



**Document:** FitCal\_Architettura

**Revision:** 0.3

**Progetto:**

**FitCal**

**Titolo del documento:**

## **Documento di Architettura**

**Document Info:**

Document name	D3-FitCal_Architettura	Document number	AR3
Description	Documento contenente diagramma classi e codice OCL		
Authors	Azemi Kevin Lollato Filippo Mattioli Michele		

# INDICE

1. [Contenuti documento](#)
2. [Diagramma classi](#)
  - 2.1. [Pagine](#)
  - 2.2. [Meal](#)
  - 2.3. [User](#)
  - 2.4. [Class Diagram Finale](#)
3. [Codice Object Constraint Language](#)
  - 3.1. [User](#)
  - 3.2. [Food](#)
  - 3.3. [Meal](#)
  - 3.4. [Homepage](#)
  - 3.5. [Personal Homepage](#)
  - 3.6. [TDEE Calculator](#)
  - 3.7. [Account Page](#)
4. [Diagramma classi con OCL](#)

## **Contenuti del documento**

Il seguente documento conterrà le specifiche dell'architettura del progetto FitCal mediante diagrammi in UML (Unified Modeling Language) e OCL (Object Constraint Language).

L'obiettivo è di fornire una chiara rappresentazione della struttura dell'applicazione e delle interazioni dei componenti.

## **Diagramma classi**

Nella seguente sezione illustreremo il diagramma delle classi. Verrà fornita una descrizione dei parametri e metodi per realizzare al meglio il sito presentato nei documenti precedenti.

Ogni relazione ed associazione fra i componenti fornirà informazioni sugli effetti delle varie azioni.

## Pagine

L'applicazione è costituita da 4 pagine principali:

1. Homepage: accessibile a tutti e permette la registrazione e login all'utente;
2. PersonalHomepage: raggiungibile solo dopo login, conterrà il monitoraggio delle calorie e macronutrienti in base alla data fornita;
3. TDEECalculator: raggiungibile solo dopo login, permette il calcolo del fabbisogno tramite l'inserimento dei dati come specificato nei requisiti funzionali;
4. AccountPage: raggiungibile solo dopo login, permette la modifica del proprio profilo.

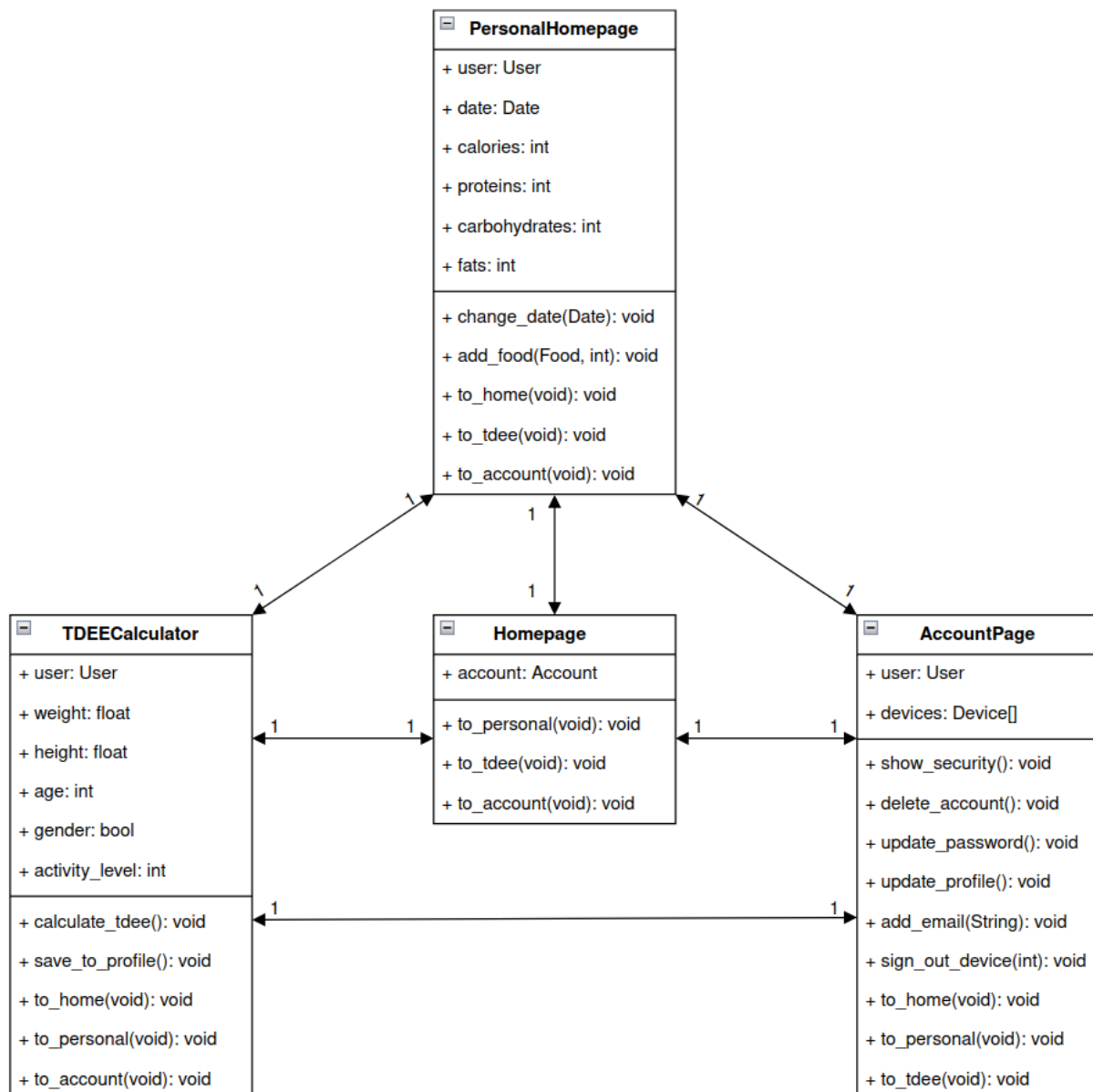


Fig. 1: Pagine principali.

## Meal

Nel seguente diagramma è rappresentata la relazione tra la classe Meal e Food.

Meal rappresenta l'insieme dei cibi consumati durante un determinato giorno, suddiviso per i 4 pasti principali:

- colazione;
- pranzo;
- cena;
- snack.

Per la rappresentazione del singolo cibo è sufficiente memorizzare i grammi consumati ed il barcode in quanto ci permette di recuperare i dati corrispondenti (Immagine, macronutrienti, nome) tramite una chiamata ad una API esterna.

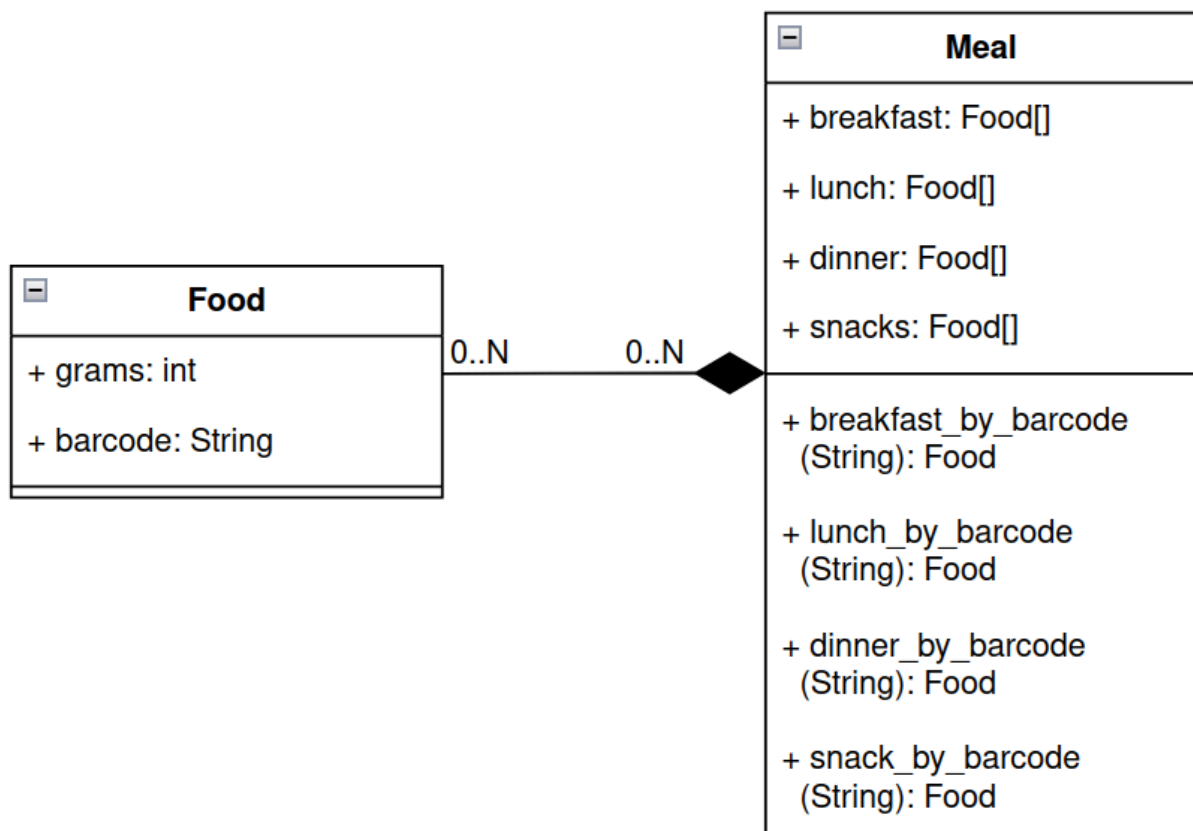


Fig. 2: Food e Meal Class Diagram.

## User

All'interno dell'applicazione verrà fatta una distinzione tra due tipi di utente:

- anonimo: potrà fare solamente il login e signup;
- autenticato: potrà accedere alle varie pagine.

Per semplificare l'implementazione è stata creata una interfaccia disponibile alla homepage per distinguere velocemente se l'utente è autenticato o meno.

Per incrementare la sicurezza e conservazione dei dati ci affideremo a Clerk per il monitoraggio degli accessi e salvataggio dei dati sensibili quali l'email, stato della sessione, dispositivo da cui si è connesso.

Questo permette la disabilitazione di accessi involuti da terze parti all'utente, in quanto può visualizzare che dispositivi sono attualmente connessi e l'ultima data di accesso.

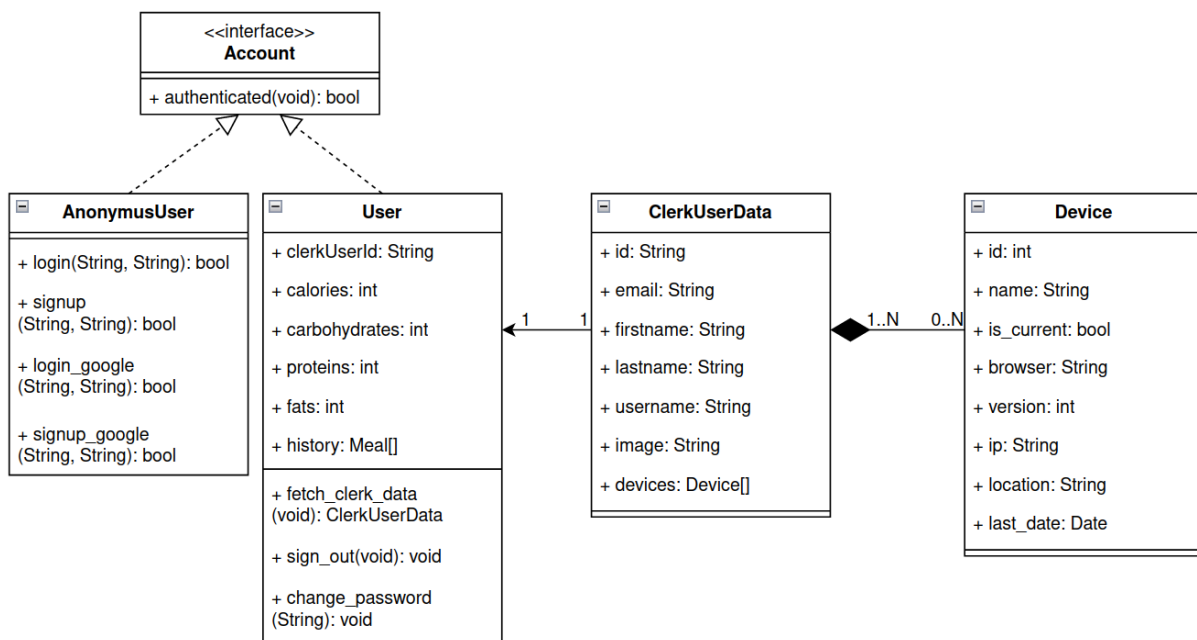


Fig. 3: User, AnonymusUser, ClerkUserData e Device Class Diagram.

## Class Diagram Finale

In seguito riportiamo il class diagram completo con le interconnessioni tra le varie classi ed interazioni.

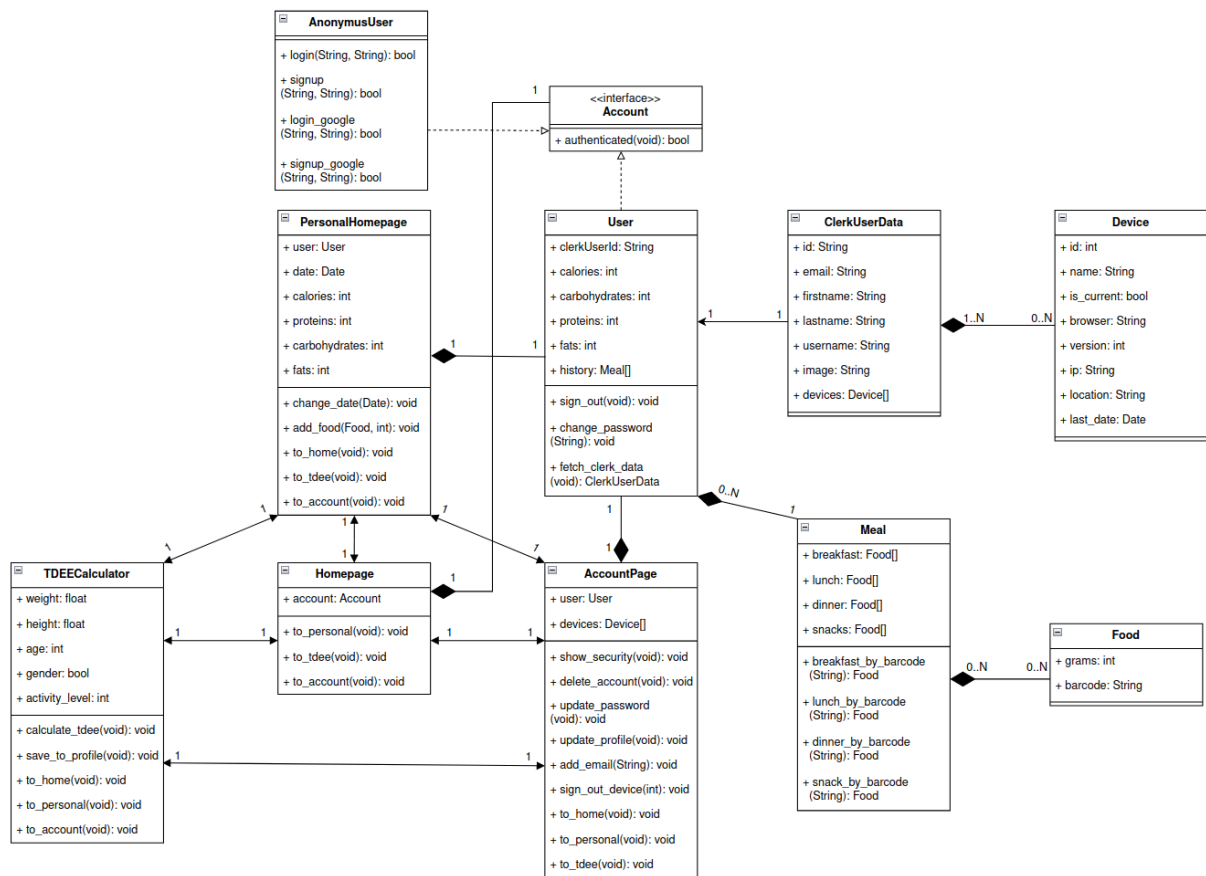


Fig. 4: Class Diagram Finale.

# Codice Object Constraint Language

Nella seguente sezione verranno riportate le interazioni tra i componenti tramite il linguaggio OCL (Object Constraint Language).

## User

Per l'utente sono presenti due invarianti:

- il *clerkUserId* non può mancare;
- i dati riguardanti i macronutrienti non possono essere mai negativi.

In codice OCL:

**context** User

**inv:** User.clerkUserId != NULL and  
 User.calories >= 0 and  
 User.carbohydrates >= 0 and  
 User.proteins >= 0 and  
 User.fats >= 0

**context** User

**inv:** User.clerkUserId != NULL and  
 User.calories >= 0 and  
 User.carbohydrates >= 0 and  
 User.proteins >= 0 and  
 User.fats >= 0

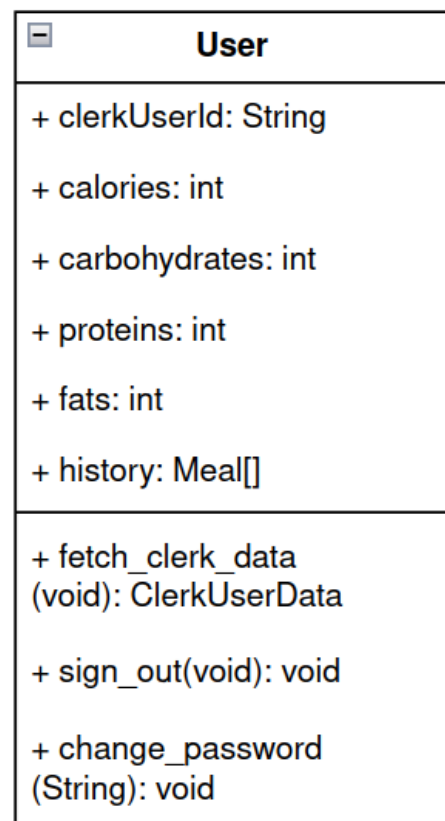


Fig. 5: User OCL.



## Food

Per il Food i grammi devono essere sempre positivi e il barcode non può mancare

In codice OCL:

**context** Food

**inv:**    Food.grams >= 0 and  
          Food.barcode != NULL

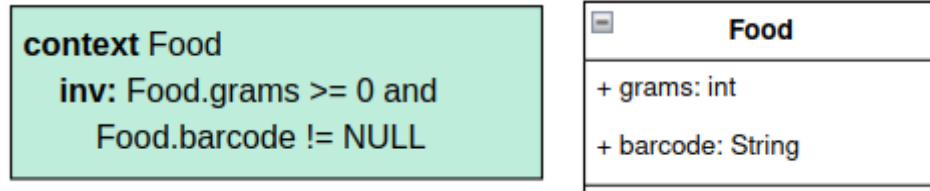


Fig. 6: Food OCL.

## Meal

Per i metodi disponibili in Meal l'unico controllo necessario è che il parametro passato non sia NULL.  
In codice OCL:

**context** Meal::breakfast\_by\_barcode(barcode: String)

**inv:** barcode != NULL

**context** Meal::lunch\_by\_barcode(barcode: String)

**inv:** barcode != NULL

**context** Meal::dinner\_by\_barcode(barcode: String)

**inv:** barcode != NULL

**context** Meal::snack\_by\_barcode(barcode: String)

**inv:** barcode != NULL

**context** Meal::breakfast\_by\_barcode(barcode: String)

**inv:** barcode != NULL

**context** Meal::lunch\_by\_barcode(barcode: String)

**inv:** barcode != NULL

**context** Meal::dinner\_by\_barcode(barcode: String)

**inv:** barcode != NULL

**context** Meal::snack\_by\_barcode(barcode: String)

**inv:** barcode != NULL

Meal
+ breakfast: Food[]
+ lunch: Food[]
+ dinner: Food[]
+ snacks: Food[]
+ breakfast_by_barcode (String): Food
+ lunch_by_barcode (String): Food
+ dinner_by_barcode (String): Food
+ snack_by_barcode (String): Food

Fig. 7: Meal OCL.

## Homepage

Nella homepage per passare alle altre pagine è necessario effettuare il login se l'account non è già autenticato.

In codice OCL:

```
context Homepage::to_personal()
pre:    Homepage.account.authenticated() != 1
post:   AnonymusUser::login()
```

```
context Homepage::to_tdee()
pre:    Homepage.account.authenticated() != 1
post:   AnonymusUser::login()
```

```
context Homepage::to_account()
pre:    Homepage.account.authenticated() != 1
post:   AnonymusUser::login()
```

```
context Homepage::to_personal()
  pre: Homepage.account.authenticated() != 1
  post: AnonymusUser::login()

context Homepage::to_tdee()
  pre: Homepage.account.authenticated() != 1
  post: AnonymusUser::login()

context Homepage::to_account()
  pre: Homepage.account.authenticated() != 1
  post: AnonymusUser::login()
```

Homepage
+ account: Account
+ to_personal(void): void
+ to_tdee(void): void
+ to_account(void): void

Fig. 8: Homepage OCL.

## Personal Homepage

Per la pagina personale le date fornite e salvate devono essere sempre corrette. Inoltre il valore per l'aggiunta del Food deve rientrare tra 0 e 3 (compresi) in quanto sono convertiti direttamente ai pasti giornalieri.

Continuano a valere le invarianze sui macronutrienti (solo valori positivi).

In codice OCL:

**context** PersonalHomepage

**inv:**    PersonalHomepage.user != NULL and  
           PersonalHomepage.date.is\_valid() == 1 and  
           PersonalHomepage.calories >= 0 and  
           PersonalHomepage.proteins >= 0 and  
           PersonalHomepage.carbohydrates >= 0 and  
           PersonalHomepage.fats >= 0

**context** PersonalHomepage::change\_date(date: Date, meal: int)

**inv:**    date.is\_valid() == 1 and  
           0 <= meal <= 3

**post:**   PersonalHomepage.date = date

**context** PersonalHomepage

**inv:**    PersonalHomepage.user != NULL and  
           PersonalHomepage.date.is\_valid() == 1 and  
           PersonalHomepage.calories >= 0 and  
           PersonalHomepage.proteins >= 0 and  
           PersonalHomepage.carbohydrates >= 0 and  
           PersonalHomepage.fats >= 0

**context** PersonalHomepage::change\_date(date: Date, meal: int)

**inv:**    date.is\_valid() == 1 and  
           0 <= meal <= 3

**post:**   PersonalHomepage.date = date


 <b>PersonalHomepage</b>
+ user: User + date: Date + calories: int + proteins: int + carbohydrates: int + fats: int
+ change_date(Date): void + add_food(Food, int): void + to_home(void): void + to_tdee(void): void + to_account(void): void

Fig. 9: Personal Homepage OCL.

## TDEE Calculator

Per la pagina del calcolo del TDEE i valori forniti devono rimanere corretti:

- peso non negativo;
- altezza non negativa;
- età non negativa;
- livello d'attività compreso tra 0 e 4 compresi (da Sedentario a Super Attivo).

Mentre per poter salvare i dati sul profilo bisogna aver prima chiamato la funzione *calculate\_tdee*

In codice OCL:

**context** TDEECalculator

**inv:** TDEECalculator.user != NULL and  
TDEECalculator.weight >= 0 and  
TDEECalculator.height >= 0 and  
TDEECalculator.age >= 0 and  
0 <= TDEECalculator.activity\_level <= 4

**context** TDEECalculator::save\_to\_profile()

**pre:** TDEECalculator::calculate\_tdee()

**context** TDEECalculator

**inv:** TDEECalculator.user != NULL and  
TDEECalculator.weight >= 0 and  
TDEECalculator.height >= 0 and  
TDEECalculator.age >= 0 and  
0 <= TDEECalculator.activity\_level <= 4

**context** TDEECalculator::save\_to\_profile()

**pre:** TDEECalculator::calculate\_tdee()

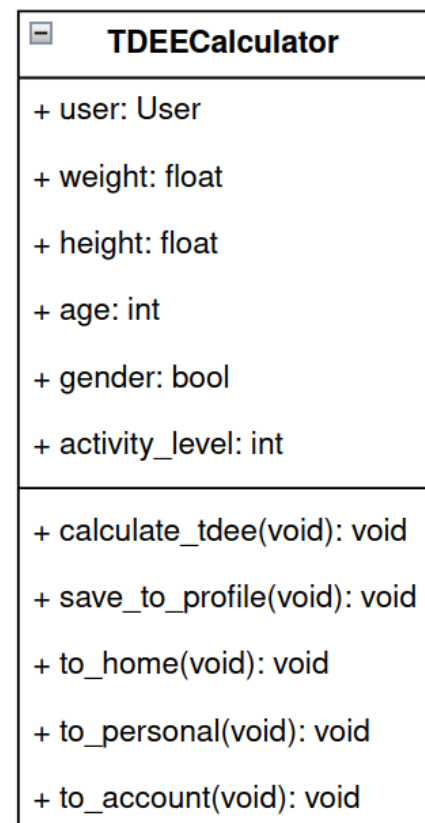


Fig. 10: TDEECalculator OCL.

## Account Page

Per la pagina di modifica del profilo bisogna assicurare che i valori forniti alle funzioni *add\_email* e *sign\_out\_device* siano corretti, ovvero rispettivamente:

- la stringa fornita sia una valida email;
- l'id device deve essere presente nella liste dei device.

In codice OCL:

**context** AccountPage

**inv:** AccountPage.user != NULL

**context** AccountPage::add\_email(email: String)

**inv:** email.is\_valid\_email() == 1

**context** AccountPage::sign\_out\_device(id: int)

**inv:** AccountPage.devices.contains(id) == 1

**context** AccountPage

**inv:** AccountPage.user != NULL

**context** AccountPage::add\_email(email: String)

**inv:** email.is\_valid\_email() == 1

**context** AccountPage::sign\_out\_device(id: int)

**inv:** AccountPage.devices.contains(id) == 1

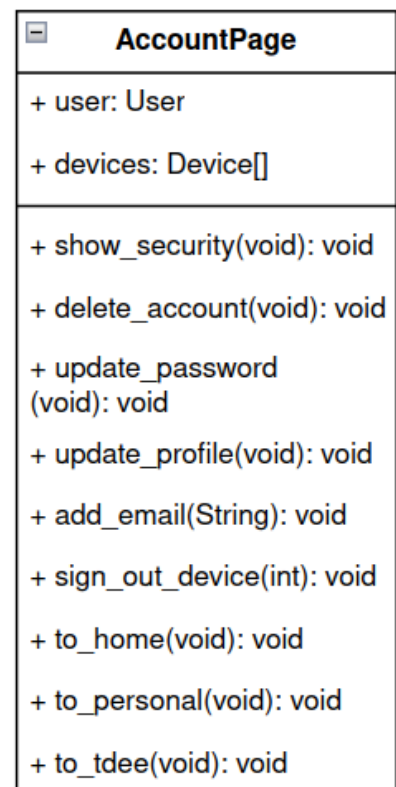


Fig. 11: AccountPage OCL.

# Diagramma classi con OCL

Concludiamo riportando il diagramma completo delle classi con corrispettivo linguaggio OCL.

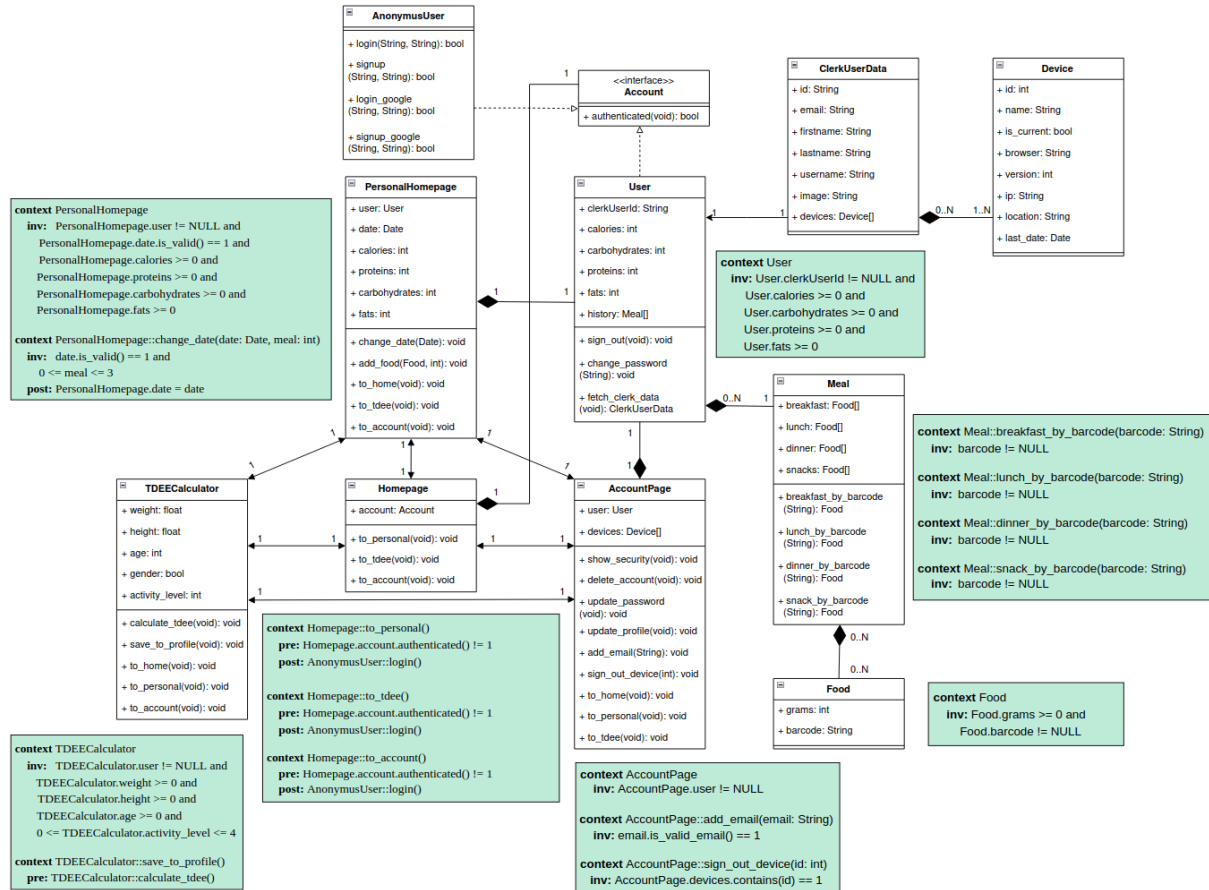


Fig. 11: Diagramma Classi con OCL.