

Progetto di Programmazione Mobile

FitTracker

Alessandro Rongoni, Federico Pretini, Gregorio Vecchiola

Anno Accademico 2021/2022



Indice

1	Introduzione	3
2	Sintesi dell'approccio	3
3	Android (Kotlin)	4
	3.1 Requisiti	4
	3.2 Utilizzo in Android	6
	3.3 Architettura in Android	9
	3.4 Schermate in Android	9
	3.5 Sviluppo	18
	3.6 Testing	29
4	Flutter (Dart)	34
	4.1 Requisiti	34
	4.2 Utilizzo in Flutter	36
	4.3 Architettura in Flutter	38
	4.4 Schermate in Flutter	38
	4.5 Sviluppo	43
5	Conclusioni, Known Bugs e Future Features	45
	5.1 Android	45
	5.2 Flutter	45

1 Introduzione

FitTracker è un'applicazione per cellulare che ha lo scopo di tenere traccia della dieta dell'utente e del suo benessere fisico e mentale.

Annota quello che mangi e raggiungi il peso che desideri. Segui i progressi verso i tuoi obiettivi nutrizionali, di fitness, peso e idratazione. L'app permette di monitorare completamente alimentazione e salute: è come avere un dietologo, un piano dietetico e un diario alimentare sempre con te.

FitTracker non è la solita app per diete restrittive. E' un'app che ti aiuterà a conoscere le tue abitudini, a capire come mangi, a fare scelte alimentari più sane, a trovare sostegno e motivazione ed a raggiungere tuoi obiettivi di benessere.

E' come avere un dietologo, un personal trainer e un coach nutrizionale sempre a portata di mano. Infatti non solo puoi registrare pasti, ma anche aggiungere esercizi fisici. Aggiungi anche i tuoi piatti preferiti e crea i tuoi alimenti, in modo di avere ancora di più il controllo su cosa mangi.



Figura 1: Logo dell'applicazione

2 Sintesi dell'approccio

L'approccio utilizzato per lo sviluppo dell'applicazione è partito dal testing di altre applicazioni simili a pagamento. Successivamente abbiamo raccolto le funzionalità più importanti che la nostra applicazione doveva svolgere, con l'idea che l'interfaccia dovesse essere il più user-friendly possibile.

3 Android (Kotlin)

3.1 Requisiti

I requisiti funzionali e non funzionali sono delle azioni/regole che la nostra applicazione dovrà rispettare.

Requisiti funzionali

REQUISITO	DESCRIZIONE
RF1 Login	L'applicazione dovrà gestire l'accesso
RF2 Logout	L'applicazione dovrà gestire la disconnessione
RF3 Registrazione	L'applicazione dovrà gestire la registrazione dell'utente
RF3 Modifica profilo	L'applicazione dovrà gestire la modifica dei dati personali dell'utente
RF4 Recupero password	L'applicazione dovrà gestire il recupero della password
RF5 Registrazione dei dati dell'utente	L'applicazione dovrà gestire il salvataggio dei dati dell'utente
RF6 Salvataggio acqua	L'applicazione dovrà gestire il conteggio dell'acqua bevuta giornalmente
RF7 Salvataggio calorie	L'applicazione dovrà gestire il conteggio delle calorie quotidiane
RF8 Salvataggio carboidrati	L'applicazione dovrà gestire il conteggio dei grammi di carboidrati
RF9 Salvataggio proteine	L'applicazione dovrà gestire il conteggio dei grammi di proteine
RF10 Salvataggio grassi	L'applicazione dovrà gestire il conteggio dei grammi di grassi
RF11 Ricerca alimenti	L'applicazione dovrà gestire la ricerca degli alimenti da salvare nel diario
RF12 Ricerca esercizi	L'applicazione dovrà gestire la ricerca degli esercizi da salvare nel diario
RF13 Ricerca barcode	L'applicazione dovrà gestire la ricerca dell'alimento tramite barcode
RF14 Salvataggio alimenti personali	L'applicazione dovrà gestire il salvataggio degli alimenti/pasti/ricette personali dell'utente
RF15 Salvataggio alimenti preferiti	L'applicazione dovrà gestire il salvataggio degli alimenti preferiti dell'utente
RF16 Salvataggio esercizi personali	L'applicazione dovrà gestire il salvataggio degli esercizi personali dell'utente
RF17 Salvataggio esercizi preferiti	L'applicazione dovrà gestire il salvataggio degli esercizi preferiti dell'utente
RF18 Aggiunta al diario	L'applicazione dovrà gestire il salvataggio nel diario delle calorie assunte e bruciate

REQUISITO	DESCRIZIONE
RF19 Modifica alimenti personali	L'applicazione dovrà gestire la modifica degli alimenti personali
RF20 Modifica esercizi personali	L'applicazione dovrà gestire la modifica degli esercizi personali
RF21 Rimozione degli alimenti selezionati	L'applicazione dovrà gestire la rimozione degli alimenti selezionati nel diario
RF22 Rimozione degli esercizi selezionati	L'applicazione dovrà gestire la rimozione degli esercizi selezionati nel diario
RF23 Rimozione degli esercizi preferiti	L'applicazione dovrà gestire la rimozione degli esercizi preferiti
RF24 Rimozione degli alimenti preferiti	L'applicazione dovrà gestire la rimozione degli alimenti preferiti
RF25 Statistiche filtrate	L'applicazione dovrà gestire le statistiche dei consumi dell'utente secondo degli intervalli di tempo
RF26 Selezionare dieta	L'applicazione dovrà gestire la selezione della dieta dell'utente e modificare i macro-nutrienti in base ad essa
RF27 Funzioni	L'applicazione dovrà gestire le funzioni utili all'utente all'interno dell'applicazione
RF28 Calcolo del fabbisogno	L'applicazione dovrà gestire e calcolare il fabbisogno calorico giornaliero dell'utente in base ai suoi dati.

Requisiti non funzionali

REQUISITO	DESCRIZIONE
RNF1 Kotlin	L'applicazione dovrà essere scritta completamente in Kotlin
RNF2 Intuitiva	L'applicazione dovrà essere il più intuitiva e semplice possibile
RNF3 Fluida	L'applicazione dovrà compiere azioni e poter navigare al suo interno in maniera fluida e senza crash
RNF4 Estetica	L'applicazione dovrà essere bella esteticamente
RNF4 UserFriendly	L'applicazione dovrà user friendly
RNF5 Privacy	L'applicazione dovrà rispettare la privacy dell'utente secondo le norme dell'UE
RNF6 Permessi fotocamera	L'applicazione richiede l'accesso alla fotocamera
RNF7 Connessione internet	L'utente deve avere una connessione ad internet disponibile

3.2 Utilizzo in Android

Di seguito è riportato lo schema dei casi d'uso che riassume le principali funzionalità dell'applicazione. **N.B.** L'applicazione necessita di connessione ad internet.



Figura 2: Casi d'uso dell'applicazione

Accesso e Registrazione

Come si può notare, all'inizio l'utente può svolgere due azioni: effettuare l'accesso cliccando sul pulsante "Accedi", oppure avviare la registrazione di un nuovo utente cliccando il pulsante "Inizia"

Avviando una nuova registrazione, appariranno una serie di schermate dove l'utente potrà inserire i suoi dati personali, utili per calcolare il suo fabbisogno calorico giornaliero. Infine dovrà inserire la sua email e la sua password che serviranno per l'accesso e il mantenimento dei dati sul cloud.

Qualora l'utente fosse già registrato, cliccando il pulsante "Accedi" potrà inserire la sua email e password per effettuare l'accesso, oppure recuperare la password tramite l'invio di un link di reset tramite l'email indicata.

Schermata Home

Una volta effettuato l'accesso, l'utente si troverà nella schermata principale dell'applicazione: la Home. La Home è suddivisa in quattro sezioni: Diario, Statistiche, Diete e Funzionalità. Sempre nella schermata di Home, è presente una toolbar, con un menù a tendina, contenente due voci: Profilo e Logout.

Cliccando sul Profilo, verrà aperta una schermata con tutti i dati dell'utente, i quali possono essere modificati per cambiare il valore del fabbisogno calorico giornaliero. Cliccando su Logout, invece, si effettuerà la disconnessione dall'account.

Schermata Home: Diario

Appena aperta l'applicazione, l'utente si troverà nella sezione Diario, nella quale potrà visualizzare il suo fabbisogno calorico giornaliero, con le calorie rimanenti, assunte e bruciate. Inoltre troverà anche delle barre di progresso le quali mostrano i grammi di macro nutrienti assunti e quelli da assumere ancora. Questi dati sono stati generati sulla base dei dati dell'utente inseriti nella registrazione e in base alla dieta che l'utente ha selezionato. Di base l'utente partirà da una dieta Mediterranea.

Sempre nella sezione del Diario, l'utente può andare a registrare e visualizzare i pasti (e l'attività fisica) consumati nel corso della giornata ed i bicchieri d'acqua bevuti (ogni bicchiere conta 250 ml di acqua, per un totale di 2 litri d'acqua al giorno). Cliccando sul pulsante dei pasti o sul pulsante esercizio, si aprirà una nuova scheda composta da tre schermate: Ricerca, Personalizzati e Preferiti, mentre la lunga pressione di questi pulsanti aprirà una schermata pop-up per mostrare i prodotti/esercizi già salvati. E' importante specificare che per avere una maggiore pulizia nella lista dei pasti/esercizi già selezionati, l'utente, qualora aggiungesse un pasto/esercizio già selezionato precedentemente, esso verrà sovrascritto. Per questo abbiamo dato la possibilità all'utente di andare a modificare, oltre che ad eliminare, i pasti/esercizi che sono già stati salvati nel diario.

Ricerca

Nella sezione Ricerca è possibile andare a cercare i pasti/esercizi tramite barra di ricerca, oppure scansionando il codice a barre del prodotto cliccando sul pulsante in alto a destra. Nel caso in cui si ricerchi il prodotto/esercizio per nome all'interno della barra di ricerca, cliccandolo si aprirà una schermata con tutti i valori nutrizionali dello stesso. Tra questi troviamo le calorie, i grammi di carboidrati, i grammi di grassi e i grammi di proteine. Le informazioni nutrizionali sono calcolate su 100 gr. di prodotto. Inoltre possiamo trovare delle informazioni aggiuntive per quanto riguarda il prodotto, come il brand, la categoria a cui appartiene e la descrizione. In fondo alla schermata è possibile inserire la quantità di prodotto da aggiungere.

N.B. In quanto questi alimenti sono presi da un database online, potrebbe essere possibile che non ci siano tutti i prodotti oppure che non abbiano le informazioni aggiuntive o immagini illustrate.

N.B. Per selezionare una quantità di grammi non multipla di 100, basta mettere la quantità come numero decimale.

Una volta selezionata la quantità, è possibile aggiungere il prodotto al diario, oppure salvarlo tra i preferiti.

N.B. La ricerca degli esercizi deve essere fatta in inglese.

Personalizzati

In questa sezione è possibile aggiungere un pasto rapido personalizzato/ricetta/alimento. Infatti, cliccando sul pulsante "+" in alto, si aprirà una schermata contenente i campi da compilare con le informazioni relative ad esso. Ciò consente all'utente di avere un database infinito di prodotti, infatti verranno salvati all'interno del cloud, permettendo all'utente di poter cambiare tranquillamente dispositivo, senza perdere i propri prodotti.

Una volta aggiunto il prodotto personalizzato, esso verrà visualizzato nella lista all'interno di personalizzati. Cliccandoci sopra sarà poi possibile compiere diverse azioni, tra cui: modificare le informazioni relative, aggiungerlo al diario (la quantità verrà scelta dall'utente e non è più basato su 100 gr. di prodotto) ed eliminarlo dalla lista.

Preferiti

All'interno di questa sezione vengono visualizzati i prodotti salvati come preferiti. Anch'essi vengono salvati sul cloud. Selezionando un prodotto preferito si potranno compiere, anche qui, diverse azioni, tra cui: aggiungere il pasto al diario oppure rimuoverlo dalla lista dei preferiti.

Schermata Home: Statistiche

La seconda sezione che troviamo nella Home sono le Statistiche. All'interno di questa schermata, l'utente può verificare diverse categorie di statistiche, come ad esempio le calorie consumate, i litri d'acqua bevuti, i grammi di carboidrati, proteine e grassi consumati ed i giorni in cui l'utente ha raggiunto l'obiettivo di 2 litri di acqua bevuti.

Si possono effettuare due tipi di filtraggio: inserendo un intervallo di date, ovvero una data di inizio e una data di fine, oppure, lasciando vuoti tali spazi, è possibile avere un resoconto totale da quando l'utente si è registrato.

Schermata Home: Dieta

Nella sezione Dieta, l'utente può selezionare una qualsiasi dieta secondo le sue esigenze. Cliccando su una di esse è possibile andare a vedere nel dettaglio cosa propone. Infatti ognuna di esse ha diverse percentuali di macro nutrienti, le quali verranno utilizzate per ricalcolare i grammi di quest'ultimi in base al fabbisogno calorico dell'utente.

Tra le diete che abbiamo inserito troviamo: la dieta "Climatica" (50% carb. , 20% proteine, 30 % grassi), la dieta "Keto leggera" (19% carb. , 15% proteine, 66 % grassi), la "Keto media" (9% carb. , 15% proteine, 76% grassi), la dieta "Mangiare pulito" (40% carb. , 25% proteine, 35 % grassi), la "Mediterranea" (40% carb. , 20% proteine, 40 % grassi) e la dieta "Scandinava" (40% carb. , 30% proteine, 30 % grassi). La dieta attualmente selezionata sarà segnata da un contorno rosso attorno alla card della stessa.

Schermata Home: Funzioni

Questa sezione è dedicata a delle funzioni inerenti al fitness ed al benessere fisico. Attualmente sono sotto implementazione in quanto dovranno essere destinate al lavoro di tesi di Rongoni Alessandro, con lo sviluppo di un'intelligenza artificiale. Si attendono nuove versioni.

3.3 Architettura in Android

Per lo sviluppo di questa versione dell'applicazione, abbiamo utilizzato il linguaggio nativo Kotlin tramite l'IDE IntelliJ.

Per quanto riguarda l'aspetto tecnico dell'applicazione, abbiamo utilizzato l'architettura MVVM. Non abbiamo implementato nessun database locale, infatti abbiamo optato per l'utilizzo di un database remoto, utilizzando Firebase. Firebase è usato sia per l'autenticazione che per lo storage dei dati persistenti dell'utente (FirebaseAuth e FirebaseStore).

Per la ricerca dei prodotti e degli esercizi abbiamo utilizzato due API diverse, rispettivamente Edamam e Ninjas.

3.4 Schermate in Android

Mockup FitTracker v1.0

Qui di seguito sono riportati i mockup della vecchia versione dell'applicazione. Per comodità sono stati divisi in diverse parti.

N.B. le schermate superflue come ad esempio per la creazione di pasti personalizzati, ricette, messaggi e schermate ripetute o di importanza poco rilevante, sono state omesse per praticità.

N.B. essendo dei mockup alcune schermate non saranno identiche al 100%, ma servono solamente come linee guida per la progettazione delle schermate definitive.

Login e Registrazione

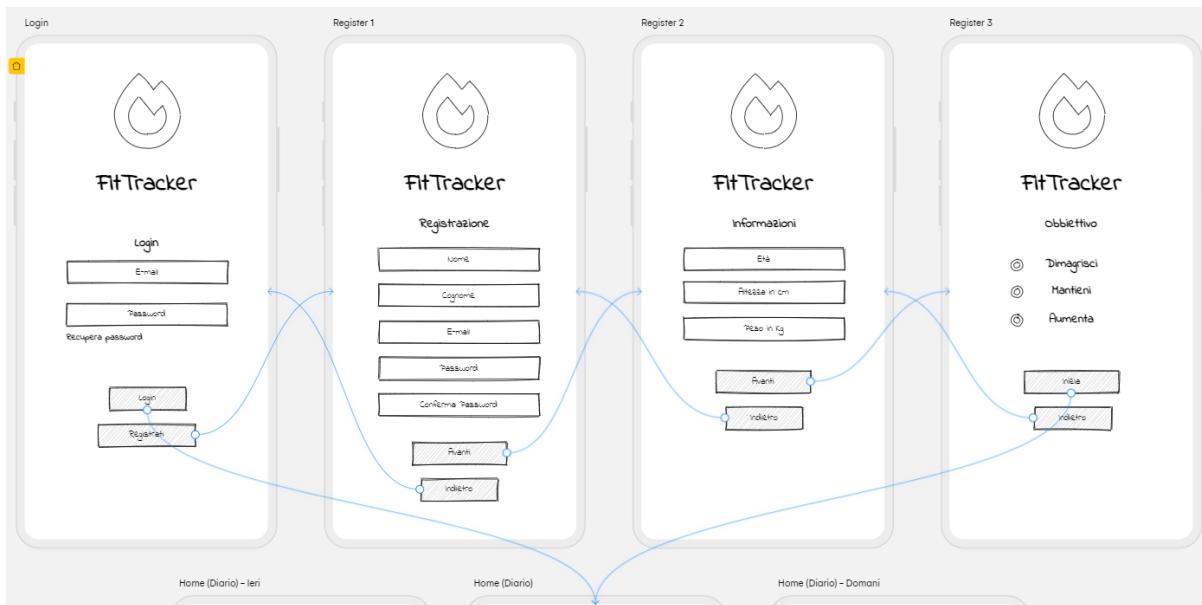


Figura 3: Dallo screen **Login** cliccando il pulsante "Login" accediamo alla **Home (Diario)**, mentre cliccando il pulsante "Registrati" verrà aperto lo screen **Register 1**, da cui poi partirà tutta la fase di registrazione. In queste schermate verranno aggiunte tutte le informazioni dell'utente. Inserite quest'ultime si verrà reindirizzati allo screen **Home (Diario)**.

Home (Diario)

La schermata **Home (Diario)** è la nostra schermata principale. Nella schermata stessa è possibile andare ad aggiungere i pasti della giornata, gli esercizi svolti e l'acqua assunta. Mentre

in cima è presente un resoconto delle calorie rimanenti da assumere, quelle assunte e quelle bruciate in allenamento. E' possibile andare a visualizzare le schede relative alle giornate precedenti ed a quelle future grazie alle frecce di navigazione.

In fondo alla schermata abbiamo un **Bottom Navigation**, il quale ci permette di navigare all'interno delle diverse schermate dell'applicazione.

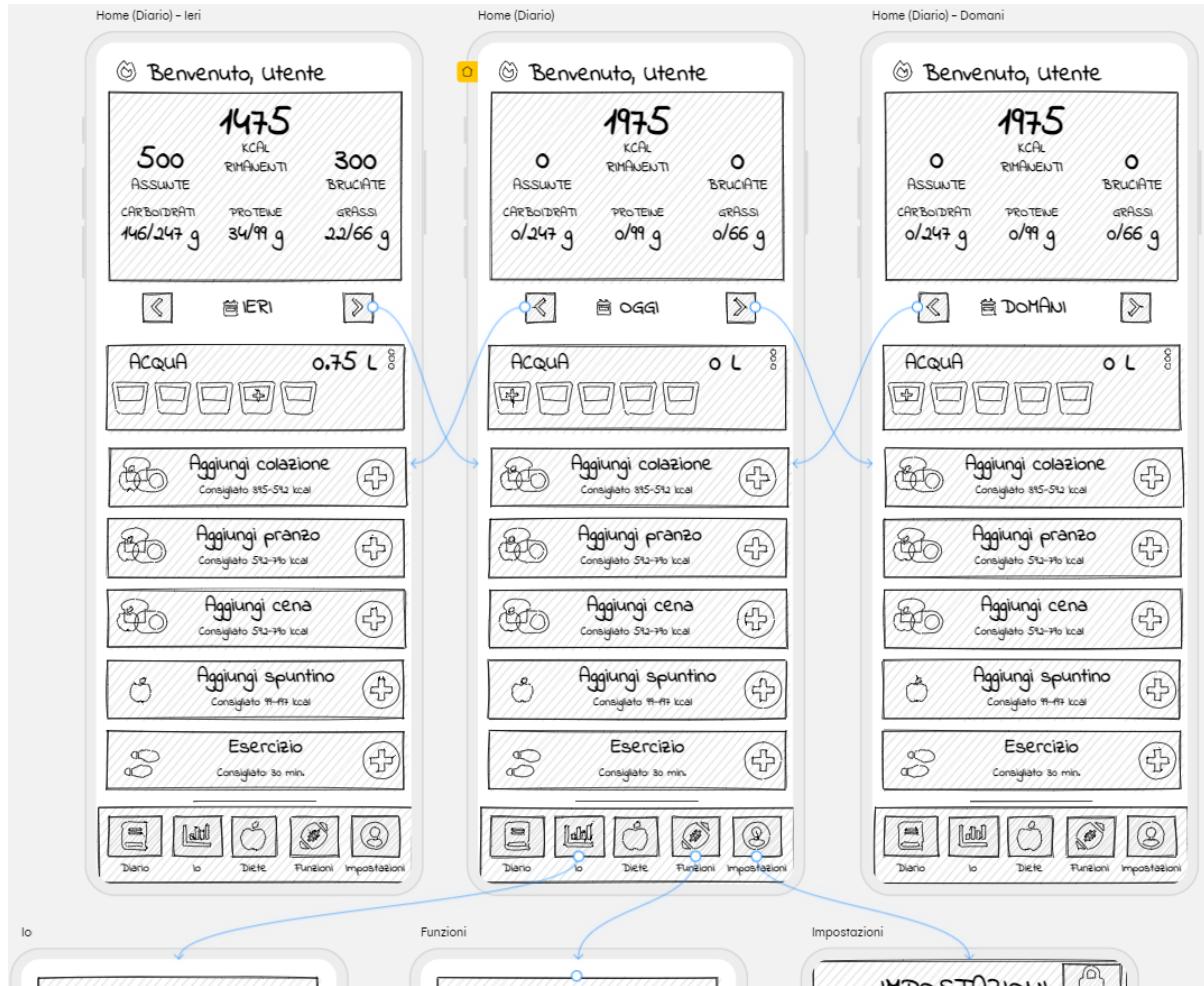


Figura 4: Cliccando sul "+" accanto ad ogni card è possibile andare ad aggiungere un alimento al pasto. Lo stesso vale con i bicchieri d'acqua. Aggiungendo degli esercizi, le calorie bruciate saranno tenute in considerazione nel resoconto delle calorie rimanenti.

Bottom Navigation

Scegliendo uno degli elementi della Bottom Navigation possiamo accedere a sezioni diverse dell'applicazioni, ognuna con una diversa funzionalità. Da esse poi si potrà accedere ancora ad altre sotto-interfacce.

Sezione "IO"

Prendiamo come esempio la sezione "IO", la quale si divide in "Corpo", "Statistiche" e "Prefe-riti". Ognuna di queste card porta ad una interfaccia diversa, con una sua funzionalità specifica.

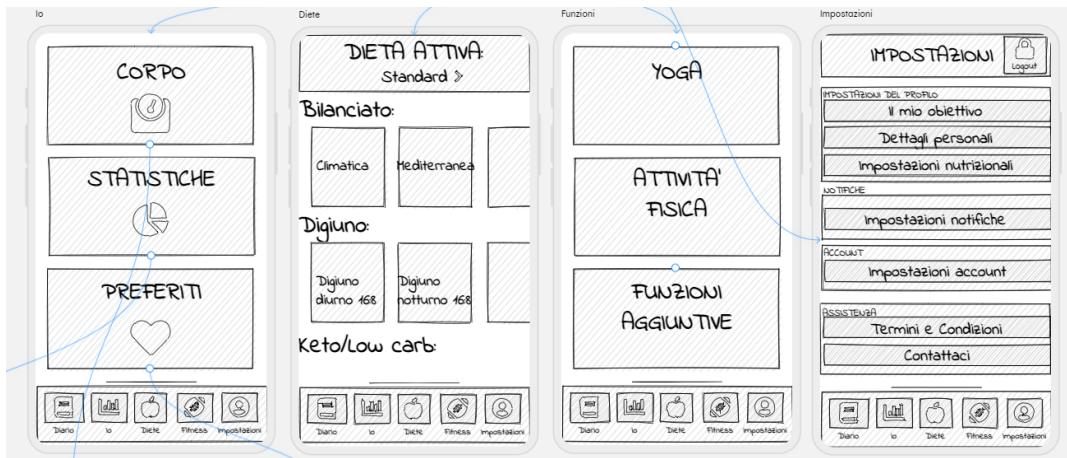


Figura 5: Le quattro schermate con la visualizzazione delle sotto viste

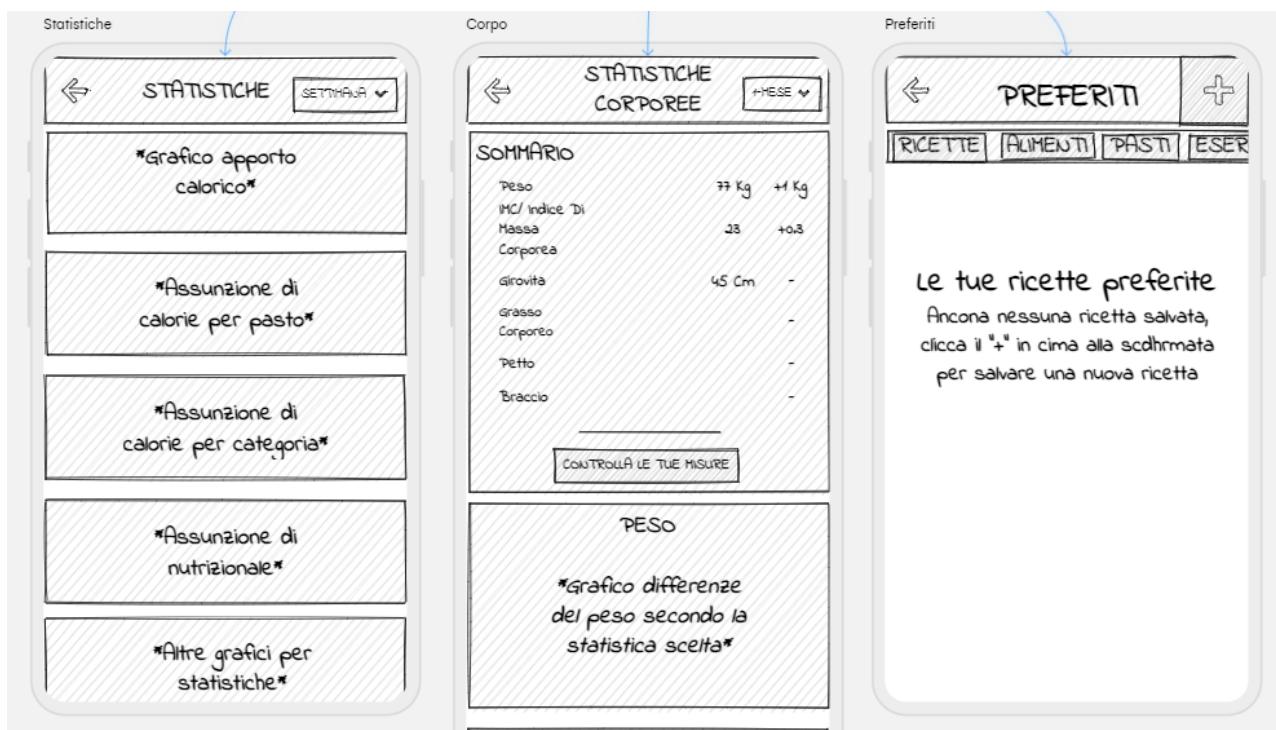


Figura 6: Sezione Io

Aggiunta di un alimento ad un pasto

Cliccando il "+" accanto ad un pasto si potrà avviare la ricerca dell'alimento tramite la barra di ricerca oppure scannerizzando il codice a barre del prodotto. E' possibile anche selezionare degli alimenti salvati tra i preferiti.

N.B. lo stesso ragionamento lo facciamo per gli esercizi.

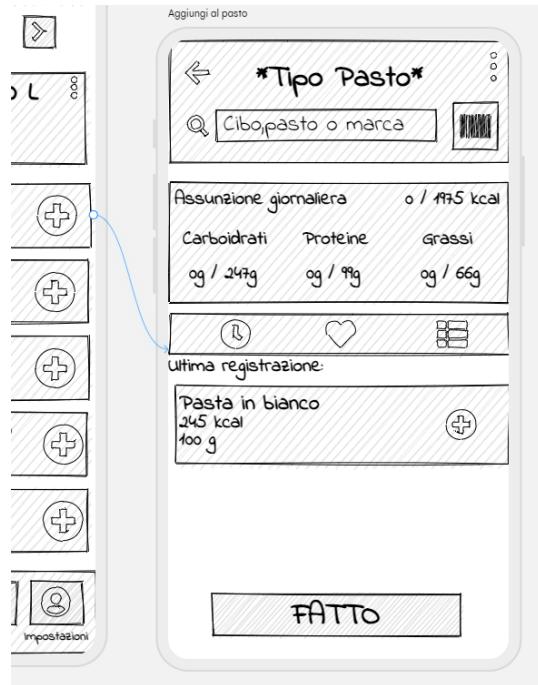


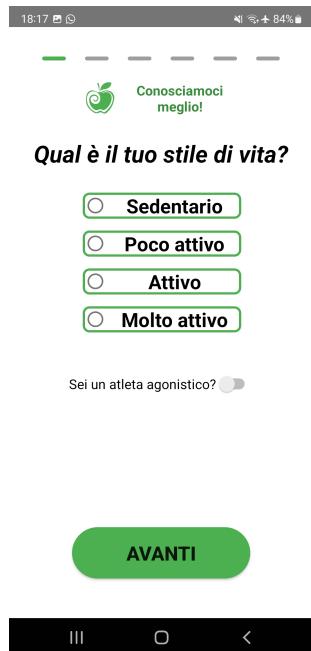
Figura 7: Sezione per aggiungere un pasto)

Mockup FitTracker v2.0

Di seguito mostriamo le schermate definitive della nostra applicazione sviluppata per dispositivi Android. Vengono mostrate tutte le Activity ed i Fragments in cui l'utente può navigare e con cui può interagire.



(1) InizioActivity



(2) StileDiVitaFragment

Uomo o Donna?

- Uomo
- Donna

AVANTI



(3) SessoFragment

Dati Personaliali

Data di nascita:

Nome
Cognome

AVANTI



(4) DatiPersonalialiFragment

La prima schermata con cui si va ad interagire è la InizioActivity, da qui possiamo prendere due strade: la prima ci porta allo StileDiVitaFragment, la seconda al LoginActivity, usato per l'accesso.

Con lo StileDiVitaFragment si inizia il percorso di registrazione, in cui l'utente va a specificare tutti i suoi dati personali utili per la creazione del diario. Infatti il calcolo delle calorie gior-

naliere dipende, oltre dall'altezza, il peso e il sesso, dal livello di attività fisica che si svolge settimanalmente.

Ovviamente i campi sono tutti obbligatori. La data è controllata e l'età minima per registrarsi è di 15 anni. L'altezza inserita e il peso devono essere compresi rispettivamente tra i 130-300 cm e 30-200 Kg. Qualora uno degli input fosse sbagliato, sarà impossibile andare avanti.

The screenshot shows two mobile application screens for height and weight entry. Both screens have a header with the time (18:18), signal strength, battery level (84%), and a back arrow icon. Below the header is a dashed green bar.

Qual è la tua altezza?

130-300 cm

Qual è il tuo peso attuale?

30-200 Kg

Both screens feature green "AVANTI" (Next) buttons at the bottom.

(5) AltezzaActivity

(6) PesoActivity

The screenshot shows two mobile application screens. The left screen asks "Quale sport pratichi?" and lists various sports with radio buttons. The right screen shows an apple icon and registration options.

Quale sport pratichi?

- Calcio
- Basket
- Pallavolo
- Nuoto
- Baseball
- Karate
- Pattinaggio
- Cricket
- Tennis

OK! Siamo arrivati all'ultimo passaggio

Email

Password

Conferma password

AVANTI

REGISTRATI

(7) SportActivity

(8) RegisterActivity

La schermata SportActivity è opzionale, in quanto viene aggiunta solamente se l'utente specifica di essere un atleta agonista in StileDiVitaActivity. La scelta dello sport non comporta alcuna modifica nel calcolo delle calorie, ma serve qualora di volesse fare una statistica sugli sport più praticati dagli utenti. Una volta completate tutte le informazioni, l'utente può registrarsi, inserendo la propria email e password.

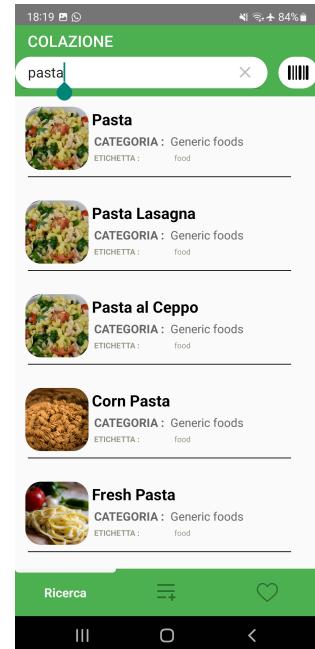
Avvenuta la corretta registrazione dell'utente, si potrà effettuare il login all'interno della LoginActivity. Inoltre, all'interno di essa, sarà possibile andare a recuperare la password qualora si fosse dimenticata. Il recupero password avviene tramite link inviato all'email specificata. Effettuato l'accesso, viene aperta la HomeActivity, nella sezione DiarioFragment, il quale contiene tutte le informazioni relative ai pasti (calorie, macro nutrienti), esercizi e acqua del giorno corrente.



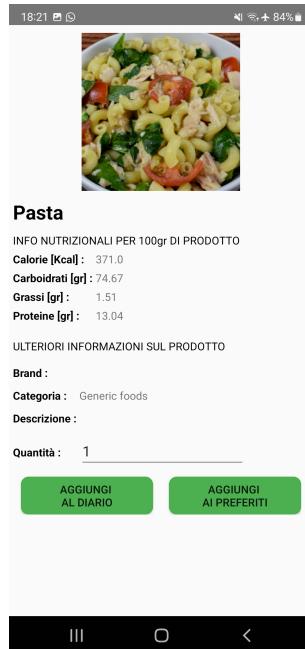
(9) LoginActivity



(10) DiarioFragment



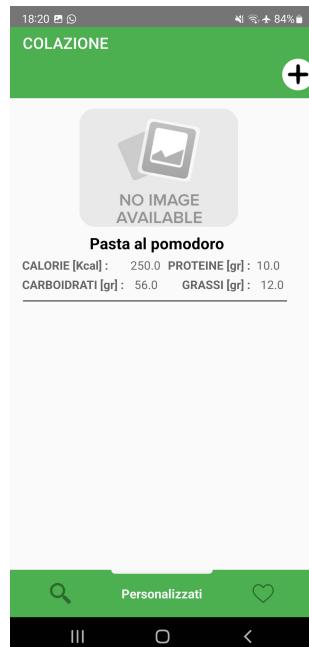
(11) RicercaFragment



(12) Selezione item ricerca

Cliccando su uno dei pulsanti relativi ai pasti o agli esercizi, è possibile andare a cercare (RicercaFragment) i pasti/esercizi da aggiungere al diario. Inoltre è possibile aggiungere propri pasti/esercizi alla lista personalizzati (PersonalizzatiFragment), oppure salvare dei prodotti/esercizi nella lista dei preferiti (PreferitiFragment).

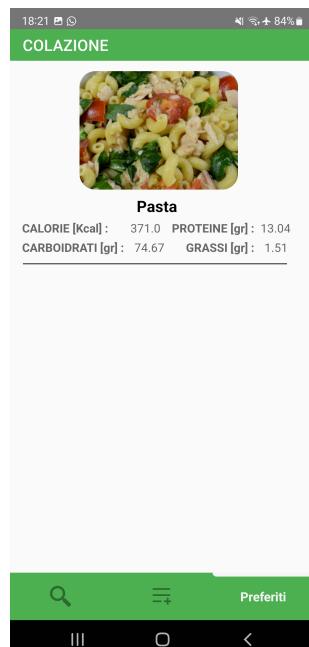
Cliccando su un elemento della lista, a seconda di dove ci si trova, si aprirà una finestra a pop-up per compiere diverse azioni con quell'item. Ad esempio cliccando un elemento nella lista dei Personalizzati è possibile andare a modificare il pasto personalizzato, aggiungerlo al diario, oppure eliminarlo. Lo stesso vale per gli elementi nella lista Preferiti.



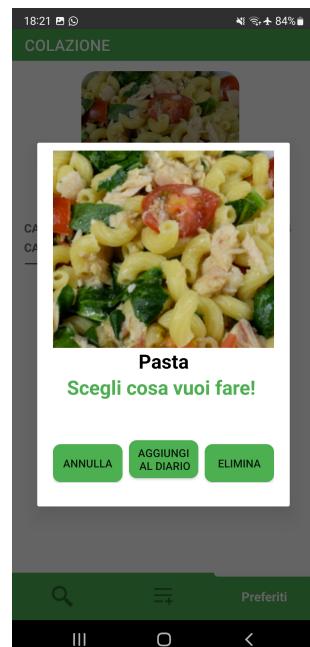
(13) PersonalizzazioneFragment



(14) Aggiunta item personale



(15) PreferitiFragment



(16) Selezione item preferito

Mentre, se invece di cliccare sul pulsante relativo al pasto/esercizio si effettua una pressione prolungata (long click) si aprirà la lista degli elementi selezionati, sui quali è possibile andare a modificare o a rimuovere dal diario, ripristinando così le calorie assunte.

Le altre tre schermate possibili della HomeActivity sono: StatisticheFragment, DieteFragment e FunzioniFragment. La schermata delle statistiche permette all'utente di controllare diversi indici in un intervallo di tempo fissato oppure in totale.

La sezione diete permette all'utente di cambiare la dieta, infatti ogni dieta ha delle diverse percentuali di macro nutrienti, di conseguenza cambieranno i grammi giornalieri di quest'ultimi. La dieta selezionata è evidenziata da una cornice rossa intorno alla card della stessa. Cliccando sopra una di esse si potrà andare a vedere nel dettaglio cosa consiglia la dieta.



(17) StatisticheFragment



(18) DieteFragment



(19) FunzioniFragment

FunzioniFragment è riservato a delle funzioni che verranno implementate nelle versioni future. Esso conterrà degli strumenti utili all'utente sempre nell'ambito del fitness e del benessere psico-fisico.

3.5 Sviluppo

Classi

Package: aggiungi_esercizio

All'interno di questo package troviamo tutte le classi che riguardano la parte di ricerca degli esercizi, salvataggio di esercizi personali e preferiti, e la visualizzazione delle informazioni sulle calorie bruciate.

AggiungiEsercizioActivity: questa è l'activity che mostra la ricerca degli esercizi, la lista degli esercizi personali e la lista di quelli preferiti. E' composta da una BottomNavigation che ci permette di selezionare le schemate di RicercaEserciziFragment, PersonalizzatiEsercizioFragment e PreferitiEserciziFragment. Di conseguenza va a gestire la selezione corretta della navigazione.

AggiungEserciziiViewModel: la classe ViewModel dell'AggiungiEsercizioActivity, si occupa del collegamento del DB con le viste. Tramite i metodi ottiene dal DB gli esercizi personalizzati e preferiti e li salva in una lista di tipo Esercizio. Inoltre recupera anche dal DB dell'API l'esercizio selezionato nella ricerca e impostata le calorie bruciate nella schemata del diario.

MyAdapterPrefPersEsercizio e **MyAdapterRicercaEsercizio:** sono gli adapter che si occupano degli item delle RecyclerView all'interno dei fragment dell'AggiungiEsercizioActivity.

PersonalizzatiEsercizioFragment, PreferitiEserciziFragment e **RicercaEserciziFragment:** sono i fragment collegati alla bottom navigation dell'AggiungiEsercizioActivity. All'interno di essi gestiamo le azioni da compiere relativi agli item cliccati e dell'impostazione dei layout. Inoltre nel RicercaEserciziFragment si gestisce anche il passaggio degli attributi da un'activity all'altra.

Package: aggiungi_pasto

All'interno di questo package troviamo tutte le classi che riguardano la parte di ricerca dei prodotti, salvataggio di pasti personali e preferiti, e la visualizzazione delle informazioni nutrizionali.

AggiungiActivity: questa è l'activity che mostra la ricerca dei pasti, la lista dei pasti personali e la lista dei preferiti. E' composta da una BottomNavigation che ci permette di selezionare le schemate di RicercaFragment, PersonalizzatiFragment e PreferitiFragment. Di conseguenza va a gestire la selezione corretta della navigazione.

AggiungiViewModel: la classe ViewModel dell'AggiungiActivity, si occupa del collegamento del DB con le viste. Tramite i metodi ottiene dal DB i pasti personalizzati e preferiti e li salva in una lista di tipo Pasto. Inoltre recupera anche dal DB dell'API il prodotto selezionato nella ricerca.

MyAdapterPrefPers e **MyAdapterRicerca:** sono gli adapter che si occupano delle RecyclerView all'interno dei fragment dell'AggiungiActivity.

PersonalizzatiFragment, PreferitiFragment e **RicercaFragment:** sono i fragment collegati alla bottom navigation dell'AggiungiActivity. All'interno di essi gestiamo le azioni da compiere relativi agli item cliccati o all'impostazione dei layout. Inoltre nel RicercaFragment si gestisce anche il passaggio degli attributi da un'activity all'altra.

Package: autenticazione

All'interno di questo package troviamo tutte le classi che riguardano la registrazione dei dati personali dell'utente, gestione dell'autenticazione e il recupero della password.

InizioActivity: questa è l'activity da dove tutto parte, a seconda del pulsante che si clicca, l'utente può registrarsi oppure effettuare l'accesso.

ConosciamociActivity: è l'activity che parte qualora l'utente scelga di registrarsi, partiranno una serie di fragments nei quali si potrà andare ad inserire tutti i dati personali.

RegisterActivity: è l'activity finale dopo che l'utente ha completato con successo il salvataggio dei dati. Questa activity gestisce la registrazione dell'utente e vede se l'email è già in uso oppure no.

LoginActivity: nel caso in cui l'utente sia già registrato, gli basterà effettuare l'accesso e questa è l'activity che gestisce tutto.

RecuperoActivity: è l'activity che gestisce il recupero della password dell'utente qualora se la fosse dimenticata.

AltezzaFragment, DatiPersonalniFragment, PesoAttualeFragment, SessoFragment, SportFragment, StileVitaFragment: sono i fragments che gestiscono l'input dei dati dell'utente e se li passano da fragment a fragment fino alla RegisterActivity, la quale si occupa di andarli a salvare definitivamente nel FirebaseDatabase.

Package: databaseFB

All'interno di questo package troviamo tutte le classi che riguardano il collegamento al FirebaseDatabase e il salvataggio degli esercizi, pasti, diari, personalizzazioni e preferiti per ogni utente, organizzati per date.

FirebaseDB: è una classe che gestisce l'istanza al FirebaseFirestore e si occupa anche di prendere l'email dell'utente attualmente autenticato, da passare poi ai path delle collection e dei document del db.

DiarioDB: è la classe che gestisce il settaggio del diario nel db (passandogli la data corrente) e l'acquisizione degli stessi dall'interno del database.

DietaDB: gestisce la restituzione delle diete dal database.

EsercizioDB: è la classe che gestisce il settaggio dell'esercizio nel db (passandogli la data corrente) e l'acquisizione degli stessi dal database.

PersonalizzatiDB: è la classe che gestisce il settaggio dei personalizzati nel db (passandogli il pasto) e l'acquisizione degli stessi dal database.

PreferitiDB: è la classe che gestisce il settaggio dei preferiti nel db (passandogli il pasto) e la restituzione degli stessi dal database.

ProdottoDB: è la classe che gestisce il settaggio del prodotto nel db (passandogli la data) e la restituzione degli stessi dal database.

UtenteDB è la classe che gestisce il settaggio dei dati dell'utente nel db (passandogli l'email) e la restituzione degli stessi dal database.

Package: diario

All'interno di questo package troviamo tutte le classi che riguardano il fragment principale dell'applicazione, in cui vengono visualizzate le calorie rimanenti, quelle consumate e quelle bruciate. Vengono, inoltre, visualizzati i pasti e gli esercizi con le relative calorie e l'acqua bevuta.

DiarioFragment: è la classe che contiene la creazione della view e il settaggio del diario dell’utente.

DiarioViewModel: è la classe che collega il database e la vista, andando a prendere i dati dell’utente per calcolare il fabbisogno calorico e i macro nutrienti.

MyAdapterEserciziSel e **MyAdapterSelezionati:** sono gli adapter che vanno a gestire gli item nelle RecyclerView per andare a modificare ed eliminare i pasti e gli esercizi già selezionati e salvati nel diario.

```
1 //Metodo per impostare i macro nutrimenti in base alla dieta e al fabbisogno  
2 //giornaliero  
3 private fun setMacro(){  
4     viewModelScope.launch {  
5         val utente = utenteDB.getUtente(auth.currentUser?.email!!)  
6         val dieta = dietaDB.getDieta(utente.dieta)  
7         _carboidratiMax.value = ((diario.value!!.fabbisogno*(dieta.  
8         perc_carb.toDouble()/100.0)) / 4).toInt() //1gr di carbo = 4Kcal  
9         _proteineMax.value = ((diario.value!!.fabbisogno*(dieta.  
10        perc_prot.toDouble()/100.0)) / 4).toInt() //1gr di prot = 4Kcal  
11         _grassiMax.value = ((diario.value!!.fabbisogno*(dieta.perc_prot.  
12       toDouble()/100.0)) / 9).toInt()//1gr di grassi = 9Kcal  
13         _diarioSettato.value = !(_diarioSettato.value)!!  
14     }  
15 }
```

```
1  
2  
3 //Metodo per calcolare il fabbisogno giornaliero in base ai dati dell’utente  
4 private fun calculateFabbisogno(utente: Utente) : Double{  
5     val today = LocalDate.now()  
6     val birthday: LocalDate = LocalDate.parse(utente.data_nascita)  
7     val period: Period = Period.between(birthday, today)  
8     if(utente.sesso == "Uomo")  
9         return ((66 + (13.7 * utente.peso_attuale) + (5 * utente.altezza  
10    ) - (6.8 * period.years)) * utente.LAF)  
11    else  
12        return ((65 + (9.6 * utente.peso_attuale) + (1.8 * utente.  
13        altezza) - (4.7 * period.years)) * utente.LAF)
```

Package: diete

All’interno di questo package troviamo tutte le classi che riguardano il fragment delle diete che l’utente può selezionare.

DieteFragment: è la view che contiene la RecyclerView contenente le diete a griglia.

DieteViewModel: è la classe che gestisce la selezione della dieta dal database e il cambio di dieta dell’utente.

MyAdapterDiete: è l’adapter che gestisce la RecyclerView e la disposizione delle diete al suo interno.

Package: home

All’interno di questo package troviamo tutte le classi che riguardano la Home, la schermata che racchiude tutto.

HomeActivity: è l'activity che gestisce la navigazione all'interno dell'applicazione e tra le schermate principali. Inoltre gestisce la toolbar per andare a cambiare i dati dell'utente e il logout.

HomeViewModel: è la classe che gestisce la disconnessione dell'account.

Package: model

All'interno di questo package troviamo tutte le data class usate per i model degli oggetti Diario, Dieta, Pasto e Utente.

Package: pasto

All'interno di questo package troviamo tutte le classi usate per la gestione dei pasti già selezionati.

PastoActivity: è la classe che gestisce l'activity che mostra le informazioni del pasto selezionato dalla lista.

PastoViewModel: è la classe che gestisce le azioni da svolgere sul pasto che è stato già selezionato, per poi andare ad aggiornare le calorie e i macro nutrienti qualora si eliminasse o modifichesse.

Package: prodotto

All'interno di questo package troviamo tutte le classi usate per la gestione dei pasti.

ProdottoActivity: è la classe che gestisce l'activity che mostra le informazioni del prodotto ricercato nel database. Poi da qui si potrà scegliere se metterlo nel diario oppure tra i preferiti.

ProdottoViewModel: è la classe che gestisce il salvataggio delle informazioni sul pasto qualora si salvasse nel diario oppure il salvataggio nella lista dei preferiti. Collega, quindi, la nostra app al database, contenente le informazioni relative alle calorie consumate in quella data su quel relativo pasto.

Package: profilo

All'interno di questo package troviamo tutte le classi usate per la gestione del profilo.

ProfiloActivity: è la classe che gestisce l'activity che mostra le informazioni dell'utente, prende le informazioni dal database grazie al ViewModel e imposta le diverse liste con i valori salvati.

ProfiloViewModel: è la classe che collega l'activity ProfiloActivity al database e si occupa nel restituire ed aggiornare i dati dell'utente, il cambio di password e ricalcolare il fabbisogno in base all'altezza, peso, sesso ed età qualora l'utente li modifichesse.

Package: scanner

All'interno di questo package troviamo tutte le classi usate per la gestione del profilo.

ScannerActivity: è la classe che gestisce l'activity dell'apertura e dei permessi d'accesso della fotocamera del dispositivo e la ricezione di un codice a barre oppure un QR code.

Package: statistiche

All'interno di questo package troviamo tutte le classi usate per la gestione delle statistiche.

StatisticheFragment: è la classe che gestisce il fragment della schermata delle statistiche, l'impostazione dell'intervallo di tempo e le view che mostrano i risultati.

StatisticheViewModel: è la classe che gestisce la filtrazione dei dati nel database secondo l'intervallo inserito e li restituisce.

```

1 private fun ottieniStatistiche(data_inizio: String, data_fine: String) {
2     viewModelScope.launch {
3         _statisticheLiveData.value = diarioDB.getStatistiche(data_inizio,
4             data_fine)
5         _kcalAssunte.value = 0
6         _grCarboidrati.value = 0
7         _grProteine.value = 0
8         _grGrassi.value = 0
9         _litriBevuti.value = 0.0
10        _giorniAcqua.value = 0
11        for (diario in _statisticheLiveData.value!!){
12            _kcalAssunte.value = _kcalAssunte.value!! + (diario.
13                chiloCalorieCena + diario.chiloCalorieColazione + diario.
14                chiloCaloriePranzo + diario.chiloCalorieSpuntino)
15            _grCarboidrati.value = _grCarboidrati.value!! + diario.
16                carboidratiTot
17            _grProteine.value = _grProteine.value!! + diario.proteineTot
18            _grGrassi.value = _grGrassi.value!! + diario.grassiTot
19            var acqua = 0
20            for(i in 0..7) {
21
22                if (diario.acqua[i]) {
23                    _litriBevuti.value = _litriBevuti.value!! + 0.25
24                    acqua +=1
25                }
26                if (acqua == 8) _giorniAcqua.value = _giorniAcqua.value
27                !! + 1
28            }
29        }
30        _getStatistiche.value = true
31    }
32 }
```

Package: esercizio

All'interno di questo package troviamo tutte le classi usate per la gestione e salvataggio degli esercizi.

EsercizioAddPrefActivity: è la classe che visualizza i dettagli dell'esercizio selezionato nelle ricerche. Contiene i pulsanti per aggiungere al diario e alla lista dei preferiti.

EsercizioRDUActivity: è la classe che permette di modificare, eliminare e leggere gli esercizi già selezionati.

EsercizioViewModel: è la classe che collega le interfacce e il database. Recupera gli esercizi svolti, esercizi preferiti e quelli personalizzati dal database di ciascun utente.

MainActivity

E' l'activiti che si apre all'avvio dell'applicazione e carica la schermata di InizioActivity.

Classi speciali

Package: retrofit

All'interno di questo package troviamo due interfacce e due classi per generare istanze retrofit. Le interfacce sono state utilizzate per dichiarare i metodi per eseguire le chiamate all'API. Ne troviamo due perché una è per l'API di Edamam (ApiInterface), che permette di fare le chiamate per ottenere i cibi, mentre la seconda interfaccia (ApiEserciziInterface) serve per fare le chiamate all'API Ninjas, quella responsabile degli esercizi.

Call è un metodo di Retrofit che consente di mandare una richiesta ad un webserver ed ottenere una risposta del tipo indicato tra parentesi angolari. La risposta dell'API sarà un Json che si va a mappare in classi kotlin grazie all'utilizzo di Moshi. Le classi utilizzate per tale scopo sono raccolte nella cartella model/json_parsing.

Le classi Instance sono delle classi che hanno un attributo che implementa una delle due interfacce a seconda dell'API a cui ci si deve collegare. Ad esempio RetrofitEserciziInstance è un object che ha come attributi api_esercizi, che implementa ApiEserciziInterface, questo permette di accedere al metodo GetEsercizi, che è il metodo vero e proprio per eseguire il GET dei dati.

```
1 interface ApiInterface {
2     //Call      un metodo di Retrofit che consente di mandare una richiesta ad
3     //un webserver ed ottenere una risposta.
4     @GET("/api/food-database/v2/parser")
5     fun getFoodFromNameOrUPC(@Query("app_id") idApp : String,
6                             @Query("app_key") keyApp : String,
7                             @Query("ingr") ingredient : String,
8                             @Query("upc") upc : String): Call<Json_FoodList
9 }
```

```
1 interface ApiEserciziInterface {
2
3
4     @GET("/v1/caloriesburned")
5     fun getEsercizi(@Header("X-Api-Key") key : String,
6                     @Query("activity") activity : String): Call<List<
7             Esercizio>>
8 }
```

Package: utils

Contiene la classe APICredentials la quale contiene gli URL utilizzati per le chiamate all'API e le API keys.

Parti di codice interessanti

```
1 /* Questa serie di funzioni sono quelle che permettono di gestire l'  
2 animazione dei bicchieri all'interno del diario  
3 La prima funzione ovvero startAnimation quella che fa partire l'  
4 animazione del riempimento del bicchiere  
5 La seconda funzione quella che controlla che controlla i bicchieri che  
6 devono essere riempiti al momento dell'apertura  
7 della pagina del diario e fa partire l'animazione. Se i bicchieri riempiti  
8 sono ad esempio i alla fine setta l' i-esimo + 1  
9 con l'immagine del bicchiere vuoto con il + dentro ad indicare che pu  
10 essere cliccato per aumentare la quantit di acqua bevuta  
11 La terza funzione invece ovvero onClickGlass il 'cuore' del funzionamento  
12 dinamico dei bicchieri  
13 essa associa ad ogni immagine del bicchiere un listener che fa la seguente  
14 operazione  
15 -Se l'i-esimo bicchiere pieno e viene cliccato allora esso dovr  
16 diventare vuoto con un + al suo interno  
17 e tutti i bicchieri dopo di lui dovranno invece essere settati con l'  
18 immagine del bicchiere vuoto  
19 -Se invece l'i-esimo bicchiere vuoto e viene cliccato allora il bicchiere  
20 dopo di lui dovr essere settato con il simbolo +  
21 mentre tutti quelli prima di lui dovranno fare lo start dell'animazione  
22 riempendosi  
23  
24  
25 La quarta funzione invece quella che si occupa di andare ad aggiornare l'  
26 etichetta in cui andiamo  
27 a visualizzare la quantit totale di acqua bevuta e nel caso di  
28 raggiungimento di una quantit pari  
29 a due litri verr visualizzato un messaggio di congratulazioni  
30 */  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42
```

```
19 private fun startAnimation(glass : ImageView){  
20     glass.setBackgroundResource(R.drawable.filling_animation)  
21     val frameAnimation: AnimationDrawable = glass.background as  
22     AnimationDrawable  
23     frameAnimation.start()  
24 }  
25  
26 fun checkFullGlasses(){  
27     for(i in 0..7) {  
28         if (model.diario.value!![i]) {  
29             acqua[i] = model.diario.value!![i]  
30             startAnimation(glasses[i])  
31             if(i < 7)  
32                 glasses[i+1].setBackgroundResource(R.drawable.  
33                 empty_glass_plus)  
34             }  
35         checkAcquaBevuta()  
36     }  
37  
38     private fun onClickGlass(){  
39         for(i in 0..7){  
40             glasses[i].setOnClickListener {  
41                 if(acqua[i]){  
42                     glasses[i].setBackgroundResource(R.drawable.  
43                     empty_glass_plus)  
44                     for(x in i..7){  
45                         if(glasses[x].background == R.drawable.  
46                         empty_glass_plus)  
47                             glasses[x].setBackgroundResource(R.drawable.  
48                             plus_glass)  
49                         else  
50                             glasses[x].setBackgroundResource(R.drawable.  
51                             empty_glass_plus)  
52                     }  
53                 }  
54             }  
55         }  
56     }  
57 }
```

```

43             if(x<7)
44                 glasses[x+1].setBackgroundColor(R.drawable.
empty_glass)
45                     acqua[x] = false
46             }
47         }else {
48             for(x in 0..i){
49                 startAnimation(glasses[x])
50                 acqua[x] = true
51             }
52             if(i < 7){
53                 glasses[i+1].setBackgroundColor(R.drawable.
empty_glass_plus)
54             }
55         }
56         checkAcquaBevuta()
57     }
58 }
59
60
61     private fun checkAcquaBevuta(){
62         var litri_acqua = 0.0
63         for (bicchiere in acqua)
64             if(bicchiere)
65                 litri_acqua += 0.25
66         model.setAcqua(litri_acqua)
67         /*Per fare in modo che il messaggio di congratulazioni venga
mostrato una sola volta nel caso di raggiungimento
dell'obiettivo controllo che il totale sia a 2 litri, se ho già
mostrato il messaggio nella stessa ciclo di vita del fragment
e se il settimo bicchiere d'acqua era già presente al momento del
caricamento del diario
*/
68         if(litri_acqua == 2.0){
69             binding.imageViewGoldMedal.isVisible = true
70             if(!flag_congratulazioni && !model.diario.value!![7]) {
71                 flag_congratulazioni = true
72                 openCongratulazioni()
73             }
74         }else{
75             binding.imageViewGoldMedal.isVisible = false
76         }
77     }
78 }
79
80
81 }

1 /* Questa classe stata creata per poter andare a cambiare in maniera
semplici i parametri di collegamento
2 all'API in modo da non dover andare a ricerca tra le varie classi i punti
esatti in cui rimpiazzare questi dati */

3
4 class APICredentials {
5     companion object {
6         @JvmStatic
7         val BASE_URL: String = "https://api.edamam.com"
8         @JvmStatic
9         val API_ID: String = "a4e62224"
10        @JvmStatic
11        val API_KEY: String = "2b0adbda47d1293d35d373d9f2ffce1"
12        @JvmStatic
13        val BASE_URL_ESERCIZI: String = "https://api.api-ninjas.com"

```

```

14     @JvmStatic
15     val API_KEY_ESERCIZI: String = "WF3P9samgfmtcdONl0pt2w=="
16     OaTsIyDXQC9EvQKf"
17 }

```

1 /*Classe che rappresenta l'utente a cui abbiamo fatto estendere l'
 interfaccia Parcelable in
2 maniera tale da poter passare un suo oggetto da un fragment ad un altro,
 questa soluzione ci ha
3 fatto molto comodo soprattutto in fase di registrazione dove ci saremmo
 trovati a dover passare un numero
4 sempre pi elevato di parametri da un fragment all'altro, in quanto la
 registrazione effettiva dell'utente
5 viene fatta solamente al termine della navigazione quando tutti i dati dell'
 utente sono disponibili.
6 Con questa soluzione invece abbiamo passato sempre e solo un oggetto di tipo
 Utente che andavamo piano
7 piano a valorizzare. Ogni utente viene registrato con una dieta di default
 che la cliamatica, ovvero
8 la dieta che ogni persona dovrebbe seguire per uno stile di vita sano.
 Abbiamo comunque lasciato la
9 possibilità all'utente di modifica la propria dieta nell'apposito
 fragment Diete */

10
11 @Parcelize
12 data class Utente(
13 var nome: String,
14 var cognome: String,
15 var email: String,
16 var LAF: Double,
17 var agonistico: Boolean,
18 var sesso: String,
19 var data_nascita: String,
20 var altezza: Int,
21 var peso_attuale: Double,
22 var sport: String?,
23 var dieta: String
24) : Parcelable { constructor(): this("", "", "", 0.0, false, "", "", 0, 0.0, "", "
 Climatica") }

1 /* Questo pezzo di codice quello che si occupa di gestire un bottone che
 fa cose diverse in base
2 in base a chi attiva l'apertura del Dialog, infatti l'utente pu aprire il
 dialog che contiene
3 questo bottone in due modi, uno quello di cliccare sul bottone '+' dei
 personalizzati, oppure cliccando su un
4 item della recycler view dei personalizzati.
5 Nel primo caso il bottone si dovrà occupare di aggiungere al DB dei
 personalizzati il
6 pasto dopo aver controllato che i valori immessi nei campi siano diversi da
 "" (stringa vuota) e 0.0.
7 Nel secondo caso invece il bottone dovrà gestire la modifica del pasto
 selezionato della
8 recycler view, al primo click su di esso dovrà rendere visibile la form di
 modifica mentre dal secondo click in
9 poi dovrà controllare che l'utente abbia effettivamente cambiato almeno un
 campo del prodotto
10 in tal caso andrà a modificare il pasto nel db */

```

11
12 var flag=false
13     btnAggiungiLista.setOnClickListener {
14         layout_info.visibility = View.VISIBLE
15         layout_quantita.visibility = View.GONE
16         var kcal_salva = kcal.text.toString()
17         var carbo_salva = carbo.text.toString()
18         var proteine_salva = proteine.text.toString()
19         var grassi_salva = grassi.text.toString()
20         var titolo = titolo.text.toString().trim()
21         if(kcal_salva != "" && carbo_salva != "" && proteine_salva != ""
22             && grassi_salva != "" && titolo != "") {
23             if(bottone == "clickItem"){
24                 if (flag && valueAreChanged(position,kcal_salva,
25                     carbo_salva, proteine_salva,grassi_salva,titolo)){
26                     model.updatePersonalizzatoOnDB(
27                         id,requireArguments().getString("bottone")!!,
28                         titolo, kcal_salva.toDouble(),
29                         proteine_salva.toDouble(), carbo_salva.toDouble()
30                         (), grassi_salva.toDouble(), requireContext())
31                         flag=false
32                         dialogLayout.visibility = View.GONE
33                     }else{
34                         Toast.makeText(requireContext(),"Cambia i valori
35                         prima di salvare",Toast.LENGTH_LONG).show()
36                         flag=true
37                     }
38                 }else {
39                     model.setPersonalizzatiOnDB(
40                         requireArguments().getString("bottone")!!, titolo,
41                         kcal_salva.toDouble(),
42                         proteine_salva.toDouble(), carbo_salva.toDouble(),
43                         grassi_salva.toDouble(), requireContext()
44                         )
45                         dialogLayout.visibility = View.GONE
46                     }
47                     model.getPersonalizzati(requireArguments().getString("bottone")!!)
48                 }
49             else
50                 Toast.makeText(requireContext(),"Per favore completa tutti i
51                 campi o modifica i valori prima di salvare",Toast.LENGTH_LONG).show()
52             }

```

```

1 /* Questo pezzo di codice      quello che gestisce l'aggiunta del prodotto dai
   preferiti al
2 diario la soluzione interessante trovata      stata quella di usare un flag
   per gestire la
3 pressione dello stesso bottone che al primo click deve solamente eseguire la
4 visualizzazione del layout all'interno del quale l'utente pu andare ad
   inserire la
5 quantit mentre dal secondo click in poi deve andare a controllare che la
   quantit
6 sia diversa da 0 o da "" (stringa vuota) in tal caso pu aggiungere il
   pasto al diario */
7
8 var flag = false
9     dialog.btnAddDiario.setOnClickListener {
10         dialog.layout_quantita.visibility = View.VISIBLE
11         val quantita = dialog.editTextQuantita.text.toString().toDouble()
12         if(quantita != 0.0 && quantita.toString() != "") {
13             model.setPastoOnDB(requireArguments().getString("bottone")
14             !!,model.preferitiLiveData.value!![position].id,
15             model.preferitiLiveData.value!![position].image,model.
16             preferitiLiveData.value!![position].nome,
17             model.preferitiLiveData.value!![position].calorie,model.
18             preferitiLiveData.value!![position].proteine,
19             model.preferitiLiveData.value!![position].carboidrati,
20             model.preferitiLiveData.value!![position].grassi,
21             quantita,requireContext())
22             dialog.dismiss()
23         }
24         else {
25             if (flag)
26                 Toast.makeText(requireContext(),"Per favore inserisci
una quantit diversa da $quantita se desideri aggiungere il prodotto al
Diario",Toast.LENGTH_LONG).show()
27             flag = true
28         }
29     }

```

3.6 Testing

Abbiamo effettuato due tipologie di TEST: il primo su JUnit (Local Unit Test) ed il secondo con Espresso (Functional UI Test).

Local Unit Test

Per quanto riguarda il primo test, ci siamo soffermati sul calcolo del fabbisogno giornaliero di calorie, in quanto la nostra applicazione si incentra tutto su di esso.

Sono stati creati 6 diversi Test per far vedere come l'algoritmo restituisca il fabbisogno in maniera coerente e secondo i dati precisi inseriti dall'utente.

```
1 package com.example.fittacker
2
3 import com.example.fittacker.model.Utente
4 import org.junit.Assert
5 import org.junit.Before
6 import org.junit.Test
7
8 class FabbisognoTest {
9
10    @Before
11    fun setUp(){
12    }
13
14
15    // Test per vedere se il metodo calculateFabbisogno calcola
16    // correttamente le calorie giornaliere
17
18    @Test
19    fun check_CorrectFabbisogno(){
20        val fabbisogno = Utente()
21        val utente = Utente("Alessandro","Rongoni","alerong@gmail.com"
22            ,1.725,true,"Uomo","2000-05-18",183,76.0,"Calcio","Climatica")
23        Assert.assertEquals(3230, fabbisogno.calculateFabbisogno(utente.
24            data_nascita,utente.sesso,utente.peso_attuale,utente.altezza,utente.LAF))
25    }
26
27    @Test
28    fun check_UncorrectFabbisogno1(){
29        val fabbisogno = Utente()
30        val utente = Utente("Alessandro","Rongoni","alerong@gmail.com"
31            ,1.725,true,"Uomo","2000-05-18",183,76.0,"Calcio","Climatica")
32        Assert.assertNotEquals(3229, fabbisogno.calculateFabbisogno(utente.
33            data_nascita,utente.sesso,utente.peso_attuale,utente.altezza,utente.LAF))
34    }
35
36    @Test
37    fun check_UncorrectFabbisogno2(){
38        val fabbisogno = Utente()
39        val utente = Utente("Alessandro","Rongoni","alerong@gmail.com"
40            ,1.725,true,"Uomo","2000-05-18",183,76.0,"Calcio","Climatica")
41        Assert.assertNotEquals(3231, fabbisogno.calculateFabbisogno(utente.
42            data_nascita,utente.sesso,utente.peso_attuale,utente.altezza,utente.LAF))
43    }
44
45
46
47
48
49
```

```

40
41
42
43
44
45     @Test
46     fun check_LAFSedentarioFabbisogno(){
47         val fabbisogno = Utente()
48         val utente = Utente("Alessandro","Rongoni","alerong@gmail.com",1.2,
49                             true,"Uomo","2000-05-18",183,76.0,"Calcio","Climatica")
50         Assert.assertEquals(2247, fabbisogno.calculateFabbisogno(utente.
51                         data_nascita,utente.sesso,utente.peso_attuale,utente.altezza,utente.LAF))
52     }
53
54     @Test
55     fun check_DonnaFabbisogno(){
56         val fabbisogno = Utente()
57         val utente = Utente("Martina","Rossi","martina.rossiOF.com",1.55,
58                             false,"Donna","2002-06-23",167,55.0,"","Climatica")
59         Assert.assertEquals(1239, fabbisogno.calculateFabbisogno(utente.
60                         data_nascita,utente.sesso,utente.peso_attuale,utente.altezza,utente.LAF))
61     }
62
63     @Test
64     fun check_DonnaCambioEtaFabbisogno(){
65         val fabbisogno = Utente()
66         val utente = Utente("Martina","Rossi","martina.rossiOF.com",1.55,
67                             false,"Donna","1999-06-23",167,55.0,"","Climatica")
68         Assert.assertNotEquals(1239, fabbisogno.calculateFabbisogno(utente.
69                         data_nascita,utente.sesso,utente.peso_attuale,utente.altezza,utente.LAF))
70     }
71
72 }
```

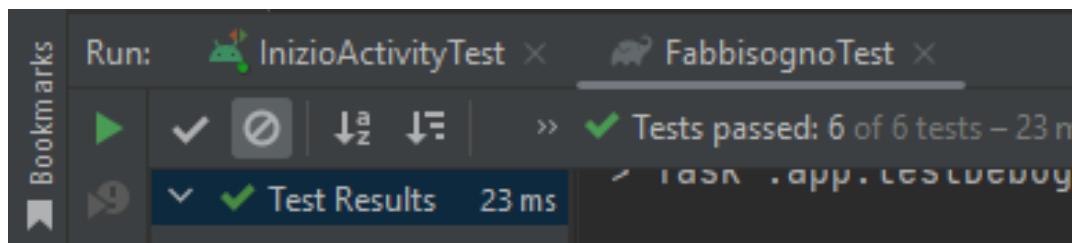


Figura 8: Risultato del test JUnit.

Svolgere questo test ci è stato molto utile, in quanto ci siamo accorti che il calcolo del fabbisogno non avveniva in modo corretto. Il motivo per cui questo succedeva era perché una condizione non era mai verificata, e ciò comportava uno squilibrio nelle calorie finali.

Functional UI Test

Il secondo ed ultimo test va a verificare il corretto funzionamento dell'UI, simulando delle azioni che l'utente può svolgere.

L'Activity che abbiamo deciso di testare è la InizioActivity, in quanto tutto parte da essa. Abbiamo simulato la parte di registrazione e di autenticazione dell'utente.

```

1 package com.example.fittacker.home
2
3 import android.widget.DatePicker
4 import androidx.lifecycle.Lifecycle
5 import androidx.test.core.app.ActivityScenario
6 import androidx.test.core.app.launchActivity
7 import androidx.test.espresso.Espresso
8 import androidx.test.espresso.Espresso.onView
9 import androidx.test.espresso.action.ViewActions
10 import androidx.test.espresso.action.ViewActions.click
11 import androidx.test.espresso.assertion.ViewAssertions.matches
12 import androidx.test.espresso.contrib.PickerActions
13 import androidx.test.espresso.matcher.ViewMatchers.*
14 import androidx.test.ext.junit.runners.AndroidJUnit4
15 import androidx.test.filters.LargeTest
16 import com.example.fittacker.R
17 import com.example.fittacker.autenticazione.InizioActivity
18 import org.hamcrest.Matchers
19 import org.junit.Before
20 import org.junit.Test
21 import org.junit.runner.RunWith
22
23
24 @LargeTest
25 @RunWith(AndroidJUnit4::class)
26 internal class InizioActivityTest{
27     private lateinit var scenario: ActivityScenario<InizioActivity>
28
29     @Before
30     fun setUp(){
31         scenario = launchActivity()
32         scenario.moveToState(Lifecycle.State.RESUMED)
33     }
34
35
36     /*=====
37      ELIMINARE I DATI E SVUOTARE LA CACHE PRIMA DI
38      EFFETTURARE IL TEST
39      Il cellulare mantiene i dati d'accesso e il LoginTest fallirebbe in
40      quanto non trova le view di input
41      per email e password
42      =====*/
43
44     @Test
45     fun registerTest(){
46
47         // TEST DI REGISTRAZIONE in questo caso l'utente    gi    registrato
48         // e non permette la registrazione
49         onView(withId(R.id.btInizia)).perform(click())
50         onView(withId(R.id.rB_sedentario)).perform(click())
51         onView(withId(R.id.sB_agonistico)).perform(click())
52         onView(withId(R.id.imageView16)).check(matches(isDisplayed()))
53         onView(withId(R.id.bt_AvantiObb)).perform(click())
54         onView(withId(R.id.rB_uomo)).perform(click())
55         onView(withId(R.id.bt_Avantisesso)).perform(click())
56         Espresso.pressBack()
57         onView(withId(R.id.rB_donna)).perform(click())
58         onView(withId(R.id.bt_Avantisesso)).perform(click())
59         onView(withText("Dati Personalisi")).check(matches(isDisplayed()))
60         onView(withId(R.id.tv_dataNascita)).perform(click())

```

```

57     val year = 2000
58     val month = 11
59     val day = 15
60     onView(withClassNameMatchers.equalTo(DatePicker::class.java.name)))
61     .perform(PickerActions.setDate(year,month,day))
62     onView(withId(android.R.id.button1)).perform(click())
63     onView(withId(R.id.tE_name)).perform(ViewActions.typeText("Giovanna"))
64   )
65     Espresso.closeSoftKeyboard()
66     onView(withId(R.id.tE_surname)).perform(ViewActions.typeText("Rossi"))
67   )
68     onView(withId(R.id.tE_name)).check(matches(withText("Giovanna")))
69     onView(withId(R.id.tE_surname)).check(matches(withText("Rossi")))
70     onView(withId(R.id.tv_dataNascita)).check(matches(withText("15-11-2000")))
71     Espresso.closeSoftKeyboard()
72     onView(withId(R.id.bt_AvantiDati)).perform(click())
73     onView(withId(R.id.eT_altezza)).perform(ViewActions.typeText("30"))
74     Espresso.closeSoftKeyboard()
75     onView(withId(R.id.bt_AvantiAltezza)).perform(click())
76     onView(withId(R.id.eT_altezza)).check(matches(withText("")))
77     onView(withId(R.id.eT_altezza)).perform(ViewActions.typeText("165"))
78     Espresso.closeSoftKeyboard()
79     onView(withId(R.id.bt_AvantiAltezza)).perform(click())
80     onView(withId(R.id.eT_PesoAttuale)).perform(ViewActions.typeText("29"))
81   )
82     Espresso.closeSoftKeyboard()
83     onView(withId(R.id.bt_AvantiPesoAttuale)).perform(click())
84     onView(withId(R.id.eT_PesoAttuale)).check(matches(withText("")))
85     onView(withId(R.id.eT_PesoAttuale)).perform(ViewActions.typeText("56"))
86   )
87     Espresso.closeSoftKeyboard()
88     onView(withId(R.id.bt_AvantiPesoAttuale)).perform(click())
89     onView(withId(R.id.cB_calcio)).perform(click())
90     onView(withId(R.id.bt_AvantiPesoOobb)).perform(click())
91     onView(withId(R.id.btnRegister)).check(matches(withText("Registrati")))
92   )
93     onView(withId(R.id.InputEmail)).perform(ViewActions.typeText("test@espresso.it"))
94     Espresso.closeSoftKeyboard()
95     onView(withId(R.id.InputPassword)).perform(ViewActions.typeText("123456"))
96     Espresso.closeSoftKeyboard()
97     onView(withId(R.id.InputCorrectPassword)).perform(ViewActions.typeText("123456"))
98     Espresso.closeSoftKeyboard()
99     onView(withId(R.id.btnRegister)).perform(click())
100
101
102
103
104
105

```

```

106
107     @Test
108     fun LoginTest(){
109         /**
110          SUCCESSO nel caso in cui non si ha una cache da ripulire e nessun
111          dato di accesso memorizzato
112          ===*/
113
114         onView(withId(R.id.btAccesso)).check(matches(withText("ACCEDI")))
115         onView(withId(R.id.btAccesso)).perform(click())
116         onView(withId(R.id.InputEmailLogin)).perform(ViewActions.typeText("test@espresso.it"))
117         Espresso.closeSoftKeyboard()
118         onView(withId(R.id.InputPasswordLogin)).perform(ViewActions.typeText("123456"))
119         Espresso.closeSoftKeyboard()
120         onView(withId(R.id.btnLogin)).perform(click())
121
122     }
123 }
```

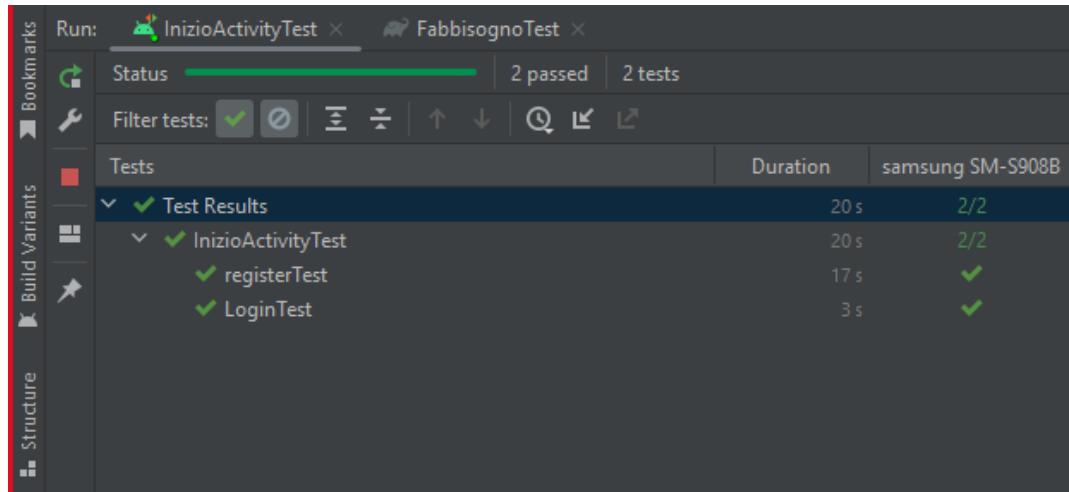


Figura 9: Risultato del test Espresso.

In questi due test di verificano i vari spostamenti tra le facciate della registrazione e dell’accesso, si verificano che i valori sbagliati inseriti vengano resettati al click sul pulsante, si verifica che gli input messi rispecchiano quello che la vista mostra (ad esempio il DataPicker, nome e cognome) e infine si verifica che tornando indietro i valori vengano resettati.

4 Flutter (Dart)

4.1 Requisiti

I requisiti funzionali e non funzionali sono delle azioni/regole che la nostra applicazione dovrà rispettare.

Requisiti funzionali

REQUISITO	DESCRIZIONE
RF1 Login	L'applicazione dovrà gestire l'accesso
RF2 Logout	L'applicazione dovrà gestire la disconnessione
RF3 Registrazione	L'applicazione dovrà gestire la registrazione dell'utente
RF4 Modifica profilo	L'applicazione dovrà gestire la modifica dei dati personali dell'utente
RF5 Registrazione dei datidell'utente	L'applicazione dovrà gestire il salvataggio dei dati dell'utente
RF6 Salvataggio calorie	L'applicazione dovrà gestire il conteggio delle calorie quotidiane
RF7 Salvataggio carboidrati	L'applicazione dovrà gestire il conteggio dei grammi di carboidrati
RF8 Salvataggio proteine	L'applicazione dovrà gestire il conteggio dei grammi di proteine
RF9 Salvataggio grassi	L'applicazione dovrà gestire il conteggio dei grammi di grassi
RF10 Salvataggio alimenti personali	L'applicazione dovrà gestire il salvataggio degli alimenti/pasti/ricette personali dell'utente
RF11 Aggiunta al diario	L'applicazione dovrà gestire il salvataggio nel diario delle calorie assunte e bruciate
RF12 Modifica alimenti personali	L'applicazione dovrà gestire la modifica degli alimenti personali
RF13 Modifica esercizi personali	L'applicazione dovrà gestire la modifica degli esercizi personali
RF14 Rimozione degli alimenti selezionati	L'applicazione dovrà gestire la rimozione degli alimenti selezionati nel diario
RF15 Calcolo del fabbisogno	L'applicazione dovrà gestire e calcolare il fabbisogno calorico giornaliero dell'utente in base ai suoi dati.

Requisiti non funzionali

REQUISITO	DESCRIZIONE
RNF1 Flutter	L'applicazione dovrà essere scritta completamente in Dart, con il framework Flutter
RNF2 Intuitiva	L'applicazione dovrà essere il più intuitiva e semplice possibile
RNF3 Fluida	L'applicazione dovrà compiere azioni e poter navigare al suo interno in maniera fluida e senza crash
RNF4 Estetica	L'applicazione dovrà essere bella esteticamente
RNF4 UserFriendly	L'applicazione dovrà user friendly
RNF5 Privacy	L'applicazione dovrà rispettare la privacy dell'utente secondo le norme dell'UE

4.2 Utilizzo in Flutter

Di seguito è riportato lo schema dei casi d'uso che riassume le principali funzionalità dell'applicazione.

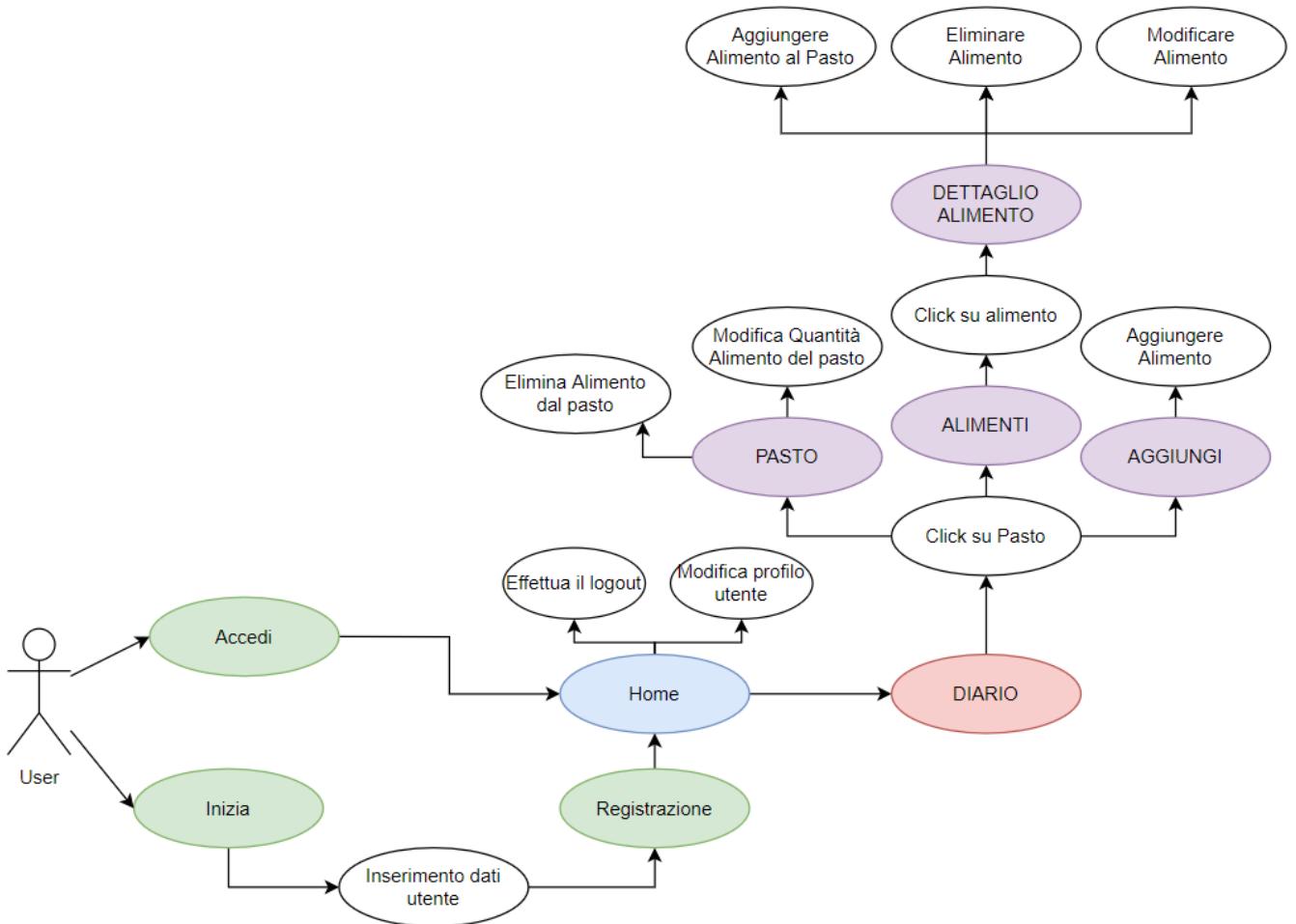


Figura 10: Casi d'uso dell'applicazione

Accesso e Registrazione

Come si può notare, all'inizio l'utente può svolgere due azioni: effettuare l'accesso cliccando sul pulsante "Accedi", oppure avviare la registrazione di un nuovo utente cliccando il pulsante "Inizia".

Avviando una nuova registrazione, appariranno una serie di schermate dove l'utente potrà inserire i suoi dati personali, utili per calcolare il suo fabbisogno calorico giornaliero. Infine dovrà inserire la sua email e la sua password che serviranno per l'accesso e il mantenimento dei dati sul cloud.

Qualora l'utente fosse già registrato, cliccando il pulsante "Accedi" potrà inserire la sua email e password per effettuare l'accesso.

Schermata Home

Una volta effettuato l'accesso, l'utente si troverà nella schermata principale dell'applicazione: il Diario, nella quale potrà visualizzare il suo fabbisogno calorico giornaliero, con le calorie

rimanenti, assunte e bruciate. Inoltre troverà anche delle barre di progresso le quali mostrano i grammi di macro nutrienti assunti e quelli da assumere ancora. Questi dati sono stati generati sulla base dei dati dell'utente inseriti nella registrazione.

Sempre nella sezione del Diario, l'utente può andare a registrare e visualizzare i pasti consumati nel corso della giornata. Cliccando sul pasto, si aprirà una nuova scheda composta da tre schermate: Pasto, Alimenti e Aggiungi.

Inoltre è presente una AppBar, con due icone, Profilo e Logout.

Cliccando sul Profilo, verrà aperta una schermata con tutti i dati dell'utente, i quali possono essere modificati per cambiare il valore del fabbisogno calorico giornaliero. Cliccando su Logout, invece, si effettuerà la disconnessione dall'account.

Pasto

Nella sezione Pasto, sono presenti tutti gli alimenti che l'utente ha registrato per quel determinato pasto della giornata. Cliccando su uno degli alimenti, si potrà decidere se eliminare l'alimento scelto dal pasto, oppure modificarne la quantità selezionata in precedenza.

Una volta selezionata la quantità, è possibile aggiungere il prodotto al diario, oppure salvarlo tra i preferiti.

Alimenti

In questa sezione sono mostrati tutti gli alimenti che sono stati registrati dall'utente. Cliccando su uno di essi, si aprirà una schermata contenente tutti i valori nutrizionali, calcolati su 100g, dell'alimento selezionato. In fondo, è presente la possibilità di aggiungere l'alimento al pasto, selezionando prima la quantità.

N.B. Per selezionare una quantità di grammi non multipla di 100, basta mettere la quantità come numero decimale.

Inoltre è possibile modificare o eliminare l'alimento selezionato, cliccando sul pulsante di modifica in alto a destra, si aprirà una schermata contenente i dati relativi al prodotto, e la possibilità di modificarli, cliccando sul pulsante di eliminazione, l'alimento verrà eliminato

Aggiungi

All'interno di questa sezione è presente una form da compilare, dove inserire tutti i dati relativi ad un nuovo alimento che l'utente vuole aggiungere alla propria lista.

4.3 Architettura in Flutter

L'applicazione è stata scritta utilizzando il framework cross-platform Flutter, in linguaggio Dart, per dispositivi Android e IOS. L'IDE utilizzato per quest'applicazione è stato Android Studio.

Per lo sviluppo dell'applicazione non è stato utilizzato nessuna specifica architettura, ma solamente una divisione dei file in diversi package.

Il package Home per la home page dell'app una volta effettuato l'accesso.

Il package Menu Pasto per la gestione degli alimenti di un utente, il quale potrà effettuare operazioni di creazione, modifica, lettura e cancellazione degli alimenti, e di aggiunta, modifica e cancellazione di alimenti di un determinato pasto.

Il package Registrazione per la gestione della registrazione di un nuovo utente.

Il package Model che contiene i modelli degli oggetti salvati nei database.

Il login e la registrazione attraverso email e password sono stati implementati attraverso Firebase Auth, mentre gli altri dati degli utenti vengono salvati, sempre in remoto, attraverso Cloud Firebase.

I dati quali gli alimenti e i pasti relativi ad ogni utente, sono stati salvati in un database SQLite tramite sqflite.

4.4 Schermate in Flutter

Mockup FitTrackerFlutter v1.0

Vedere Mockup FitTracker v1.0 in 3.4

Mockup FitTrackerFlutter v2.0

Di seguito mostriamo le schermate definitive della nostra applicazione sviluppata per dispositivi Android e IOS. Vengono mostrate tutte le Page in cui l'utente può navigare e con cui può interagire.

La prima schermata con cui si va ad interagire è la InizioPage, da qui possiamo prendere due strade: la prima ci porta alla NomePage, la seconda alla LoginPage, usata per l'accesso.

Con la NomePage si inizia il percorso di registrazione, in cui l'utente va a specificare tutti i suoi dati personali utili per la creazione del diario. Infatti il calcolo delle calorie giornaliere dipende, oltre dall'altezza, il peso e il sesso, dal livello di attività fisica che si svolge settimanalmente.

Ovviamente i campi sono tutti obbligatori. La data è controllata e l'età minima per registrarsi è di 16 anni. L'altezza inserita e il peso devono essere compresi rispettivamente tra i 130-200 cm e 40-200 Kg. Qualora uno degli input fosse sbagliato, sarà impossibile andare avanti.

Una volta completate tutte le informazioni, l'utente può registrarsi, inserendo la propria email e password.



(1) InizioPage



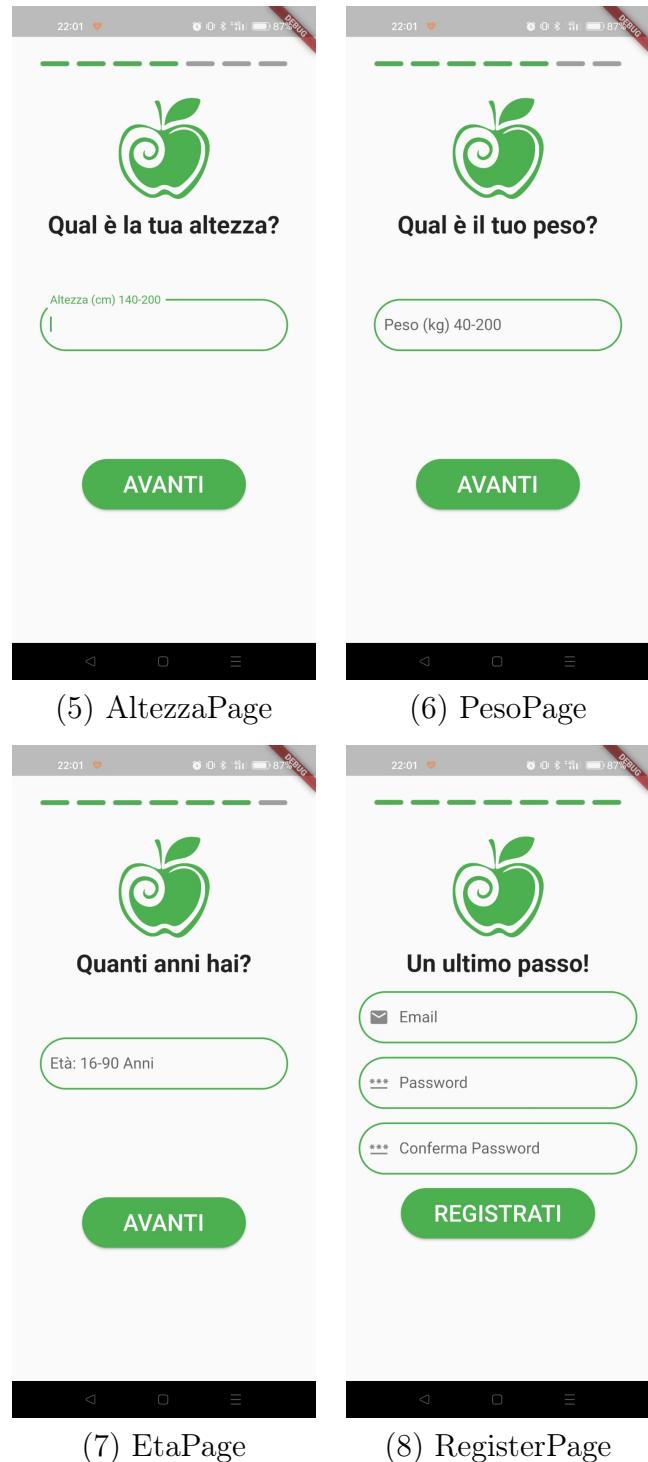
(2) NomePage



(3) StileDiVitaPage

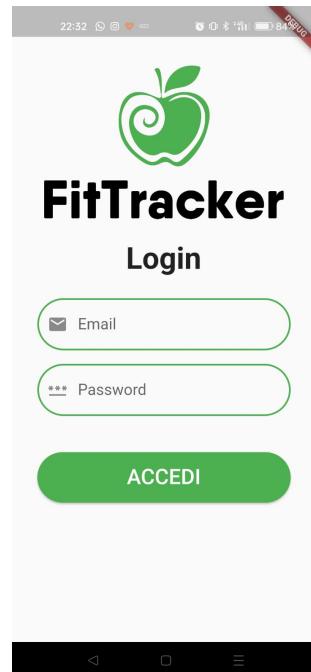


(4) SessoPage



Avvenuta la corretta registrazione dell'utente, si potrà effettuare il login all'interno della LoginPage.

Effettuato l'accesso, viene aperta la HomePage, la quale contiene tutte le informazioni relative ai pasti con calorie e macro nutrienti



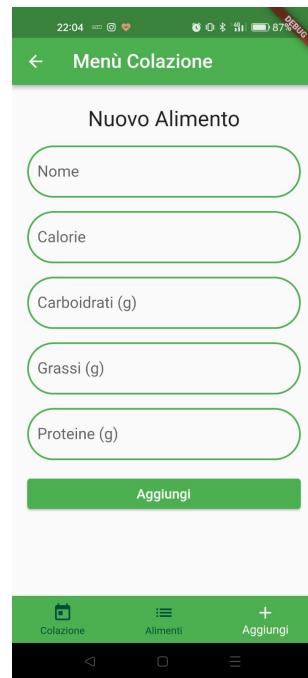
(9) LoginPage



(10) HomePage



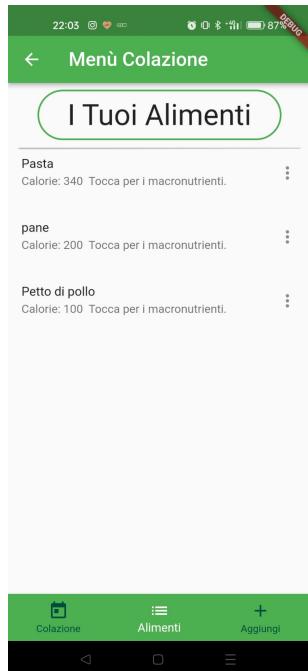
(11) PastoPage



(12) AggiungiAlimentoPage

Cliccando su uno dei pulsanti relativi ai pasti è possibile andare a vedere gli alimenti del relativo pasto (PastoPage). Inoltre è possibile visualizzare tutti gli alimenti che sono stati inseriti dall'utente (PersonalizzatiPage), oppure aggiungere un nuovo alimento alla lista degli alimenti (AggiungiAlimentoPage).

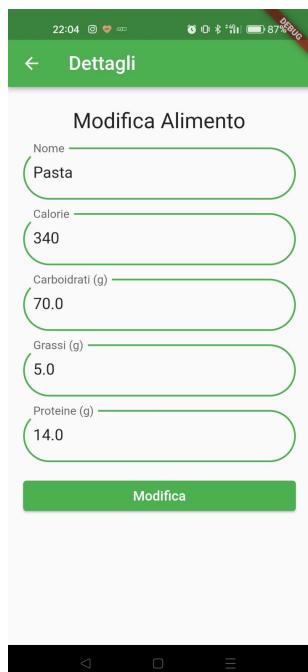
Cliccando su un elemento della lista, a seconda di dove ci si trova, si aprirà una finestra a pop-up, o una nuova schermata, per compiere diverse azioni con quell'item. Ad esempio cliccando un elemento nella lista degli alimenti si aprirà una schermata di dettaglio (DettagliAlimentoPage) dove si potrà vedere il dettaglio dell'alimento, aggiungerlo al diario, modificarlo (ModificaAlimentoPage) oppure eliminarlo.



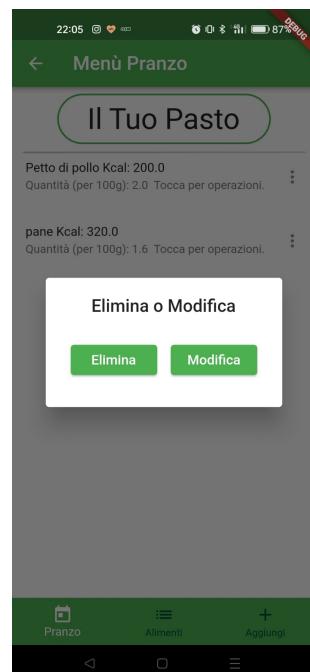
(13) PersonalizzatiPage



(14) DettagliAlimentoPage



(15) ModificaAlimentoPage



(16) Modifica o elimina

Mentre, se si va a cliccare su un elemento della pagina Pasto sarà possibile andare a modificare la quantità dell'alimento o a rimuovere l'elemento dal diario, ripristinando così le calorie assunte.

4.5 Sviluppo

Classi

firebase_options/DefauktFirebaseOptions: classe che contiene le opzioni per la connessione al firebase database

fit_tracker_database/FitTrackerDatabase: classe che gestisce la creazione, l'inserimento, la modifica e la cancellazione dei dati nel database locale.

inizio_page/InizioPage: questa è la pagina iniziale dell'app dove l'utente può decidere se effettuare il login o registrarsi.

login_page/LoginPage: pagina di login dell'applicazione dove l'utente può eventualmente richiedere una nuova password.

main/MyApp: classe iniziale dell'app utilizzata per l'avvio della stessa.

Package: Home

All'interno di questo package troviamo tutte le classi che riguardano la home page e il profilo utente

home_page/HomePage: questa è la pagina che contiene tutte le informazioni sulla nutrizione dell'utente, calorie e macronutrienti assunti e rimanenti, e le calorie per ogni pasto. Viene inoltre gestito il logout.

profilo_page/ProfiloPage: pagina che contiene le informazioni riguardo l'utente e ne permette la modifica. In caso di modifica di parametri quali l'età, il peso, l'altezza, lo stile di vita e il sesso, viene ricalcolato il fabbisogno energetico giornaliero e i relativi macronutrienti.

Package: Menu Pasto

All'interno di questo package troviamo tutte le classi che riguardano la parte di salvataggio di nuovi, salvataggio di alimenti di un pasto, e la visualizzazione delle informazioni nutrizionali di alimenti.

aggiungi_alimento_page/AggiungiAlimentoPage: questa è la classe che permette all'utente di aggiungere un nuovo alimento alla lista dei propri alimenti, che potranno poi essere aggiunti ad un pasto.

dettagli_alimento/DettagliAlimento: classe che mostra all'utente i dettagli nutrizionali di un determinato alimento e ne permette la sua aggiunta al pasto corrente, la sua modifica o la sua eliminazione.

meal_page/MealPage: pagina che contiene tutte le informazioni riguardo gli alimenti e il pasto selezionato, e ne permette le operazioni di aggiunta modifica e cancellazione

modifica_alimento_page/ModificaAlimentoPage: classe che permette la modifica delle informazioni di un alimento selezionato.

pasto_page/PastoPage classe che mostra all'utente tutti gli alimenti di un determinato pasto scelto.

personalizzati_page/PersonalizzatiPage classe che mostra all'utente tutti gli alimenti che possono essere selezionati per un pasto.

Package: Model

All'interno di questo package troviamo tutte le classi che riguardano il salvataggio dei dati sul database Firebase e il database locale.

alimento/AlimentoFields: classe che contiene i campi della tabella degli alimenti nel database.

alimento/Alimento: classe che contiene il modello dell'oggetto Alimento

diario/DiarioFields: classe che contiene i campi della tabella del diario nel database.

diario/Diario: classe che contiene il modello dell'oggetto Diario

alimento/Utente: classe che contiene il modello dell'oggetto Utente

Package: Registrazione

All'interno di questo package troviamo tutte le classi che riguardano la registrazione dell'utente all'applicazione.

altezza_page/AltezzaPage: classe che permette l'inserimento dell'altezza per la registrazione all'applicazione.

eta_page/EtaPage: classe che permette l'inserimento dell'età per la registrazione all'applicazione.

nome_page/NomePage: classe che permette l'inserimento del nome e del cognome per la registrazione all'applicazione.

peso_page/PesoPage: classe che permette l'inserimento del peso per la registrazione all'applicazione.

register_page/RegisterPage: classe che permette l'inserimento dell'email e della password per la registrazione all'applicazione.

sesso_page/SessoPage: classe che permette l'inserimento dell'altezza per la registrazione all'applicazione.

stile_di_vita_page/StileDiVitaPage: classe che permette l'inserimento dello stile di vita per la registrazione all'applicazione.

5 Conclusioni, Known Bugs e Future Features

5.1 Android

Un bug che abbiamo riscontrato riguarda l'aggiornamento delle progressBar dei carboidrati, proteine e dei grassi alla modifica della dieta. Infatti, i valori massimi vengono aggiornati correttamente in base alla tipologia di dieta, ma la visualizzazione della lunghezza della progressBar rimane invariata. Cliccando su un qualsiasi pulsante di pasto/esercizio, la lunghezza delle barre viene aggiornata.

Implementazioni future potrebbero riguardare nuove statistiche da controllare e funzionalità relative al fitness.

5.2 Flutter

Nell'app Flutter, quando un alimento viene inserito nel diario di un pasto, prima di poter modificare o eliminare quell'alimento dalla lista di tutti gli alimenti, eliminare dal diario del pasto l'alimento, altrimenti rimarrà traccia nel diario dell'alimento non modificato o eliminato.