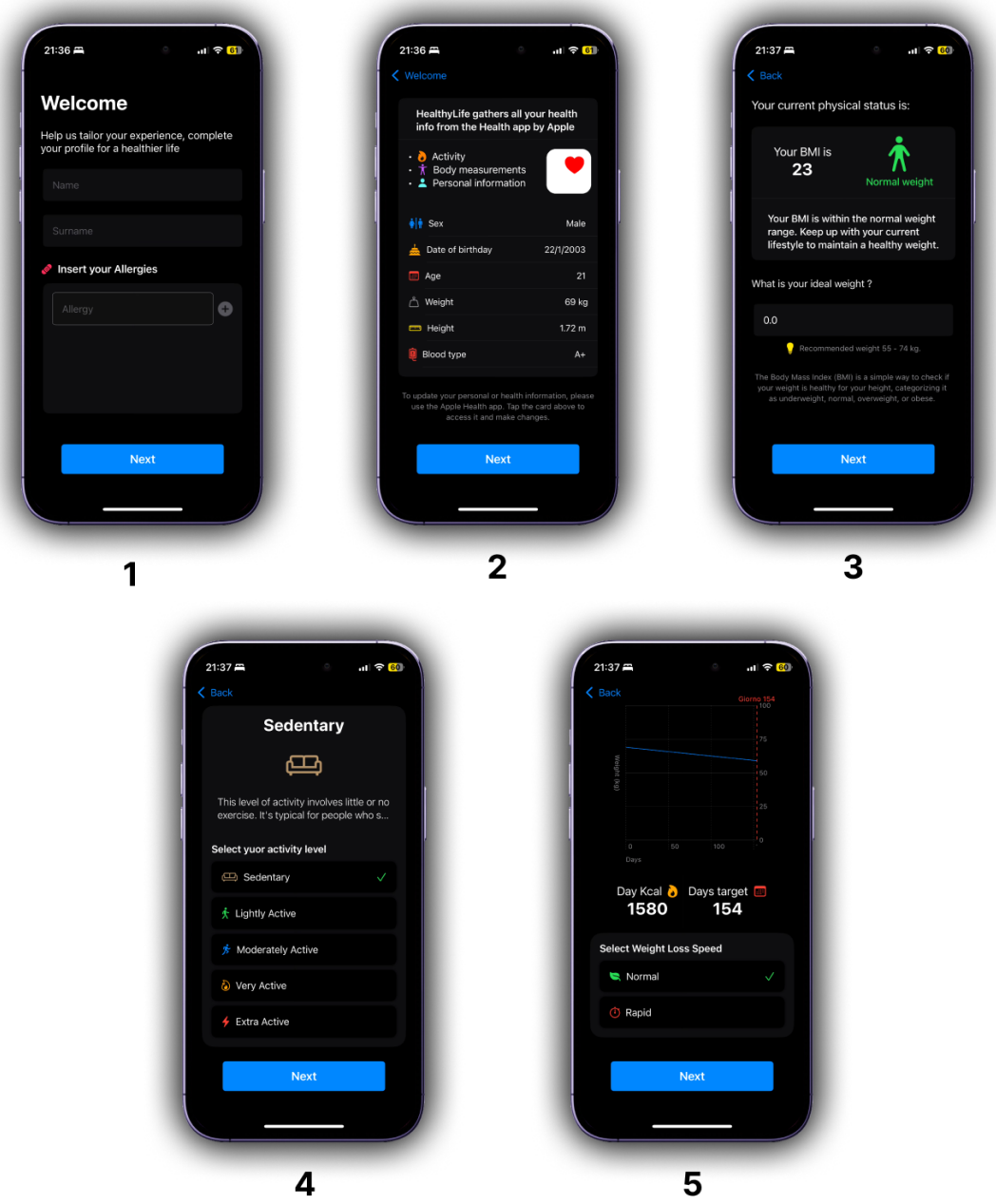

Figura 4.4: Interfacce per la registarzione

1. **Figura 4.4 (1)**: mostra la schermata di registrazione in cui l'utente è invitato a fornire informazioni come nome, cognome e allergie. Questi dati vengono utilizzati per personalizzare l'esperienza d'uso e gestire eventuali restrizioni alimentari.

2. **Figura 4.4 (2)**: mostra la schermata di visualizzazione dei dati raccolti tramite HealthKit. Inoltre, cliccando sulla card corrispondente, l'utente viene reindirizzato all'applicazione Salute per modificare i propri dati sanitari.

3. **Figura 4.4 (3)**: mostra la schermata dove viene calcolato e mostrato l'indice di massa corporea (BMI) dell'utente. L'utente ha la possibilità di specificare il proprio obiettivo di peso per creare un piano personalizzato.

4. **Figura 4.4 (4)**: mostra la schermata dove l'utente può selezionare il proprio livello di attività fisica, un'informazione che verrà utilizzata per personalizzare il calcolo del fabbisogno calorico giornaliero.

5. **Figura 4.4 (5)**: mostra la schermata si visualizzazione del fabbisogno calorico giornaliero , calcolato in base ai dati forniti dall'utente. Inoltre, vengono generati un grafico e una stima dei giorni necessari per raggiungere l'obiettivo di peso impostato.

4.2.3 Home

Una volta completata la registrazione o l'accesso, l'utente viene reindirizzato alla schermata Home. In Figura 4.5 sono illustrate le schermate relative alla schermata Home:

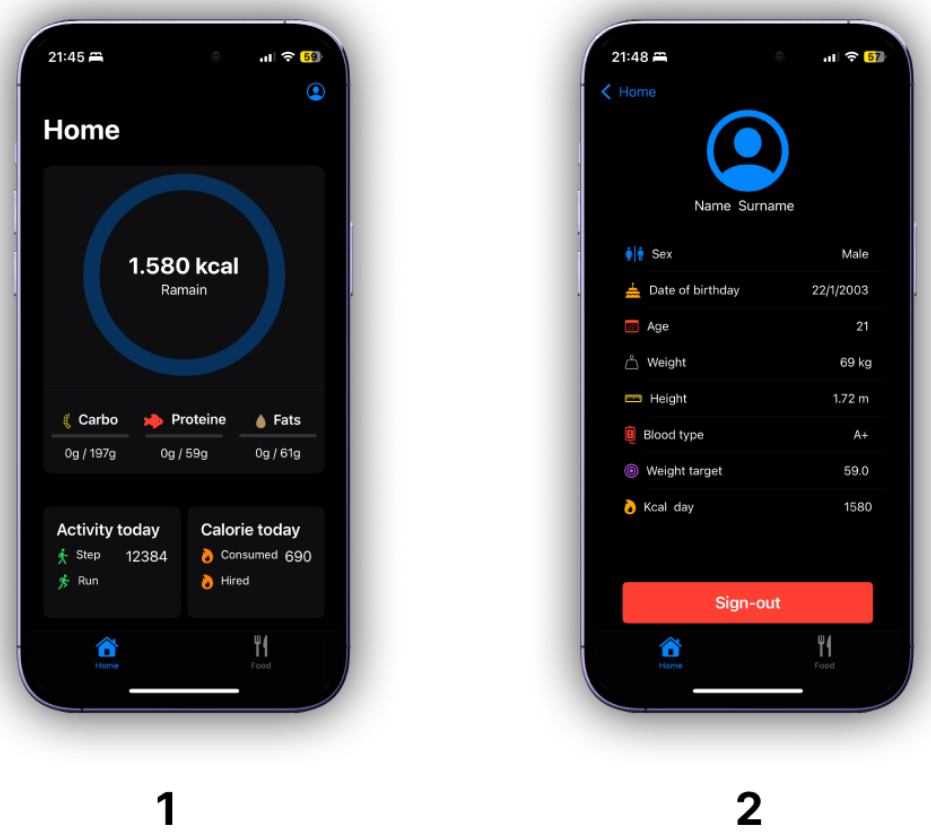

Figura 4.5: Viste relative alla sezione "Home"

1. **Figura 4.5 (1)**: mostra la schermata principale che offre una dashboard intuitiva dove l'utente può visualizzare rapidamente i dati relativi alla propria attività fisica, alle calorie consumate e ai macronutrienti.

2. **Figura 4.5 (2)**: mostra la schermata dove l'utente ha la possibilità di gestire le proprie informazioni personali.

4.2.4 Food

La sezione dedicata alla gestione degli alimenti comprende cinque schermate che permettono agli utenti di monitorare la loro alimentazione in modo efficace. Qui è possibile registrare i pasti, consultare informazioni nutrizionali e gestire la lista degli alimenti consumati. Di

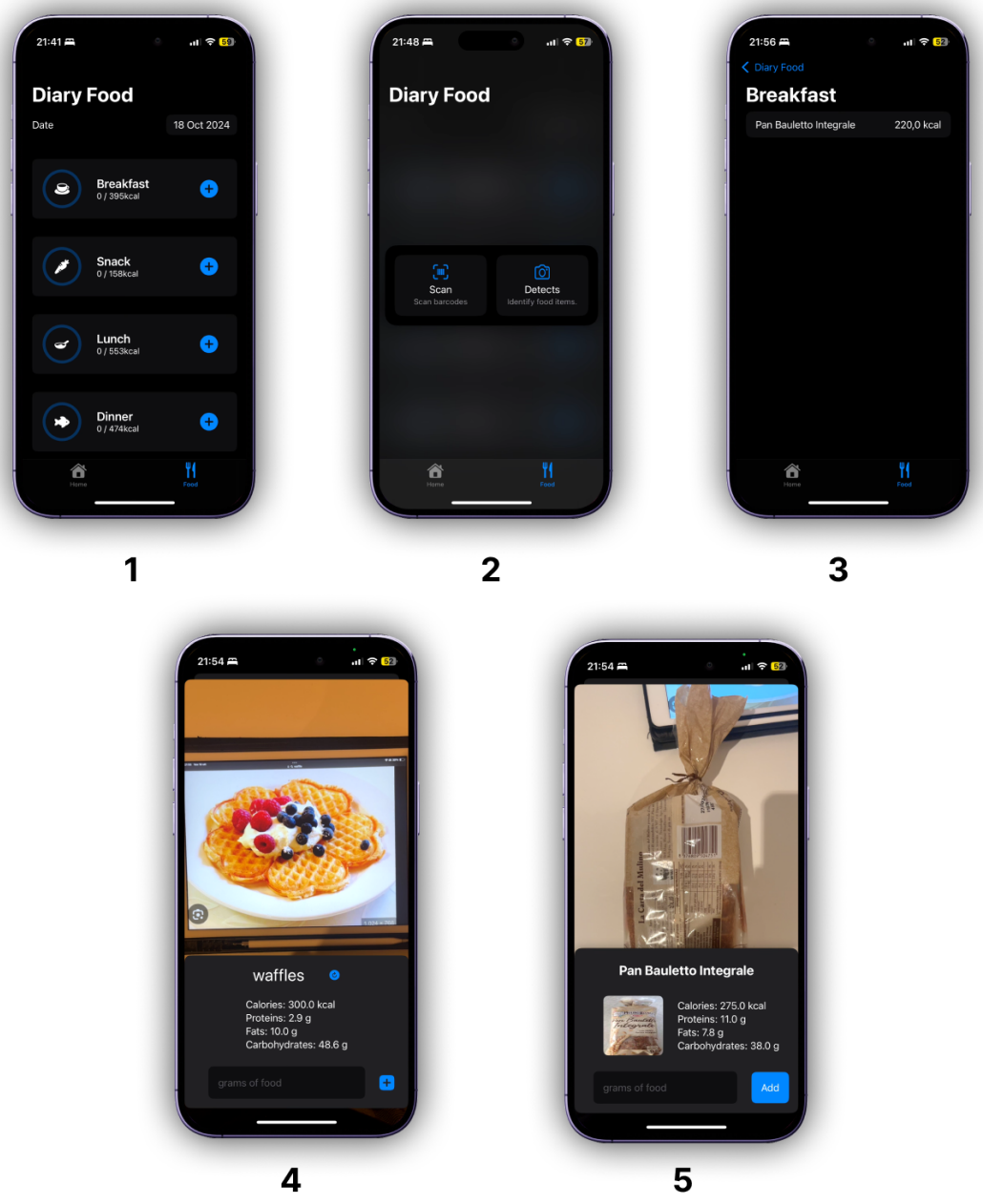

Figura 4.6: Viste relative alla gestione del cibo nella sezione "Food"

seguito sono illustrate le schermate principali della sezione Food:

1. **Figura 4.6 (1)**: mostra la schermata che offre una visione completa degli alimenti consumati e delle calorie totali assunte durante la giornata. L'utente può cambiare la data visualizzata utilizzando il selettore di data, facilitando così il monitoraggio dei pasti anche nei giorni precedenti.

2. **Figura 4.6 (2)**: mostra la schermata che consente all'utente di aggiungere nuovi alimenti al proprio diario alimentare. Cliccando sul pulsante, l'utente può scegliere tra diverse modalità di identificazione degli alimenti: scansione del codice a barre oppure riconoscimento tramite un modello di Machine Learning.

3. **Figura 4.6 (3)**: mostra la schermata che fornisce una panoramica completa degli alimenti registrati in un singolo pasto. L'utente può visualizzare la lista degli alimenti consumati, con i relativi dettagli nutrizionali, e gestire facilmente ogni elemento.

4. **Figura 4.6 (4)**: mostra la schermata dove l'utente può utilizzare la fotocamera del dispositivo per identificare automaticamente gli alimenti tramite un modello di Machine Learning integrato. Una volta riconosciuto un alimento, viene mostrata una card che include i valori nutrizionali (come calorie, proteine, carboidrati e grassi) e un campo di testo dove l'utente può inserire la grammatura consumata.

5. **Figura 4.6 (5)**: mostra la schermata che permette all'utente di scansionare il codice a barre di un prodotto alimentare per identificarlo automaticamente. Dopo la scansione, viene mostrata una card che include informazioni dettagliate, come i valori nutrizionali del prodotto e una foto dello stesso, insieme a un campo di testo per specificare la quantità consumata.

CAPITOLO 5

Conclusione

5.1 Discussione

Lo sviluppo di questa applicazione ha rappresentato un'esperienza significativa e ricca di sfide, mirata alla creazione di uno strumento innovativo per promuovere uno stile di vita sano. In un contesto in cui sovrappeso e obesità rappresentano condizioni sempre più diffuse e preoccupanti, l'applicazione si pone l'obiettivo di semplificare e rendere immediato il monitoraggio dell'alimentazione e dell'attività fisica. Questo risultato è stato raggiunto grazie all'integrazione di tecnologie avanzate, come HealthKit e sistemi di intelligenza artificiale, che garantiscono agli utenti un'esperienza completa ed efficace. Il progetto ha permesso di affrontare e approfondire diverse aree di interesse, dalla progettazione di un modello di Machine Learning, alla creazione di un'interfaccia utente progettata per essere intuitiva e minimale. Ogni fase dello sviluppo, dalla definizione dei requisiti al testing finale, è stata caratterizzata da sfide tecniche significative, tra cui la comprensione dei principi del Machine Learning e l'integrazione del modello con l'applicazione. Tra i risultati più rilevanti ottenuti spicca la capacità dell'applicazione di offrire un supporto concreto agli utenti, consentendo loro di monitorare parametri fondamentali come calorie, macronutrienti e attività fisica in modo semplice e immediato. In conclusione, questo progetto non rappresenta soltanto la realizzazione di un'applicazione innovativa, ma ha costituito un'importante occasione di crescita personale e professionale. Ha permesso di consolidare competenze tecniche avanzate, sviluppare un pensiero analitico e creativo e soprattutto, di scoprire una profonda passione per la progettazione di applicazioni. Il percorso intrapreso ha quindi posto basi solide per il mio futuro professionale, alimentando la mia determinazione a sviluppare strumenti tecnologici innovativi, progettati per migliorare la qualità della vita e favorire il benessere degli utenti. Come affermava Alan Turing, "La scienza di oggi è la tecnologia di domani". Questo progetto, coniugando innovazione tecnologica e attenzione alle esigenze delle persone, rappresenta un passo in questa direzione, ponendo le basi per futuri sviluppi che possano continuare a migliorare la qualità della vita e il loro benessere.

5.2 Sviluppi futuri

L'applicazione, pur essendo già completa, può essere ulteriormente arricchita con nuove funzionalità. Di seguito vengono proposte alcune idee per sviluppi futuri che potrebbero ampliarne le potenzialità e offrire un'esperienza utente ancora più avanzata e personalizzata:

- **Ampliamento del modello di riconoscimento alimentare**: l'attuale modello di Machine Learning è stato addestrato utilizzando il dataset FOOD101, che include un numero limitato di categorie alimentari. Un possibile sviluppo futuro potrebbe prevedere l'espansione del dataset e l'ottimizzazione del modello per riconoscere una gamma più ampia di alimenti, migliorando la capacità di classificazione e includendo anche cibi meno comuni o appartenenti a diverse culture alimentari.

- **Integrazione di una sezione per la creazione di diete personalizzate**: si propone l'introduzione di una funzionalità che consenta agli utenti di generare piani alimentari personalizzati basati sulle proprie esigenze nutrizionali e obiettivi di salute. Questa sezione fornirebbe suggerimenti dietetici specifici, adattati alle preferenze personali, alle eventuali intolleranze o allergie e ai traguardi di fitness, aiutando gli utenti a seguire un regime alimentare bilanciato e su misura.

- **Sezione dedicata alle ricette salutari**: un ulteriore sviluppo potrebbe essere l'aggiunta di una raccolta di ricette salutari, che offrano spunti per pasti bilanciati e nutrienti. Le ricette potrebbero includere informazioni dettagliate sui valori nutrizionali, istruzioni di preparazione e varianti personalizzabili per adattarsi a diverse preferenze o esigenze dietetiche. Questa funzionalità favorirebbe abitudini alimentari sane e arricchirebbe l'esperienza culinaria degli utenti.

- **Chatbot nutrizionale intelligente**: per migliorare il supporto e l'interazione con gli utenti, si potrebbe integrare un chatbot nutrizionale. Questo assistente virtuale sarebbe in grado di rispondere a domande sull'alimentazione, offrire suggerimenti personalizzati e spiegare le proprietà nutritive degli alimenti. Inoltre, il chatbot potrebbe fornire raccomandazioni proattive basate sui dati raccolti, aiutando gli utenti a migliorare la loro dieta e a raggiungere i propri obiettivi di salute in modo consapevole ed efficace.

Questi possibili sviluppi non solo arricchirebbero le funzionalità dell'applicazione, ma la renderebbero uno strumento ancora più versatile e mirato, in grado di rispondere alle diverse esigenze degli utenti.

Bibliografia

Websites consulted

- Obesità: il rapporto 2022 dell'OMS Europa– https://poisson.phc.dm.unipi.it/~quattrocchi/ML.pdf
- Machine Learning– https://poisson.phc.dm.unipi.it/~quattrocchi/ML.pdf
- Core ML – https://developer.apple.com/documentation/coreml/
- Firebase – https://firebase.google.com

Ringraziamenti

"Desidero ringraziare profondamente il mio relatore, il Dott. Enrico Corradini, per la sua disponibilità, professionalità e il costante supporto durante questo percorso. La sua guida è stata preziosa, e attraverso il suo insegnamento ho avuto modo di crescere sia dal punto di vista accademico che personale."

"Alla mia mamma, la persona più speciale e insostituibile della mia vita. Sei stata il mio sostegno più grande, sempre al mio fianco, pronta a incoraggiarmi con il tuo amore incondizionato e quel sorriso capace di rendere ogni giornata più serena. Mi hai insegnato il valore della forza, della gentilezza e della libertà, lasciandomi scegliere il mio cammino senza mai impormi limiti, ma accompagnandomi con una fiducia che mi ha fatto sentire sicuro e amato. La tua presenza, discreta ma fondamentale, è stata il pilastro su cui ho potuto contare nei momenti più importanti. Ogni traguardo che ho raggiunto, ogni obiettivo conquistato, porta con sé la tua presenza e il tuo sostegno. Se oggi sono arrivato fino a qui, lo devo anche a te, che hai sempre creduto in me e mi hai dato la forza per non arrendermi. Grazie, mamma, per tutto ciò che hai fatto e che continui a fare per me. Non smetterò mai di esserti grato."

"A mio papà, un uomo di poche parole, che davanti a me si mostra sempre duro, quasi a voler nascondere ogni emozione. Non mi dice mai 'bravo' o 'ce l'hai fatta', ma quando parla con i suoi amici mi trasforma in un dio dell'informatica, capace di fare cose straordinarie. È lì, in quelle parole dette a loro e non a me, che scopro tutto il suo orgoglio e il suo affetto, celati dietro quel carattere silenzioso e riservato. Grazie, papà, per il tuo modo unico di esserci, per il tuo sostegno che si fa sentire anche senza troppe parole, e per farmi capire, a modo tuo, quanto credi in me."

"A mio fratello Michele, con cui, da piccoli, sembrava che l'unico modo di comunicare fosse prenderci a botte senza un motivo apparente. Forse era il nostro modo un po' strano di dirci che ci volevamo bene, anche se allora non lo capivamo. Crescendo, però, ho capito quanto siamo importanti l'uno per l'altro e quanto possiamo sempre contare su questo legame, che è diventato più forte di qualsiasi scazzottata. Grazie per essere sempre stato il mio complice, il mio rivale e il mio migliore alleato. Nonostante tutto, so che ci vogliamo bene, anche se non ce lo diciamo troppo spesso."

"Un ringraziamento speciale va a mia nonna Chiara, la mia seconda mamma. Sei stata tu a crescermi e ad accudirmi quando mamma e papà erano a lavoro, prendendoti cura di me con amore e pazienza ogni giorno. Sei stata sempre presente per me, il mio punto di riferimento nei momenti più importanti della mia infanzia. So quanto mi vuoi bene, anche se non lo dimostri apertamente, e ogni volta che torno a casa i tuoi occhi brillano di felicità, ricordandomi quanto sono fortunato ad averti accanto. Per me è importantissimo vederti felice, perché la tua felicità è anche la mia. Ogni volta che

mi chiami, sento la tua gioia e il tuo amore sincero, e questo mi riempie il cuore. Grazie di cuore per tutto ciò che hai fatto e continui a fare per me, nonna. Sei unica e speciale, e ti voglio un bene immenso."

"A mia nonna Giuliana, un pilastro della mia vita. Non dimenticherò mai tutto quello che abbiamo condiviso: quando mi venivi a prendere con la tua Seicento blu, quando cucinavi quelle magnifiche patate al forno che solo tu sapevi fare, e quando mi accompagnavi ovunque desiderassi andare. Ogni volta che tornavo a casa, venivo subito a trovarti, e mi raccontavi tutto ciò che accadeva con quella tua semplicità che mi faceva sentire al sicuro. Non eri una persona molto chiacchierona, ma con te bastava il silenzio per sentirmi protetto. Ora purtroppo non ci sei più e non hai potuto vedere il tuo nipotino arrivare a questo traguardo, ma spero con tutto il cuore che tu sia fiera di me, perché tutto ciò che sono lo devo anche a te. Grazie, nonna, per essere stata così speciale."

"A mio zio Massimiliano, che ha acceso in me la passione per l'informatica fin da quando ero piccolo. Sei stato tu a regalarmi il mio primo computer e a condividere con me l'esperienza di aprire insieme un vecchio computer, facendomi scoprire un mondo affascinante. Ti ringrazio di cuore per avermi avvicinato a questo universo che oggi è una parte fondamentale della mia vita."

"A mia zia Monica: essere il tuo primo nipotino è stato un dono, perché mi hai sempre trattato come un figlio. Mi hai dato amore, sostegno e una presenza che ha significato tanto per me. Non dimenticherò mai tutto quello che hai fatto per me, e quanto il tuo affetto abbia reso più bello il mio cammino. Grazie di cuore per esserci sempre stata, con tutto il tuo amore"

"Un ringraziamento dal profondo del cuore va alla mia migliore amica Sara. Sei stata al mio fianco fin da quando eravamo piccoli, condividendo con me gioie, momenti difficili e qualche piccolo inciampo lungo il cammino. Nonostante tutto, siamo sempre riusciti a riprenderci e a rendere il nostro legame ancora più forte. Sei stata l'unica che mi ha sostenuto davvero, quella che ha sempre creduto in me, anche nei momenti in cui io stesso non ci riuscivo. Non riesco a trovare le parole giuste per dirti quanto ti devo e quanto sei importante per me. Il tuo supporto, il tuo affetto, e la tua presenza hanno significato tutto. Spero con tutto il cuore che questo legame che ci unisce duri per sempre, perché non riesco nemmeno a immaginare la mia vita senza di te. Sei una parte di me, una certezza che non voglio mai perdere. Grazie, Sara, per tutto ciò che sei e per tutto ciò che hai fatto per me."

"Un ringraziamento speciale va ai miei coinquilini Andrea e Samuel. Con voi questi anni sono letteralmente volati! Condividere questa esperienza con voi è stato qualcosa di speciale, pieno di momenti che porterò sempre nel cuore. Le avventure e le storie che abbiamo vissuto insieme dentro quella casa sono uniche, e solo noi possiamo davvero capire quanto siano state divertenti e significative. Grazie per il vostro supporto, per esserci stati nei momenti seri e anche in quelli in cui non si poteva far altro che ridere. Siete riusciti a farmi sentire davvero a casa, anche quando ero lontano dalla mia. La vostra compagnia ha reso questi anni indimenticabili, e non posso che dirvi grazie di cuore per tutto. Mi mancheranno le risate, le chiacchierate e, sì, anche le piccole follie. Siete stati più che coinquilini, siete stati una famiglia."

"Un grande grazie va ai miei amici Mattia, Tullio e Alessandro. Da più di dieci anni siamo uniti, e in tutto questo tempo il vostro supporto è stato fondamentale per me. Avete sempre creduto in me, anche nei momenti in cui io stesso facevo fatica, e questo ha fatto la differenza. La vostra amicizia è speciale, e sono grato per tutte le risate, le chiacchierate e il sostegno che mi avete dato lungo questo percorso. Grazie di cuore per esserci sempre stati."

"Un ringraziamento va al mio amico Filippo, l'unica persona con cui ho davvero condiviso questo percorso accademico. Siamo molto simili, e spesso ci capiamo al volo, anche nei momenti più

complessi. Grazie per la tua sincerità e la complicità che mi hanno aiutato tanto lungo questo cammino. Condividere questa esperienza con te è stato davvero importante."