Progetto:

FitGYM44

Titolo del documento:

Documento di architettura

Document Info

Doc. Name	*D3-FitGYM44_Architettura*	Doc. Number	3
Description	Documento contenente diagrammi delle classi e codice in OCL		

Indice:

0. Scopo del documento

Il presente documento riporta la definizione dell'architettura del progetto FitGYM44 usando diagrammi delle classi in Unified Modeling Language (UML) e codice in Object Constraint Language (OCL). Nel precedente documento è stato presentato il diagramma degli use case, il diagramma di contesto e quello dei componenti. Ora, tenendo conto di questa progettazione, viene definita l'architettura del sistema dettagliando da un lato le classi che dovranno essere implementate a livello di codice e dall'altro la logica che regola il comportamento del software. Le classi vengono rappresentate tramite un diagramma delle classi in linguaggio UML. La logica viene descritta in OCL perché tali concetti non sono esprimibili in nessun altro modo formale nel contesto di UML.

1. Elenco delle classi

1.1 – User and Credentials

1.1.1 – User

Questa classe implementa le funzionalità dell'attore utente (vedi RF 11). Essa ha lo scopo di contenere le varie tipologie di dati, informazioni relative all'utente e i metodi che svolgono le sue funzionalità.

Attributi:

- `session_token` è un campo di tipo `string` che consente all'utente di poter accedere alle varie pagine dell'applicazione in sicurezza. Esso va reimpostato ad un valore di default dopo la chiusura di ogni sessione
- `user_info` è un campo che contiene un oggetto di tipo `Form` che fornisce tutte le informazioni fisiche dell'utente necessarie per registrarsi nell'applicazione
- `plan_type` è un campo di tipo `enum` che contiene la tipologia di piano che l'utente ha selezionato
- `training_schedule` è un campo di tipo `TrainingSchedule` che contiene tutta la serie di esercizi personalizzati per l'utente in base ai suoi obiettivi.
- `choosen_mode` è un campo di tipo `Theme` (nell'utente base può corrispondere solo a "scuro" o "chiaro") che contiene il tipo di tema scelto dall'utente.
- `steps_made` è un campo di tipo `int` che indica i numeri di passi fatti durante la giornata collegandosi tramite bluetooth al dispositivo di monitoraggio (vedi RF 31).
- `pics` è un campo array di tipo `Photo` che contiene tutte le foto che l'utente ha caricato sull'applicazione per monitorare i suoi progressi.
- `notification_prefereces` è un campo di tipo `NotificationSystem` che gestisce le notifiche selezionate dall'utente.
- `badges` è un campo array di tipo `Badge` che contiene tutti i badge che l'utente ha ricevuto al seguito del raggiungimento di un `Achievement`
- `calendar` è il campo che consente all'utente di visualizzare all'interno di un calendario le proprie attività: `Training`, `Meal`, `Achievement` ecc..
- `mail_system` è un campo di tipo `MailSystem`. È un'istanza del sistema delle mail, si occupa di gestire tutte le mail relative al singolo utente.

- `payment_system`: è un attributo di tipo `PaymentSystem` che si occupa dell'interazione dell'utente con il sistema di pagamento.

Metodi:

- `signUp`: Questo è un metodo `boolean` che viene utilizzando solo ed esclusivamente quando l'utente si registra per la prima volta al sistema (vedi RF 4). Se la registrazione viene effettuata correttamente ritorna '`true`', altrimenti '`false`'.
- `setTraining`, metodo di tipo `void` che imposta l'allenamento che l'utente ha deciso svolgere.
- `SetTheme`, metodo di tipo `void` che cambia il tema dell'applicazione con il tema che ha selezionato l'utente
- `upgradePlan` è un metodo di tipo `void` che è utilizzato per modificare il campo `plan_type` a `Premium`
- `deleteAccount`, metodo di tipo `void` che consente all'utente di eliminare il proprio account qualora non desideri più utilizzarlo.
- `getUserData`: questa funzione ha lo scopo di ottenere tutte le informazioni rilevanti relative all'utente e ritorna un oggetto di classe `Form`.
- `getScheda`: metodo che recupera la scheda di allenamento personalizzata dell'utente e ritorna un oggetto di tipo `TrainingSchedule`.
- `deleteScheda`: metodo che elimina la scheda di allenamento personalizzata dell'utente, impostando `training_schedule` a `null`.

1.1.2 - Form

Questa classe viene utilizzata per poter raccogliere tutte le informazioni personali e fisiche dell'utente che serviranno poi al sistema per fornirgli tutte le funzionalità personalizzate (vedi RF 1). Al suo interno sono presenti degli attributi insieme ai loro relativi getter che ritornano un valore corrispondente alla tipologia dell'attributo richiesto e in alcuni casi anche dei setter, dei metodi di tipo `void` che modificano i campi con dei valori presi come argomento. Le categorie di attributi appartenenti alla classe sono: quelle sulla persona, quelle sul fisico (tutti quanti di tipo `int`) che servono al sistema per capire il tipo di fisico dell'utente e infine i dati relativi all'autenticazione all'interno del sistema.

Attributi:

- name
- surname
- date_of_birth
- weight
- target
- height
- thighs
- shoulders
- waist
- biceps
- email
- username
- password

Metodi:

- **Vincoli sulla password(vedi RNF 9):**
  - `containsCharacters`: **Questo è un metodo** `boolean` **che prende come argomento una stringa di input "**`input_data`**" e controlla che questa sia formata da soli caratteri dell'alfabeto. Se sì, ritorna '**`true`**', altrimenti '**`false`**'.**
  - `containsUpperCase`: **Questo è un metodo** `boolean` **che prende una stringa di input "**`input_data`**" e controlla che questa contenga almeno una lettera maiuscola. Se è presente almeno una maiuscola ritorna '**`true`**', altrimenti '**`false`**'.**
  - `containsLowerCase`: **Questo è un metodo** `boolean` **che prende una stringa di input "**`input_data`**" e controlla che questa contenga almeno una lettera minuscola. Se è presente almeno una minuscola ritorna '**`true`**', altrimenti '**`false`**'.**
  - `containsNumber`: **Questo è un metodo** `boolean` **che prende una stringa di input "**`input_data`**" e controlla che questa contenga almeno un numero. Se è presente almeno un numero ritorna '**`true`**', altrimenti '**`false`**'.**
- `correctFormat`: **Questo è un metodo** `boolean` **che prende come argomento una stringa "**`date`**" e controlla che questa rappresenti una data nel formato YYYY/MM/DD. Se lo rispetta ritorna '**`true`**', altrimenti '**`false`**'(vedi RNF 9).**

- `existEmail`: Questo è un metodo `boolean` che prende come argomento una stringa "`input_email`" e controlla che la mail esista davvero. Se esiste ritorna '`true`', altrimenti '`false`'(vedi RNF 9).

1.1.3 - Controllore di accesso

La classe `LogController` è una classe che gestisce le sessioni di accesso dell'utente all'applicazione: essa concede all'utente di accedere all'applicazione se l'utente si è autenticato correttamente e di chiudere la sessione nel caso in cui voglia uscire (vedi RF 5).

Attributi:

- `user`, attributo che corrisponde alla classe dell'utente che intende effettuare l'accesso all'applicazione

Metodi:

- `findUser`, è un metodo di tipo `boolean`, che riceve come argomento l'`email` e la `password` usata dall'utente per autenticarsi. Esse vengono utilizzate chiedendo alla classe `Database` di effettuare una ricerca all'interno del database, per vedere se esiste al suo interno un utente che possiede `input_email` e `input_password` come credenziali. Il metodo `findUser`, quando riceverà il risultato della ricerca della classe `Database`, se conterrà un oggetto di tipo `User` imposterà il campo `user` uguale ad esso e ritornerà "`true`" come risultato, altrimenti imposterà il campo `user` a "`null`" e ritornerà "`false`"
- `setToken` è il metodo di tipo `void`, che si occupa di impostare il campo `token` dell'oggetto `user` ad un valore generato casualmente, con cui l'utente potrà navigare all'interno dell'applicazione in maniera sicura
- `logOut`, metodo di tipo `void`, che imposta ad un valore di default il campo `token` dell'oggetto `user`

1.1.4 - Controllore delle credenziali

La classe `CredentialController` è una classe che si occupa di effettuare tutta quella serie di operazioni necessarie per: effettuare un recupero password in caso l'utente l'abbia dimenticata chiedendo alla classe `MailSystem` di inviare una email

all'indirizzo ricevuto come argomento; oppure di cambiare la propria password chiedendo alla classe `MailSystem` di inviare una email di conferma (vedi RF 8 e 9).

Attributi:

- `credential_owner`, campo di tipo `User` che corrisponde all'utente che intende effettuare l'operazione sulle proprie credenziali
- `email`, attributo di tipo `string` che corrisponde alla mail inserita dall'utente

Metodi:

- `setCredentialOwner`, metodo di tipo `void`, che si occupa di ricevere come argomento un oggetto di tipo `User` che corrisponde alla classe dell'utente che intende effettuare una operazione sulle proprie credenziali e la utilizza per impostare il campo `credential_owner`
- `findUser`, è un metodo di tipo `boolean`, che riceve come argomento l'`email` dell'utente che intende effettuare una operazione sulle proprie credenziali. Essa viene utilizzata per di chiedere alla classe `Database` di effettuare una ricerca per vedere se all'interno del database esiste un utente che possiede `input_email` come `email`. Il metodo `findUser`, quando riceverà il risultato della ricerca della classe `Database`, se conterrà un oggetto di tipo `User` imposterà il campo `credential_owner` uguale ad esso e ritornerà "`true`" come risultato, altrimenti imposterà il campo `credential_owner` a "`null`" e ritornerà "`false`"
- `passwordRecovery`, metodo di tipo `boolean` che ritorna "`false`" se le due password ricevute come argomento non sono uguali, mentre ritorna "`true`" se è stata impostata con successo una nuova password
- `passwordChange`, metodo di tipo `boolean` che ritorna "`false`" se la vecchia password ricevuta come argomento non è corretta, mentre ritorna "`true`" se è stata impostata con successo una nuova password

1.1.5 – Utente premium

La classe `PremiumUser` implementa le funzionalità aggiuntive possedute dall'utente premium tra cui ricevere una dieta personalizzata composta da una vasta gamma di pasti consigliati in base ai suoi obiettivi (vedi RF 11). Avrà anche quindi la possibilità di avere un conteggio preciso delle calorie assunte e consumate durante il giorno,

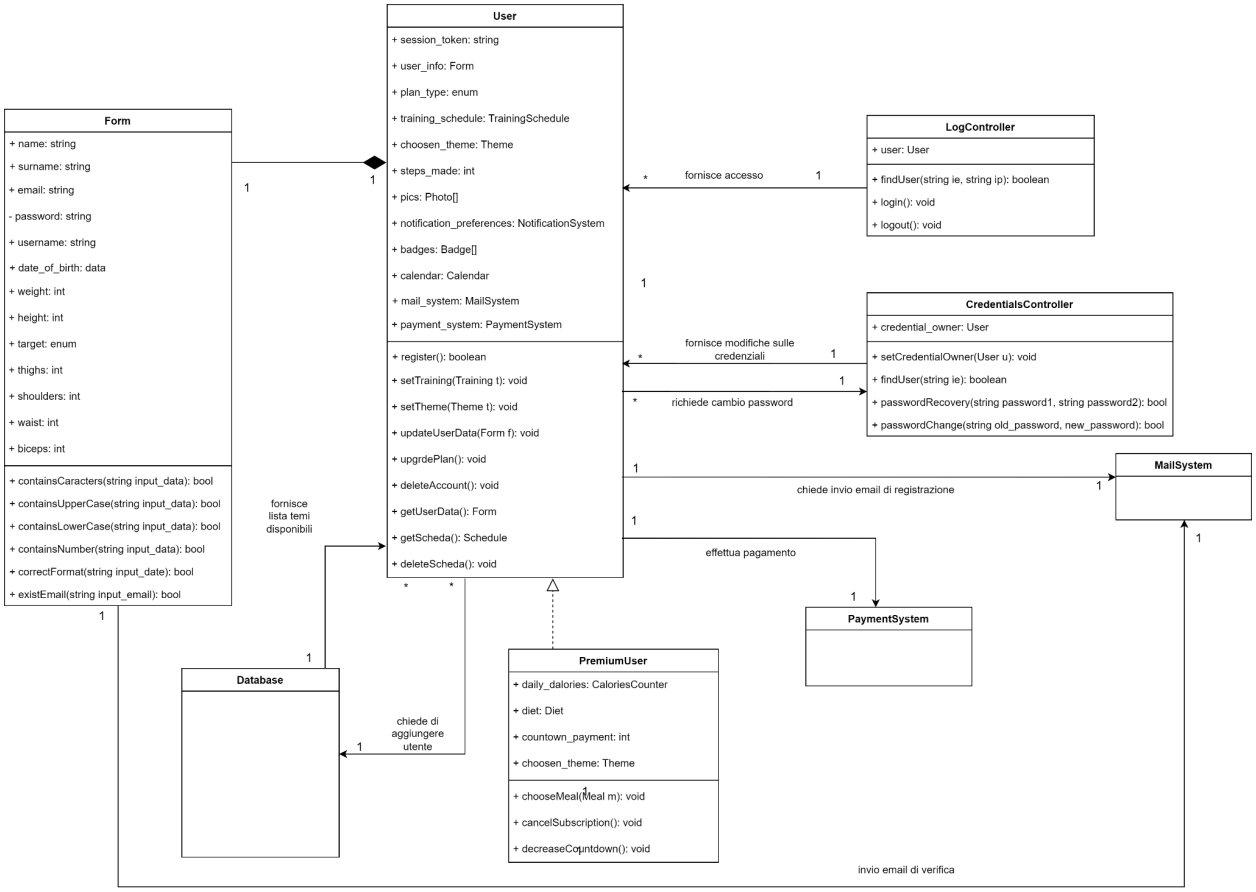
infine gli sarà concesso scegliere il tema da applicare alla propria applicazione da un elenco maggiore rispetto a quello dell'utente premium:

Attributi:

- `daily_calories`, attributo di tipo `CaloriesCounter`, che ha il compito di calcolare la somma delle calorie assunte e consumate durante il giorno
- `diet`, attributo di tipo `Diet`, che contiene l'elenco di pasti consigliati per l'utente
- `countdown_payment`, campo che tiene conto della scadenza nell'effettuare il pagamento, in modo che ogni trenta giorni l'utente paghi la quota dell'abbonamento
- `choosen_theme` è un campo di tipo `Theme` che contiene il tipo di tema scelto dall'utente.

Metodi:

- `chooseMeal` è un metodo di tipo `void`, che invia le calorie del pasto selezionato dall'utente alla classe `CaloriesCounter` e che mostra i vari attributi del pasto.
- `cancelSubscription` è un metodo di tipo `void`, che annulla l'abbonamento premium e imposta il campo plan dell'utente a `Normal`.
- `decreaseCountdown` è un metodo di tipo `void`, che si occupa di diminuire la scadenza al pagamento dell'abbonamento: se il suo valore è "0" chiamerà il metodo `pay` del campo `payment_system` e lo resetterà impostandolo a "30".



1.2 – Achievements

1.2.1 – Badge

La classe Badge definisce la struttura dei riconoscimenti che l'utente ottiene al raggiungimento di determinati obiettivi previsti dal sistema. (Vedi RF 26)

Attributi:

- `id`: attributo di tipo `int` che identifica univocamente il badge.
- `descrtiption`: attributo di tipo stringa che contiene la descrizione testuale del badge.
- `is_obtained`: attributo di tipo `boolean` che indica l'ottenimento o meno del badge.
- `achievement_required` : attributo di tipo `Achievement` che indica l'obiettivo necessario da raggiungere per sbloccare il badge.

Metodi:

- `isObtained`: metodo che permette di ottenere il valore dell'attributo `is_obtained`.
- `getDescription`: metodo che permette di ottenere il valore dell'attributo `description`
- `setDescrtiption`: metodo che permette di assegnare come valore all'attributo "description" la stringa passata.
- `getId`: metodo che permette di ottenere il valore dell'attributo `id`.
- `addBadge`: metodo che prende come argomento un `User` e permette di assegnargli il badge.

1.2.2 – Achievement

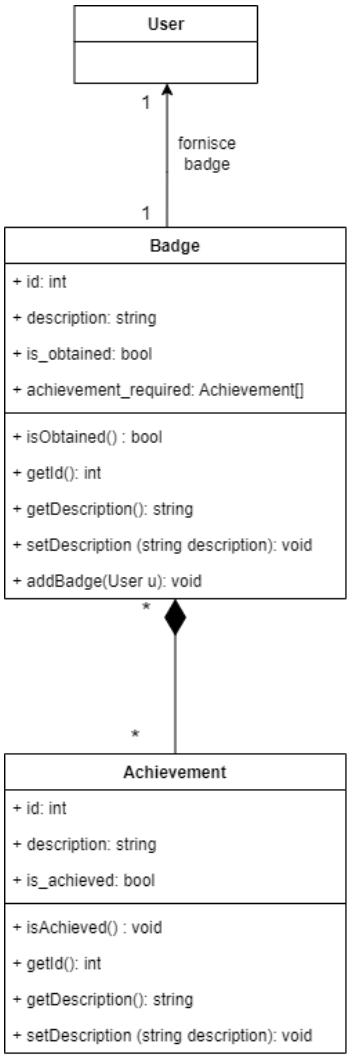
La classe `Achievement` implementa tutti i campi e i metodi necessari per far sì che l'utente possieda degli obiettivi da raggiungere in modo da essere costantemente motivato (vedi RF 27).

Attributi:

- `id`: è il campo di tipo `int` che identifica in maniera univoca l'oggetto `Achievement`.
- `description`: è un campo di tipo `string` che contiene una descrizione di quello che deve fare l'utente per ottenere un oggetto `Achievement` specifico.
- `is_achieved`: è un campo di tipo `boolean` che indica se l'utente ha ottenuto un oggetto `Achievement` specifico.

Metodi:

- `isAchieved`: è un metodo di tipo `boolean` che ritorna il valore del parametro `is_achieved`
- `getId`: è un metodo di tipo void che ritorna l'intero corrispondente al campo id
- `getDescription`: è un metodo che ritorna la stringa corrispondente al campo `description`
- `setDescription`: è un metodo di tipo `void` che prende come argomento una stringa che verrà utilizzata per modificare il campo `description`



1.3 – Training

1.3.1 – Exercise

Questa classe presenta la composizione di ciascun esercizio disponibile all'interno della piattaforma. (Vedi RF 17)

Attributi:

- `name`: attributo di tipo `string` che contiene il nome dell'esercizio. Funge anche da identificativo univoco dell'esercizio.
- `reps_number`: attributo di tipo `int` che indica il numero di ripetizioni o il tempo in isometria da eseguire nell'esercizio.

- `muscolar_group` : attributo di tipo `enum` che indica quale gruppo muscolare si sta allenando, in modo tale che il sistema suggerisca gli esercizi corretti a seconda del target da allenare.
- `rest_time` : attributo di tipo `Time` che contiene il tempo di riposo indicato tra le ripetizioni dell'esercizio.
- `ex_description` : attributo di tipo `string` che contiene la descrizione testuale dell'esercizio insieme a vari tips&tricks per una corretta esecuzione in modo da evitare problemi e lesioni.
- `sets_number`: attributo di tipo `int` che contiene il numero di set previsti dall'esercizio
- `set_state`: attributo di tipo `string` che identifica lo stato attuale del set dell'esercizio (`'in svolgimento'`, `'finito'`).

Metodi:

- `startSet`: metodo che avvia un set dell'esercizio, se ancora disponibile
- `getName`: metodo che permette di ottenere il valore dell'attributo "name"
- `getSetState`: metodo che permette di ottenere il valore dell'attributo "`set_state`"
- `getSetsNumber`: metodo che permette di ottenere il valore dell'attributo "`sets_number`"
- `getExDescription`: metodo che permette di ottenere il valore dell'attributo "`ex_description`"
- `getRestTime`: metodo che permette di ottenere il valore dell'attributo "`rest_time`"
- `getMuscolarGroup`: metodo che permette di ottenere il valore dell'attributo "`muscolar_group`"
- `getReps`: metodo che permette di ottenere il valore dell'attributo "`reps_number`"

1.3.2 - Training

La classe Training rappresenta l'allenamento, composto quindi da una serie di esercizi e diviso per gruppo muscolare. (Vedi RF 17)

Attributi:

- `exercises` : attributo di tipo array di "`Exercise`" che corrisponde all'insieme di esercizi che compongono l'allenamento
- `muscolar_group` : attributo di tipo `enum` a cui corrisponde un determinato gruppo muscolare su cui si concentra l'allenamento

Metodi:

- `startRestTime` : metodo `void` che permette all'utente di segnare la fine dell'esercizio che stava svolgendo e al contempo di avviare il tempo di riposo previsto da esso.
- `startExercise` : metodo `void` che permette all'utente di iniziare lo specifico esercizio richiesto.
- `skipExercise` : metodo `void` che permette all'utente di saltare lo specifico esercizio indicato.
- `skipSet` : metodo `void` che permette all'utente di saltare il set intero dello specifico esercizio indicato.
- `skipRestTime` : metodo `void` che permette all'utente di saltare il tempo di riposo rimanente in caso non ne avesse la necessità.

1.3.3 - TrainingSchedule

Questa classe rappresenta la scheda di allenamento vera e propria che viene assegnata all'utente ed è costituita da un insieme di oggetti di tipo `Training`. (Vedi RF 17-18)

Attributi:

- `schedule` : è un array di oggetti `Training` che va a comporre la scheda di allenamento

Metodi:

- `createSchedule` : è il metodo di tipo `void` che assegna allo specifico utente una scheda di allenamento creata ad hoc per lui in base agli obiettivi ed esigenze impostate.

1.3.4 – Photo

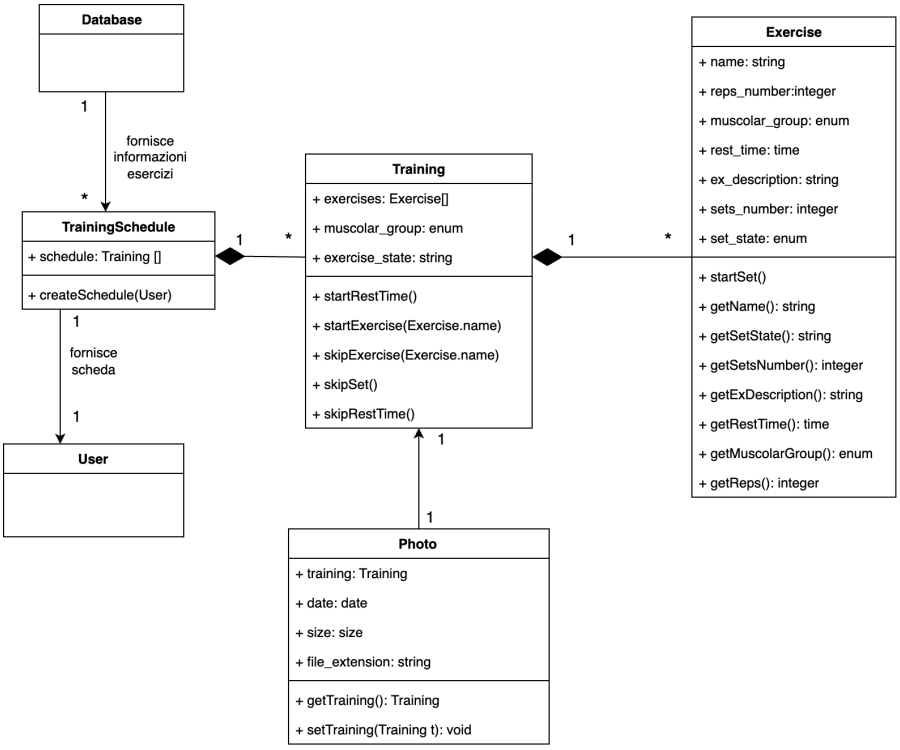
Questa classe definisce la struttura delle fotografie post allenamento che l'utente può caricare sul sistema per tenere traccia dei propri progressi. (Vedi RF 30)

Attributi:

- `training` : attributo di tipo `Training` contenente l'allenamento specifico a cui è riferita la fotografia.
- `date`: attributo di tipo `Date` indicante la data in cui è stata caricata la fotografia
- `size` : attributo di tipo `Size` indicante la dimensione del file contenente la fotografia.
- `file_extension`: attributo di tipo `string` indicante il tipo di formato del file contenente la fotografia.

Metodi:

- `getTraining`: metodo che permette di ottenere lo specifico allenamento correlato alla fotografia contenuto nell'attributo `Training`.
- `setTraining`: metodo che permette di assegnare la fotografia ad uno specifico allenamento impostando l'attributo `Training`.



1.4 - Dieting

1.4.1 - Food

La classe Food contiene tutte le caratteristiche inerenti agli alimenti contenuti all'interno del database in modo tale da permettere al sistema di generare una dieta equilibrata e che contiene una giusta quantità di macronutrienti.(Vedi RF 20-21)

Attributi:

- `food_name` : attributo di tipo `string` contenente il nome del cibo.
- `kcal`: attributo di tipo `int` indicante le kilocalorie contenute all'interno del cibo.
- `prots`: attributo di tipo `int` indicante le proteine contenute all'interno del cibo.
- `fats`: attributo di tipo `int` indicante i grassi contenuti all'interno del cibo.
- `carbs`: attributo di tipo intero indicante i carboidrati contenuti all'interno del cibo.
- `food_desc` : attributo di tipo `string` che elenca gli ingredienti utilizzati per preparare il cibo. (e.g. pasta al pomodoro = 150g pasta + 100g pomodoro).

Metodi:

- `getFoodName`: metodo che permette di ottenere il valore dell'attributo `food_name`.
- `getFoodDesc`: metodo che permette di ottenere il valore dell'attributo `food_desc`.
- `getKcal`: metodo che permette di ottenere il valore dell'attributo `kcal`.
- `getProts` : metodo che permette di ottenere il valore dell'attributo `prots`.
- `getCarbs` : metodo che permette di ottenere il valore dell'attributo `carbs`.
- `getFats` : metodo che permette di ottenere il valore dell'attributo `fats`.

1.4.2 - Meal

La classe `Meal` rappresenta un pasto e contiene tutti i cibi presenti all'interno di esso, oltre al suo nome. (Vedi RF 20-21)

Attributi:

- `food_arr`: attributo di tipo array di `Food`, contiene tutti i cibi presenti all'interno del pasto.
- `meal_name`: attributo di tipo `string` che contiene il nome del pasto (colazione, pranzo, cena) in modo tale che il sistema a seconda del nome del pasto prediliga certi cibi rispetto ad altri.

Metodi:

- `getFood`: metodo che permette di ottenere l'array di Food contenuto nell'attributo `food_arr`
- `getMealName`: metodo che permette di ottenere il valore dell'attributo `meal_name`

1.4.3 - Diet

La classe `Diet` corrisponde alla dieta che viene creata ad hoc per l'utente premium in base ai suoi obiettivi, caratteristiche ed esigenze.(Vedi RF 20)

Attributi:

- `meals`: attributo di tipo array di `Meal` contenente tutti i pasti che compongono la dieta

Metodi:

- `createDiet` : metodo di tipo `void` che assegna allo specifico utente premium un oggetto `Diet` creato appositamente per lui in conformità agli obiettivi che desidera raggiungere e al contempo tenendo in considerazione le caratteristiche fisiche attuali dell'utente.

1.4.4 - Contatore calorie

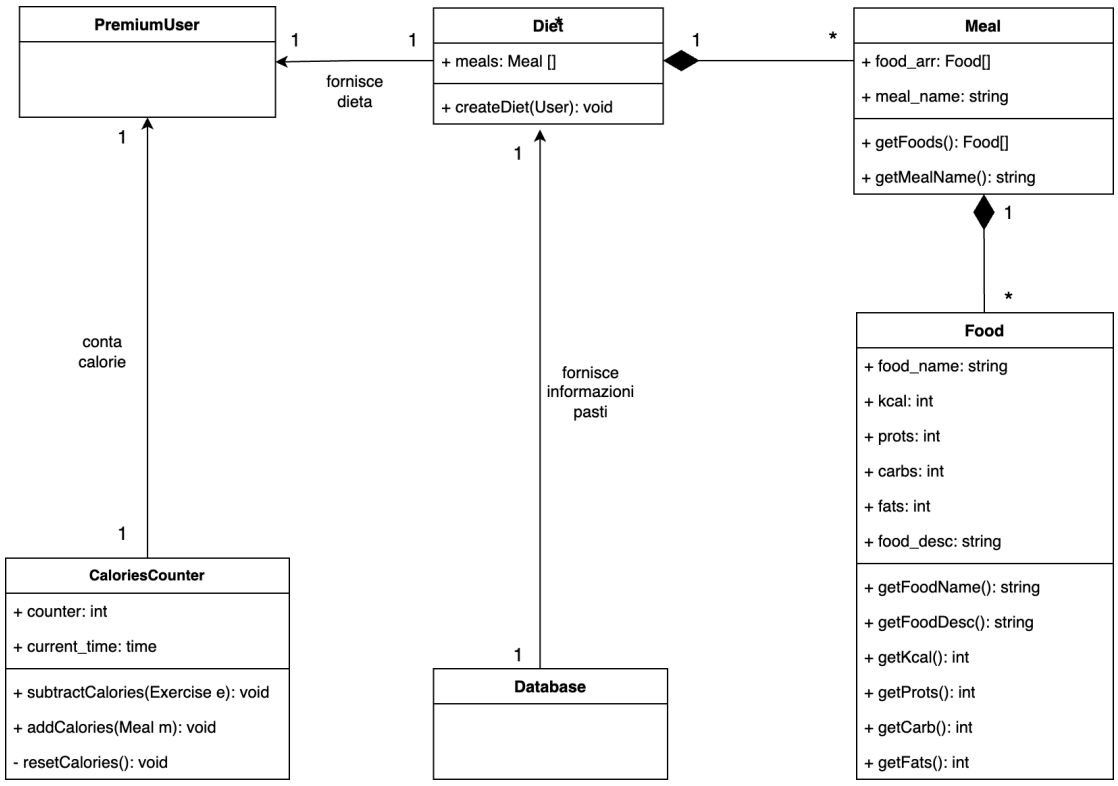
La classe `CaloriesCounter` consente all'utente premium di tenere monitorato in maniera costante l'insieme di calorie che l'utente ha consumato e assunto durante il giorno, in modo da capire se sta rispettando o meno la dieta a lui assegnata (vedi RF 22).

Attributi:

- `counter`, campo di tipo `int`, che viene aggiornato ogni qual volta l'utente premium svolge un esercizio o assume un pasto.
- `current_time`, campo di tipo `Time` che serve alla classe per essere a conoscenza dell'orario del giorno, in modo da rendersi conto se è necessario resettare il campo `counter`.

Metodi:

- `SubtractCalories`, metodo di tipo `void`, che sottrae al campo `counter` la quantità di calorie consumate durante l'allenamento che prende come argomento.
- `addCalories` è un metodo di tipo `void`, che invece somma al campo `counter` la quantità di calorie assunte con il pasto preso come argomento.
- `resetCounter` infine, è il metodo privato di tipo `void`, che imposta `counter` a `0` all'inizio di ogni giorno.



1.5 – Payments

1.5.1 – PaymentSystem

Questa classe si occupa di gestire il pagamento degli utenti che decidono di fare un upgrade alla versione premium. (Vedi RF 23-24)

Attributi:

- `id`: Questo è un attributo di tipo `int`. È l'id del pagamento che svolge ogni singolo utente. Serve in caso di necessità (ad esempio se un pagamento dovesse fallire si può verificare ricercando tramite questo campo l'effettivo esito del pagamento)
- `day`: Questo è un attributo di tipo `string`. È il giorno in cui viene effettuato il pagamento.
- `payment_data`: Questo è un attributo di tipo `PaymentData`. Contiene i dati di pagamento dell'utente.

Metodi:

- `pay`: Questo è un metodo di tipo `boolean` che prende in input un elemento di tipo `PaymentData`. Questo metodo restituisce `true` se il pagamento è andato a buon fine, altrimenti restituisce `false`.

1.5.2 - PaymentData

Questa classe serve a memorizzare i dati di pagamento degli utenti. Tutti gli attributi sono privati e mantenuti al sicuro all'interno del Database. (Vedi RF 23-24)

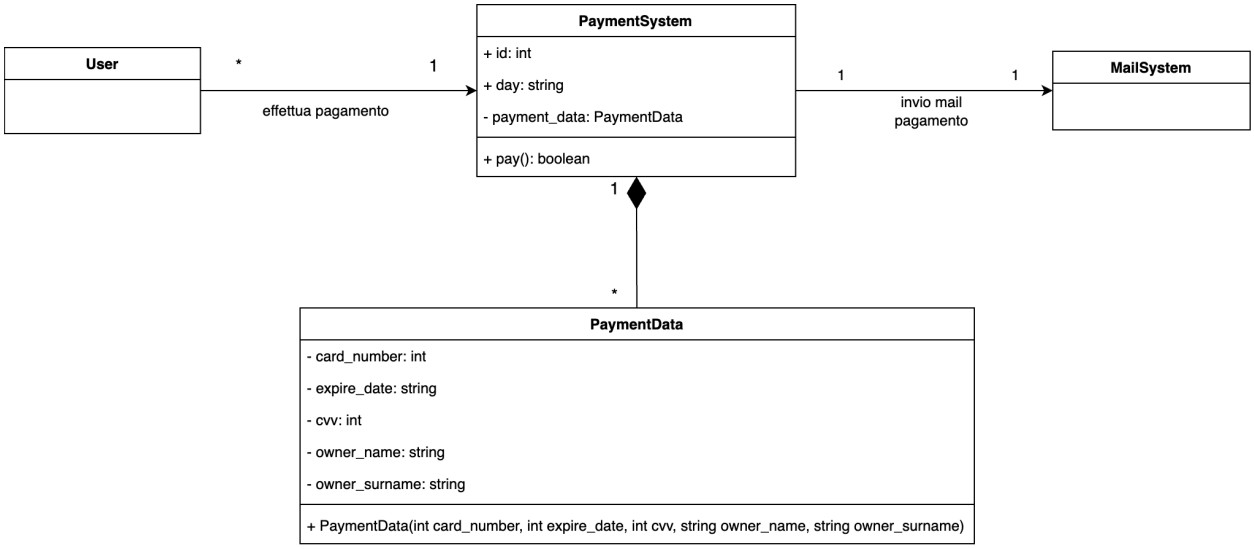
Attributi:

- `card_number`: Questo è un attributo di tipo `int`. Contiene il numero della carta dell'utente.
- `expire_date`: Questo è un attributo di tipo `string`. Contiene la data di scadenza della carta dell'utente.
- `cvv`: Questo è un attributo di tipo `int`. Contiene il cvv della carta dell'utente.
- `owner_name`: Questo è un attributo di tipo `string`. Contiene il nome dell'utente.
- `owner_surname`: Questo è un tipo di attributo di tipo `string`. Contiene il cognome dell'utente.

Metodi:

- `PaymentData`: Questo è il costruttore della classe. Prende in input:
  - `card_number`,
  - `expire_data`,
  - `cvv`,
  - `owner_name`,
  - `owner_surname`

  e crea un oggetto `PaymentData` con i dati inseriti come argomento.



1.6 - Mail

1.6.1 - MailSystem

Questa classe serve a mandare le mail relative alle credenziali e alla prima registrazione dell'utente.

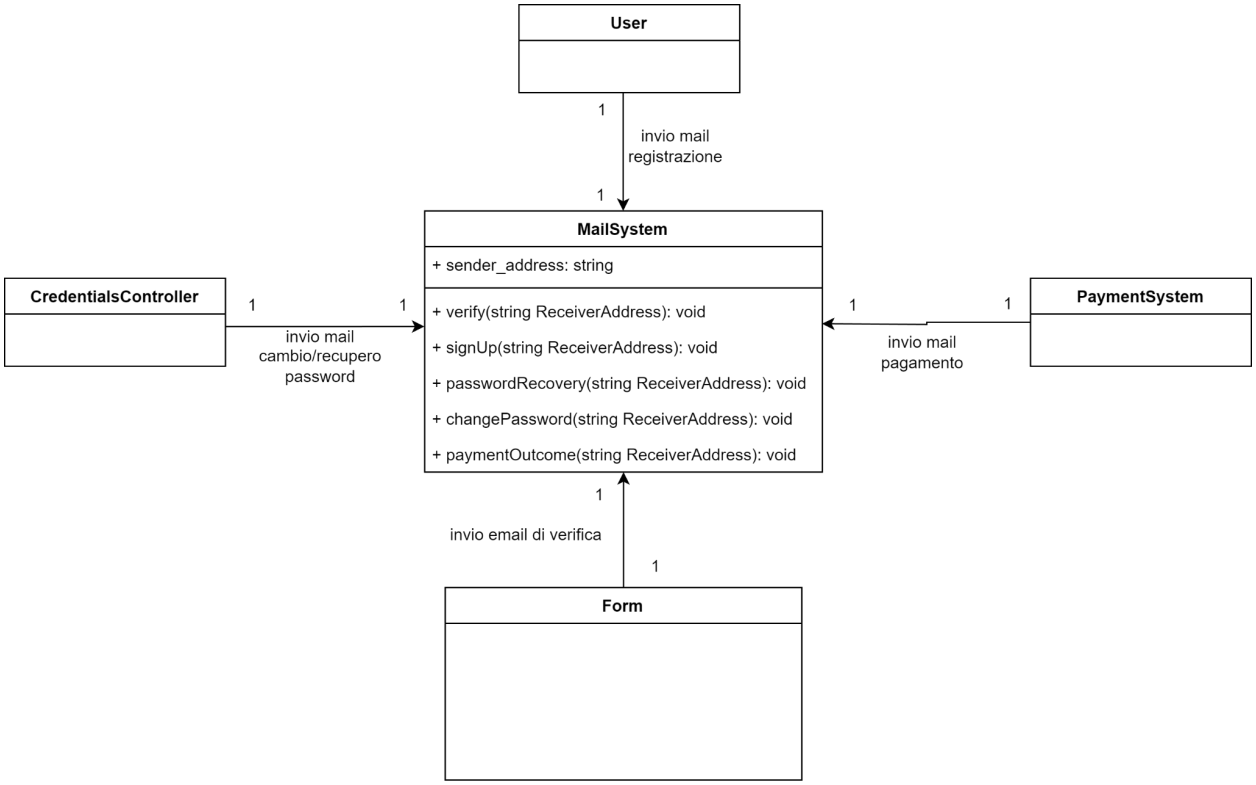
Attributi:

- `sender_address`: Questo è un attributo di tipo string e contiene l'indirizzo email ufficiale di FitGym44 (fitgym44@outlook.com).

Metodi:

- `verify` metodo di tipo void che prende come argomento un oggetto di tipo `string` che corrisponde alla email dell'utente. Questo metodo consiste nel inviare una mail all'utente quando sta effettuando la registrazione per verificare che l'email da lui inserita esista.
- `signUp`: Questo è un metodo di tipo `void` che prende come argomento una `string`. Questa è l'indirizzo dell'utente, serve a confermare la corretta registrazione dell'account all'applicazione.
- `passwordRecovery`: Questo è un metodo di tipo `void` che prende come argomento l'indirizzo dell'utente (`string`), serve in caso l'utente non ricordi la sua password e quindi desidera che gli venga modificata.

- `changePassword`: Questo è un metodo di tipo `void` che prende come argomento l'indirizzo dell'utente (`string`), serve in caso l'utente abbia necessità di cambiare la sua password.
- `paymentOutcome`: Questo è un metodo `void` che prende come argomento una `string`. Questa è l'indirizzo dell'utente, serve per comunicare all'utente l'esito del pagamento.



1.7 - Notifications

1.7.1 - NotificationSystem

Questa classe serve per mandare notifiche e gestire le preferenze relative ad esse. (Vedi RF 25)

Attributi:

- `notificationsList`: Questo è un array di tipo "`Notification`", contiene la lista di tutte le notifiche che possono arrivare a un utente.

Metodi:

- `getNotification`: Questo è un metodo che restituisce la lista delle preferenze scelte dall'utente relative alle notifiche.
- `addNotification`: Questo è un metodo void che prende come argomento una "`Notification`", grazie a questo metodo l'utente può aggiungere un ulteriore tipologia di notifica relativa a un "`Event`".
- `removeNotification`: Questo è un metodo `void` che prende come argomento una "`Notification`", grazie a questo metodo l'utente può rimuovere una notifica che non vuole più ricevere.

1.7.2 - Notification

Questa classe serve ad indicare le caratteristiche delle notifiche che riceve l'utente. (Vedi RF 25)

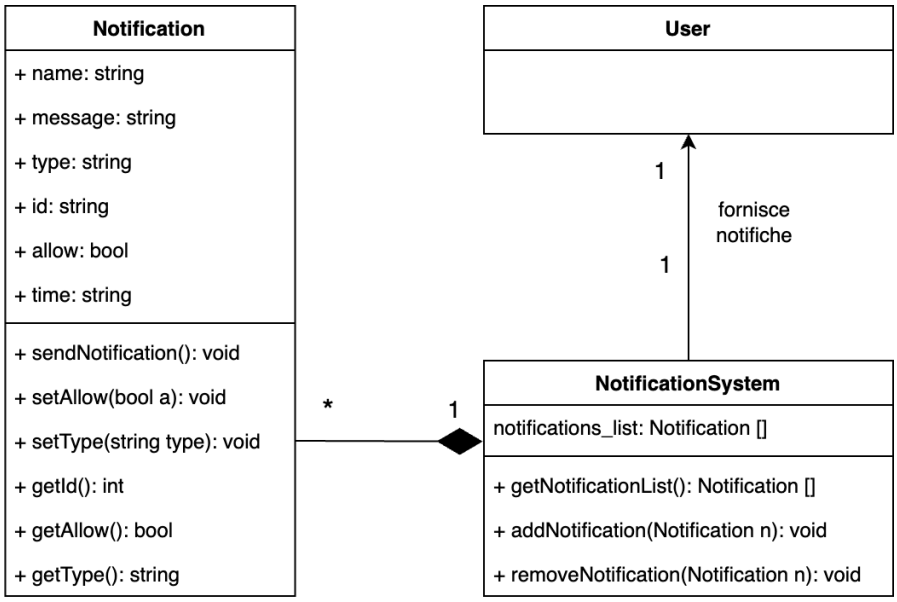
Attributi:

- `name`: Questo è un attributo di tipo `string`. Contiene il "titolo" della notifica.
- `message`: Questo è un attributo di tipo `string`. Contiene il messaggio effettivo della notifica.
- `id`: Questo è un attributo di tipo `int`. Contiene l'identificativo univoco della notifica.
- `allow`: Questo è un attributo di tipo `boolean`. Indica se l'utente ha acconsentito a ricevere questa notifica.
- `time`: Questo è un attributo di tipo `string`. Indica l'orario in cui è stata mandata la notifica.

Metodi:

- `sendNotification`: Questo è un metodo di tipo `void`. Serve a mandare la notifica all'utente.
- `setAllow`: Questo è un metodo di tipo `void`, prende in input un `boolean` che indica il permesso dell'utente alla ricezione o non ricezione della notifica.
- `getId`: Metodo che ritorna un `int`. Questo è l'identificativo della notifica.

- `getAllow`: Metodo che ritorna un `boolean`. Questo è il permesso dell'utente a ricevere o meno la notifica.



1.8 - Calendar

1.8.1 - Calendar

Questa classe contiene un array di giorni, serve a gestire gli eventi. (Vedi RF 29)

Attributi:

- `days`: Questo attributo è un array di "`Day`".

1.8.2 - Day

Un oggetto della classe "`Calendar`" è composto da n elementi della classe "`Day`", quest'ultima serve a contenere gli oggetti della classe "`Event`". (Vedi RF 29)

Attributi:

- `events`: Questo attributo è un array di "`Event`", contiene la lista di tutti gli eventi presenti in un determinato giorno.

- `name`: Questo attributo è di tipo `string` ed indica il nome del giorno (Lunedì, Martedì, ...).
- `is_festive`: Questo è un attributo di tipo `boolean` e indica se il giorno è festivo o meno.

Metodi:

- `addEvent`: Questo è un metodo di tipo `boolean` che prende come argomento un `Event`. Serve ad aggiungere un oggetto della `Event` all'array `events`. Ritorna `true` se l'oggetto è stato aggiunto correttamente, altrimenti `false`.
- `removeEvent`: Questo è un metodo di tipo `void` che prende come argomento un `Event`. Serve a rimuovere un oggetto `Event` dall'array `events`.
- `isFestive`: Questo è un metodo di tipo `bool`. Serve a verificare se un determinato giorno è festivo o meno. Ritorna `true` se lo è, `false` se non lo è.
- `getEvents`: Questo è un metodo di tipo `void`. Serve a stampare a schermo tutte le `description` degli oggetti `Event` presenti dentro `events`.
- `getName`: Questo è un metodo di tipo `string`. Serve a stampare il nome di uno specifico oggetto della classe `Day`.

1.8.3 - Event

Questa classe rappresenta i vari allenamenti che l'utente può aggiungere autonomamente all'interno di un giorno. (Vedi RF 29)

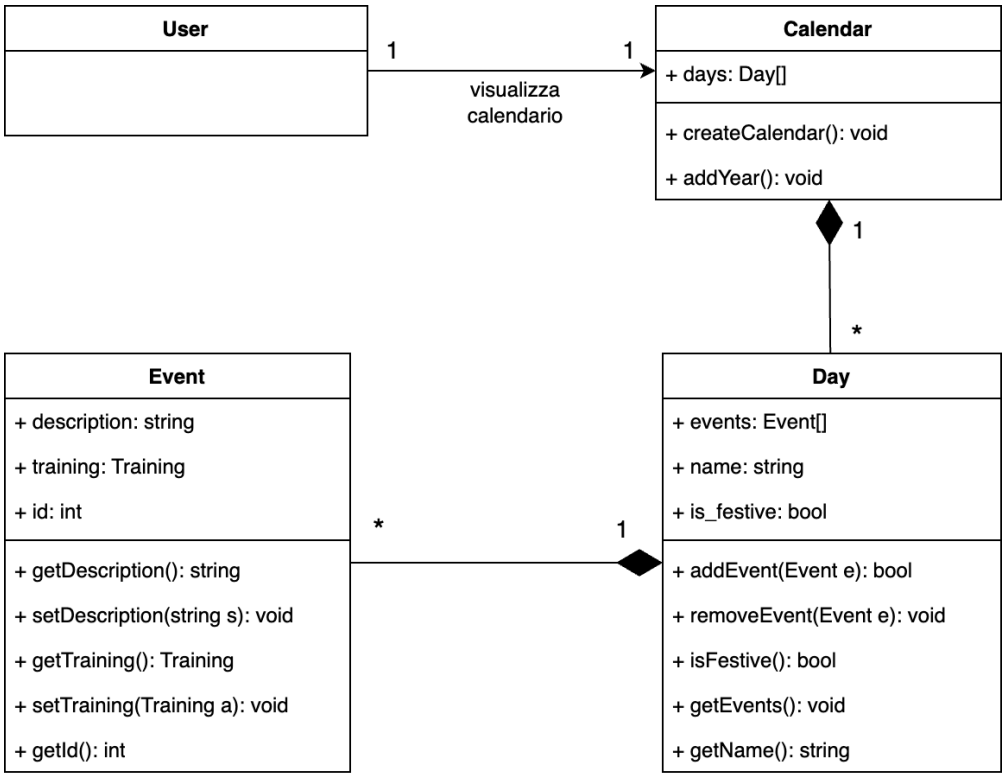
Attributi:

- `description`: Questo è un attributo di tipo `string`. Indica all'utente cosa si è prefissato di fare.
- `training`: Questo è un attributo di tipo `Training`. Indica all'utente l'allenamento che ha scelto di fare nel giorno dell'evento.
- `id`: Questo è un attributo di tipo `int`. Identifica univocamente l'evento.

Metodi:

- `getDescription`: Questo è un metodo di tipo `string`. Ritorna la `description`.
- `setDescription`: Questo metodo prende come argomento una `string` e la setta come valore della descrizione.

- `getTraining`: Questo è un metodo di tipo `Training`. Ritorna il valore dell'attributo `training`.
- `getId`: Questo è un metodo di tipo `int`. Ritorna l'`id` di questo evento.



1.9 – Database

1.9.1 – Database

Questa classe si occupa delle interazioni dell'applicazione con il Database vero e proprio. Di seguito elenco attributi e metodi:

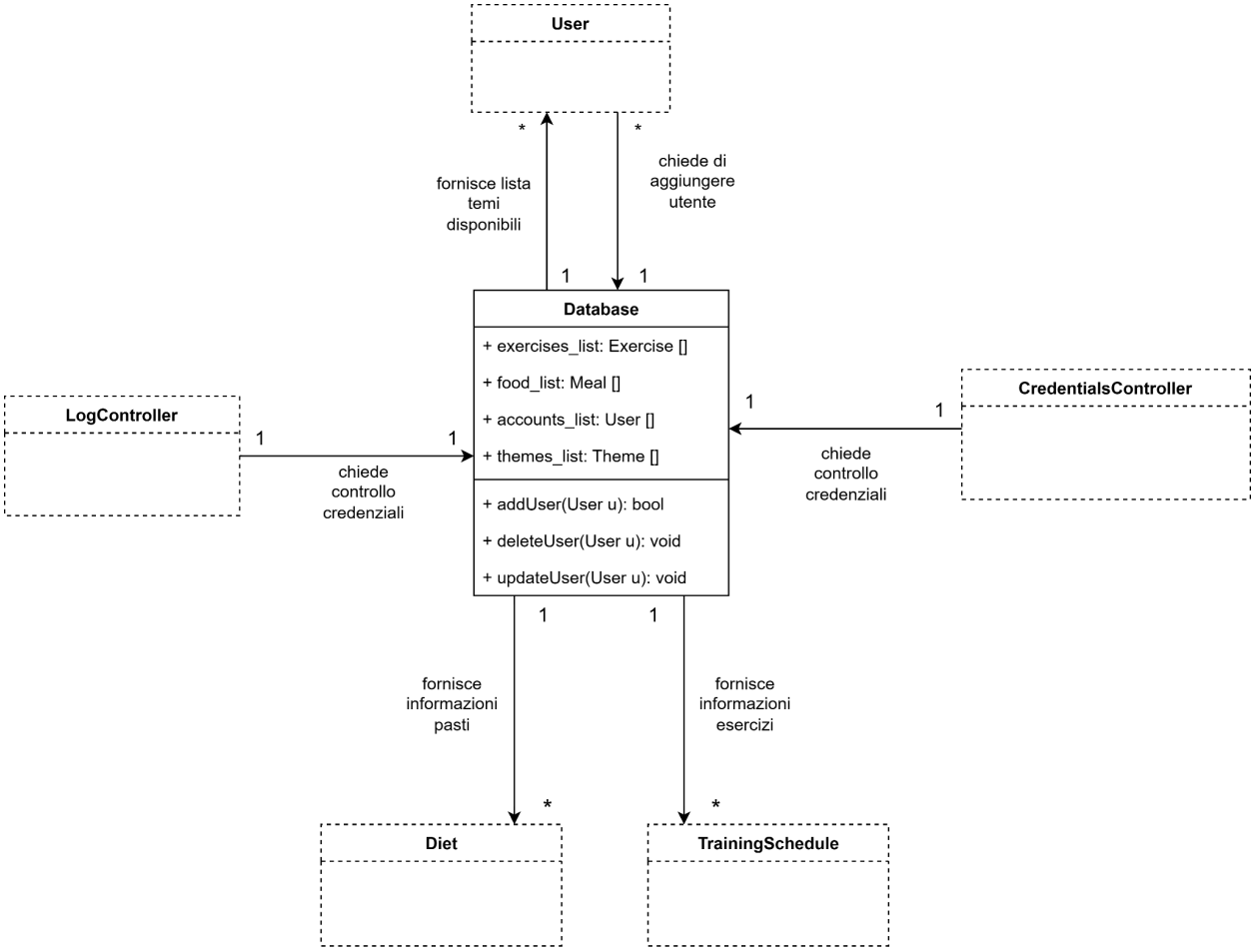
Attributi:

- `exercises_list`: Questo è un array di tipo `Exercise`. È ottenuto facendo una richiesta al Database degli esercizi contenuti al suo interno. (Vedi RF 17-18-19)

- `food_list`: Questo è un array di tipo `Food`. È ottenuto facendo una richiesta al Database dei cibi contenuti al suo interno, in output verranno forniti anche i relativi valori nutrizionali. (Vedi RF 20-21-22)
- `accounts_list`: Questo è un array di tipo `User`. È ottenuto facendo una richiesta al Database degli utenti con i relativi identificativi a loro associati (`email` e `username`) e le loro preferenze e statistiche (ad esempio che `Theme` preferiscono). I valori personali come le password vanno mantenute al sicuro, questa classe non ne ha bisogno, quindi non ne richiede.
- `themes_list`: Questo è un array di tipo `Theme`. È ottenuto facendo richiesta al Database dei temi presenti all'interno dell'applicazione. (Vedi RF 16)

Metodi:

- `addUser`: Questo è un metodo `boolean` che prende in input un oggetto della classe `User`. Si occupa di aggiungere un utente e i suoi relativi dati al Database. Ritorna `true` se tutto è stato effettuato correttamente o `false` se sono avvenuti errori (ex: indirizzo e-mail già utilizzato). (Vedi RF 4)
- `deleteUser`: Questo è un metodo `void` che prende come argomento un `User`. Serve ad eliminare dal Database un utente che ne fa richiesta. (Vedi RF 7)
- `updateUser`: Questo è un metodo `void` che prende come argomento un `User`. Serve ad aggiornare le preferenze dell'utente (ad esempio il cambio di tema). Una volta cambiati i valori di alcuni variabili all'interno dell'utente verranno inviati al Database gli aggiornamenti e i campi da cambiare al suo interno.



1.10 - Custom Datatype

1.10.1 - Theme

Questa classe rappresenta il tema scelto dall'utente premium per personalizzare l'aspetto dell'applicazione. Di seguito elenco attributi e metodi:

Attributi:

- `name`: Questo attributo è di tipo `string` e rappresenta il nome del tema.
- `bg_color`: Questo attributo è di tipo `string` e definisce il colore di sfondo del tema.
- `lateral_color`: Questo attributo è di tipo `string` e indica il colore laterale del tema.

- `font_color`: Questo attributo è di tipo `string` e specifica il colore del font utilizzato nel tema.

Metodi:

- `getName()`: Questo è un metodo che restituisce una `string`, rappresentante il nome del tema.
- `display(string name)`: Questo è un metodo `void` che prende in input una `string` (nome) e mostra il tema selezionato con il nome specificato.

1.10.2 – Size

Questa classe rappresenta la dimensione delle foto/immagini che possono essere caricate dall'utente o che fanno parte dell'applicazione. Di seguito elenco attributi e metodi:

Attributi:

- `kbyte`: Questo attributo è di tipo `float` e rappresenta la dimensione dell'immagine in kilobyte.

Metodi:

- `getSize()`: Questo è un metodo che restituisce un valore `float`, rappresentante la dimensione dell'immagine in kilobyte.

1.10.3 – Date

Questa classe rappresenta una data, utilizzata per segnare gli eventi nel calendario degli allenamenti. Di seguito elenco attributi e metodi:

Attributi:

- `day`: Questo attributo è di tipo `int` e rappresenta il giorno della data.
- `month`: Questo attributo è di tipo `int` e indica il mese della data.
- `year`: Questo attributo è di tipo `int` e specifica l'anno della data.

Metodi:

- `isValid()`: Questo è un metodo che restituisce un valore `bool`, indicando se la data è valida.
- `getDate()`: Questo è un metodo che restituisce un array di `int` rappresentante la data nel formato [giorno, mese, anno].

1.10.4 – Time

Questa classe rappresenta un intervallo di tempo, ad esempio il tempo di recupero tra una serie e l'altra di un allenamento. Di seguito elenco attributi e metodi:

Attributi:

- `minutes`: Questo attributo è di tipo `int` e rappresenta i minuti dell'intervallo di tempo.
- `seconds`: Questo attributo è di tipo `int` e indica i secondi dell'intervallo di tempo.

Metodi:

- `getTime()`: Questo è un metodo che restituisce un array di `int` rappresentante l'intervallo di tempo nel formato [minuti, secondi].

2. Codice e vincoli in OCL

Nel seguente capitolo, esploreremo gli OCL associati alle varie classi presenti nel sistema, offrendo una comprensione approfondita dei vincoli che governano le interazioni e le proprietà delle sue entità. Questi vincoli OCL sono stati definiti per garantire la coerenza e la correttezza del sistema.

2.1 – Vincoli OCL User and Credentials

2.1.1 – Vincoli OCL classe User

1. ***Vincolo sulla positività del valore dei passi fatti:***

   L'applicazione offre all'utente la possibilità di calcolare i passi fatti durante la giornata ricevendo informazioni da un'applicazione esterna e questo vincolo

impone che il campo che memorizza questo valore deve essere maggiore di zero.

```
context User

inv: self.steps_made > 0
```

2. ***Vincolo sulla modalità scelta:***

   Quando l'utente sceglie la modalità da applicare all'applicazione, viene settato il campo `choosen_mode` della classe `User` all'oggetto di tipo `Theme` corrispondente. Quest'ultimo può corrispondere solo a quello `'light'` o `'dark'`.

```
context User

inv: self.choosen_mode > 0
```

2.1.2 - Vincoli OCL classe Form

1. ***Vincoli sulla correttezza di nome e cognome:***

   Questi codici in OCL vanno a indicare che i campi `name`, `email` devono contenere obbligatoriamente soltanto dei caratteri alfanumerici.

```
context Form

inv: containsCaracters(self.name) = true


context Form

inv: containsCaracters(self.surname) = true
```

2. ***Vincolo sulla correttezza del formato della data di nascita:***

   Questo vincolo in OCL va a indicare che la data di nascita inserita deve obbligatoriamente possedere un formato adeguato.

```
context Form

inv: correctFormat(self.date) = true
```

3. ***Vincolo sulla positività dei dati sul fisico:***

Questa serie di vincoli in OCL precisa che i campi `weight`, `height`, `target`, `thighs`, `shoulders`, `waist` **e** `biceps` **devono essere maggiori di 0.**

```
context Form

inv: (self.weight > 0) and (self.height > 0) and (self.biceps > 
0) and (self.waist > 0) and (self.shoulders > 0) and 
(self.thighs > 0)
```

4. ***Vincolo sulla esistenza della email:***

Questa espressione in OCL specifica che la `email` **inserita durante la registrazione deve essere una** `email` **esistente**

```
context Form

inv: exist(self.email) = true
```

5. ***Vincolo sulla sicurezza della password:***

Questi codici in OCL vanno a specificare le restrizioni che sono imposte sul campo password per essere considerato sicuro: infatti devono contenere dei caratteri alfanumerici sia minuscoli che maiuscoli, deve essere maggiore uguale di 8 caratteri e contenere dei numeri.

```
context Form

inv: (containsUpperCase(self.password) = true) and 
(containsLowerCase(self.password) = true)


context Form

inv: self.password.size() >= 8


context Form
```

```
inv: containsNumber(self.password) = true
```

2.1.3 - Vincoli OCL classe PremiumUser

1. ***Vincoli sul limite della scadenza del pagamento:***

   Questa espressione OCL va ad indicare che il dominio del campo `countdown_payment` deve essere compreso tra 0 e 30.

   ```
   context PremiumUser

   inv: (countdown_payment >= 0) and (countdown_payment <= 30)
   ```

2. ***Vincoli sull'annullamento dell'abbonamento:***

   Quando l'utente decide di disdire il proprio abbonamento ritorna ad essere un utente normale e questo vincolo specifica che i campi `daily_calories` e `diet` devono essere uguali a 'null', mentre il campo `plan_type` uguale a 'normal'

   ```
   context PremiumUser::cancelSubscription(): void

   post: (daily_calories == null) and (diet == null) and
   (plan_type == 'normal')
   ```

3. ***Vincoli sull'annullamento dell'abbonamento:***

   Questo codice OCL specifica che prima di eseguire il metodo `paySubscription` è necessario che il campo `countdown_payment` deve essere 0 e successivamente deve risultare uguale a 30.

   ```
   context PremiumUser::paySubscribtion(): void

   pre: countown_payment = 0

   post: countown_payment = 30
   ```

2.1.4 - Vincoli OCL classe LogController

1. ***Vincoli sul settaggio del token della sessione:***

Questo vincolo in OCL va a indicare che prima dell'esecuzione del metodo `setToken` il risultato del metodo `findUser` deve essere 'true' e dopo il campo `session_token` del campo user all'interno della classe non deve essere `null`.

```
context LogController::setToken()

pre: findUser(string ie, string ip) = true

post: user.session_token <> null
```

2. ***Vincoli sulla chiusura della sessione:***

Questo codice OCL va a precisare che prima dell'esecuzione del metodo `logOut` il campo `session_token` del campo `user` all'interno della classe non deve essere null, inoltre impone che dopo la sua esecuzione il campo `session_token` dell'oggetto `user` e l'oggetto `user` devono essere uguali a `null`.

```
context LogController::logOut()

pre: user.session_token <> null

post: (user.session_token == null) and (user == null)
```

2.1.5 - Vincoli OCL classe CredentialController

1. ***Vincoli sul settaggio del proprietario delle credenziali:***

Questo vincolo OCL specifica che prima dell'esecuzione del metodo `setCredentialOwner` il campo `credential_owner` non deve essere `null`.

```
context CredentialController::setCredentialOwner(User u)

pre: credential_owner = null
```

2. ***Vincoli sulla individuazione del proprietario delle credenziali:***

Questa espressione OCL indica che prima dell'esecuzione del metodo `findUser` il campo `credential_owner` non deve essere `null`.

```
context CredentialController::findUser(string ie)

pre: credential_owner = null

post: credential_owner <> null
```

3. ***Vincoli sul recupero della password:***

   Quando viene effettuato il recupero della password è fondamentale che venga eseguito il metodo `findUser` precedentemente e che il suo risultato sia `true`. Successivamente l'esecuzione del metodo `passwordRecovery` il campo `credential_owner` deve essere uguale a `null`

```
context CredentialController::passwordRecovery()

pre: findUser(string ie) = true

post: credential_owner == null
```

4. ***Vincoli sul cambio della password:***

   Successivamente l'esecuzione del metodo `passwordRecovery` il campo `credential_owner` deve essere uguale a `null`

```
context CredentialController::passwordChange()

post: credential_owner == null
```

2.2 – Vincoli OCL Achievements

2.2.1 – Vincoli OCL classe Achievement

1. ***Vincolo sull'integrità dell'attributo id:***

   Questa espressione OCL garantisce che il campo `id` sia obbligatoriamente maggiore di zero.

```
context Achievement

inv: self.id > 0
```

2.2.2 - Vincoli OCL classe Badge

1. ***Vincolo correttezza assegnazione badge:***

   Questa espressione OCL afferma che se un badge è ottenuto (`is_obtained` è `true`), allora l'obiettivo associato (`achievement_required`) deve essere stato raggiunto (`is_achieved` è `true`). In altre parole, non può essere ottenuto un badge se l'obiettivo correlato non è stato raggiunto.

```
context Badge

inv NoObtainedWithoutAchievement:

self.is_obtained implies (self.achievement_required.is_achieved
= true)
```

2. ***Vincoli corretta definizione della classe badge:***

   Questi vincoli definiscono i requisiti che devono essere soddisfatti prima e dopo l'esecuzione dei metodi `setDescription` e `addBadge` per garantire la consistenza dell'oggetto `Badge` e delle relazioni con altri oggetti, come gli utenti che possiedono i `badge`.

```
context Badge::setDescription(newDescription : String)

pre: newDescription <> null and not newDescription.isEmpty()

post: self.description = newDescription


context Badge::addBadge(user : User)

pre: self.is_obtained = true
```

2.3 - Vincoli OCL Trainings

2.3.1 - Vincoli OCL classe Exercise

1. ***Vincolo integrità del numero di ripetizioni:***

Garantisce che il numero di ripetizioni sia sempre un valore positivo.

```
context Exercise

inv PositiveRepsNumber:

    reps_number > 0
```

2. ***Vincolo consistenza del numero di set:***

   Garantisce che il numero di Sets sia sempre un valore positivo.

```
context Exercise

inv PositiveSetsNumber:

    sets_number > 0
```

2.3.2 - Vincoli OCL classe Training

1. ***Vincolo sulla consistenza degli esercizi:***

   Questo vincolo garantisce che non esista un'istanza della classe `Training` che non abbia esercizi.

```
context Training

inv NonEmptyExercises:

    exercises->notEmpty()
```

2.3.3 - Vincoli OCL classe TrainingSchedule

1. ***Vincolo sulla corretta creazione di una scheda di allenamento:***

   Questo vincolo assicura che una nuova scheda di allenamento sia creata solo se è stato specificato un utente. Dopo la creazione della scheda di allenamento, l'array di allenamenti non è vuoto.

```
context TrainingSchedule::createSchedule(user : User)

pre: user <> null
```

```
post: schedule->notEmpty()
```

2.3.4 - Vincoli OCL classe Photo

1. ***Vincolo per il rispetto delle dimensioni:***

   Questo vincolo verifica che la dimensione della fotografia, oltre a non essere negativa, non superi i 10MB come richiesto dal RF30

```
context Photo

inv SizeInRange:

      self.size >= 0 and self.size <= 10 * 1024 * 1024
```

2. ***Vincolo per il rispetto dell'estensione:***

   Questo vincolo garantisce che siano rispettati i requisiti riguardo il formato del file definiti dal RF30

```
context Photo

inv FileExtensionValid:

self.file_extension = 'PNG' or self.file_extension = 'JPG' or
self.file_extension = 'JPEG'
```

2.4 - Vincoli OCL Dieting

2.4.1 - Vincoli OCL classe Food

1. ***Vincolo sull'integrità del nome:***

   Questo vincolo verifica che il nome del cibo non sia vuoto.

```
context Food
```

```
inv FoodNameNotEmpty:
      self.food_name <> ''
```

2. ***Vincoli sulla consistenza dei valori nutrizionali:***

   Queste espressioni OCL verificano che la quantità di kilocalorie, proteine, grassi e carboidrati di ciascun cibo non sia negativa.

```
context Food
inv KcalNonNegative:
      self.kcal >= 0
inv ProtsNonNegative:
      self.prots >= 0
inv FatsNonNegative:
      self.fats >= 0
inv CarbsNonNegative:
      self.carbs >= 0
```

2.4.2 - Vincoli OCL classe Meal

1. ***Vincolo sull'integrità della lista dei cibi:***

   Questo vincolo OCL verifica che l'array dei cibi non sia vuoto.

```
context Meal
inv FoodArrayNotEmpty:
      not self.food_arr->isEmpty()
```

2.4.3 - Vincoli OCL classe Diet

1. ***Vincolo sull'integrità della lista dei pasti:***

   Questo codice OCL verifica che l'array di pasti non sia vuoto.

```
context Diet

inv MealsNotEmpty:

      not self.meals->isEmpty()
```

2.4.4 - Vincoli OCL classe CaloriesCounter

1. ***Vincoli sullo stato dell'allenamento:***

   Questo vincolo in OCL precisa che prima dell'esecuzione del metodo `substractCalories` **il campo** `set_state`**, dell'oggetto di tipo** `Exercise` **passato come argomento della funzione, deve essere uguale alla stringa 'finito'.**

```
context CaloriesCounter::substractCalories(Exercise e)

pre: e.set_state = 'finito'
```

2. ***Vincoli sul reset delle calorie:***

   Questa espressione in OCL specifica che prima dell'esecuzione del metodo `resetCalories` **l'orario corrente del giorno deve essere mezzanotte. Mentre successivamente l'esecuzione dell'metodo il campo** `counter` **deve essere uguale a 0.**

```
context CaloriesCounter::resetCalories()

pre: current_time = '00:00:00'

post: counter = 0
```

2.5 - Vincoli OCL Payments

2.5.1 - Vincoli OCL classe PaymentSystem

3. ***Vincolo sull'unicità dell'ID del pagamento:***

   L'Id identifica univocamente un pagamento, c'è quindi bisogno che questo non venga ripetuto molteplici volte in pagamenti diversi tra loro. Questo vincolo assicura che ogni pagamento abbia un ID unico all'interno del sistema.

```
inv UniquePaymentId:

  PaymentManager.allInstances()->isUnique(id)
```

4. ***Vincolo di validità sul giorno di pagamento:***

   Il giorno di pagamento deve essere esistente. Questo vincolo verifica che la stringa del giorno del pagamento non sia vuota e che segua il formato "YYYY-MM-DD".

```
inv ValidPaymentDay:

day->notEmpty() and day.size() = 10 and
day.matches('[0-9]{4}-[0-9]{2}-[0-9]{2}')
```

5. ***Vincolo sulla presenza dei dati di pagamento:***

   I dati di pagamento devono essere validi e riconducibili a una carta esistente. Questo vincolo assicura che ogni pagamento abbia associati dati di pagamento validi.

```
inv PaymentDataPresent:

payment_data->notEmpty()and
payment_data.oclIsTypeOf(PaymentData)
```

6. ***Vincolo sul risultato del metodo di pagamento:***

   Questo vincolo assicura che il metodo 'pay' restituisca "true" solo se i dati di pagamento sono validi e l'importo è maggiore di 0. Inoltre, verifica che l'ID del pagamento sia incrementato correttamente e che il giorno sia impostato correttamente alla data corrente.

```
context PaymentManager::pay(paymentData: PaymentData):

  post:

     result = (paymentData.isValid() and paymentData.amount > 0)
implies

       (self.id = PaymentManager.allInstances()->size() + 1 and

        self.day = Date.today().toString('yyyy-MM-dd'))
```

2.5.2 - Vincoli OCL classe PaymentData

1. ***Vincolo sull'inizializzazione dei dati di pagamento***

   Questo vincolo assicura che i parametri passati al costruttore rispettino le condizioni di lunghezza e formato previste, e che i dati di pagamento vengano inizializzati correttamente.

```
context PaymentData::PaymentData(cardNumber: Integer,
expireDate: String, cvv: Integer, ownerName: String,
ownerSurname: String):

  pre: cardNumber.toString()->size() = 16 and

       expireDate.matches('[0-9]{2}/[0-9]{2}') and

       cvv.toString()->size() = 3 and

       ownerName->notEmpty() and ownerSurname->notEmpty()

  post:

    self.card_number = cardNumber and

    self.expire_date = expireDate and

    self.cvv = cvv and

    self.owner_name = ownerName and

    self.owner_surname = ownerSurname
```

2.6 - Vincoli OCL Mails

2.6.1 - Vincoli OCL classe MailSystem

1. ***Vincolo sull'indirizzo email del mittente:***

   Questo vincolo assicura che l'indirizzo email del mittente sia quello ufficiale di FitGym44.

```
inv ValidSenderAddress:

  sender_address = 'fitgym44@outlook.com'
```

2. ***Vincolo sull'indirizzo email dell'utente:***

   Questo vincolo assicura che l'indirizzo email fornito durante la registrazione abbia un formato valido

```
context EmailManager::signUp(userEmailAddress: String):

                                                                    pre:
userEmailAddress.matches('[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-
zA-Z]{2,}')
```

2.7 - Vincoli OCL Notifications

2.7.1 - Vincoli OCL classe NotificationSystem

1. ***Vincolo sull'aggiunta delle notifiche:***

   Questo vincolo assicura che dopo l'aggiunta di una notifica questa sia presente nella lista delle notifiche.

```
context NotificationSystem::addNotification(notification:
Notification):

  pre: not notificationsList->includes(notification)

  post: notificationsList->includes(notification)
```

2. ***Vincolo sulla rimozione di una notifica:***

   Questo vincolo assicura che dopo la rimozione di una notifica, questa non sia più presente nella lista delle notifiche.

```
context NotificationSystem::removeNotification(notification:
Notification):

  pre: notificationsList->includes(notification)

  post: not notificationsList->includes(notification)
```

3. ***Vincolo sulla coerenza della lista di notifiche rispetto alle preferenze dell'utente:***

   Questo vincolo assicura che le notifiche presenti nella lista siano coerenti con le preferenze dell'utente, garantendo che solo le notifiche relative alle preferenze dell'utente siano presenti nella lista.

```
inv ConsistentPreferences:

  let userPreferences = self.getNotification() in

  userPreferences->forAll(p | notificationsList->exists(n |
n.event.type = p.type))
```

2.7.2 - Vincoli OCL classe Notification

1. ***Vincolo sulla lunghezza del nome della notifica:***

   Questo vincolo assicura che il nome della notifica abbia almeno 5 caratteri.

```
inv MinimumNameLength:

  name.size() >= 5
```

2. ***Vincolo sulla lunghezza massima del messaggio della notifica:***

Questo vincolo assicura che il messaggio della notifica non superi i 200 caratteri.

```
inv MaximumMessageLength:
  message.size() <= 200
```

3. ***Vincolo sull'unicità dell'ID della notfica:***

Questo vincolo assicura che ogni notifica abbia un ID univoco all'interno del sistema.

```
inv UniqueId:
  Notification.allInstances()->isUnique(id)
```

4. ***Vincolo sull'indicazione del permesso dell'utente:***

Questo vincolo assicura che dopo l'impostazione del permesso, lo stato dell'attributo "`allow`" sia `true`.

```
inv AllowTrue:
  self.allow = true
```

2.8 - Vincoli OCL Calendar

2.8.1 - Vincoli OCL classe Calendar

1. ***Vincolo sull'esistenza di giorni univoci:***
   Questo vincolo assicura che ogni giorno nell'array dei giorni sia unico, basato sulla data.

```
inv UniqueDays:
  self.days->isUnique(d | d.date)
```

2. ***Vincolo sull'ordinamento dei giorni:***

   Questo vincolo assicura che i giorni nell'array dei giorni siano ordinati in modo crescente in base alla data.

```
inv SortedDays:

  self.days->isOrderedBefore(a, b | a.date < b.date)
```

2.8.2 - Vincoli OCL classe Day

1. ***Vincolo sulla consistenza del nome del giorno:***

   Questo vincolo assicura che il nome del giorno sia valido e corrisponda a uno dei giorni della settimana.

```
inv ValidDayName:

  name = 'Lunedì' or name = 'Martedì' or name = 'Mercoledì' or
name = 'Giovedì' or name = 'Venerdì' or name = 'Sabato' or name
= 'Domenica'
```

2. ***Vincolo sull'aggiunta di un evento:***

   Questo vincolo assicura che dopo l'aggiunta di un evento, questo sia presente nell'array degli eventi.

```
context Day::addEvent(event: Event):

  pre: not events->includes(event)

  post: events->includes(event)
```

3. ***Vincolo aulla rimozione di un evento:***

   Questo vincolo assicura che dopo la rimozione di un evento, questo non sia più presente nell'array degli eventi.

```
context Day::removeEvent(event: Event):

  pre: events->includes(event)

  post: not events->includes(event)
```

2.8.3 - Vincoli OCL classe Event

1. ***Vincolo sull'unicità dell'ID dell'evento:***

   Questo vincolo assicura che ogni evento abbia un ID univoco all'interno del sistema.

```
inv UniqueId:

  Event.allInstances()->isUnique(id)
```

2. ***Vincolo sulla lunghezza minima della descrizione:***

```
inv MinimumDescriptionLength:

  description.size() >= 10
```

2.9 - Vincoli OCL Database

2.9.1 - Vincoli OCL classe Database

1. ***Vincolo sull'unicità degli utenti:***

   Quando un utente effettua la registrazione, c'è bisogno di verificare se questo sia effettivamente l'unico ad utilizzare una specifica mail. Questo vincolo assicura che non ci siano duplicati all'interno del database.

```
inv UniqueUsers:

  self.accounts_list->isUnique(u | u.email)
```

2. ***Vincolo sulla rimozione degli utenti:***

   Quando un utente decide di eliminare il suo account e non usare più FitGYM44 il suo account deve essere eliminato dall'attributo "accounts_list". Questo vincolo assicura che, dopo la cancellazione di un utente, esso non sia più presente all'interno del database.

```
context Database::deleteUser(user: User):

  pre: self.accounts_list->includes(user)

  post: not self.accounts_list->includes(user)
```

3. ***Vincolo sull'aggiornamento dell'utente:***

Quando un utente vuole aggiornare delle preferenze relative al suo account, c'è bisogno che queste preferenze vengano memorizzate all'interno del database e non vengano lasciati campi vuoti.

```
context Database::updateUser(user: User):

  pre: self.accounts_list->includes(user)

  post:

    user.preferences->notEmpty() and

                              user.preferences->forAll(p       |
user.preferences->oclIsTypeOf(Theme)) and

    user.preferences->forAll(p | self.themes_list->includes(p))
```