Relazione del progetto "Workit-out"

Programmazione ad Oggetti

Autori:
Salvatore Novelli
Lorenzo Belletti
Leonardo Castiglione
Diego Gessaroli

15 Febbraio 2026

Indice

Capitolo 1

Analisi

1.1 Descrizione e requisiti

Il software **Workit-out** si pone come obiettivo quello di realizzare un sistema focalizzato sul benessere dell'utente, in grado perciò di gestire l'assunzione e il consumo di calorie monitorando dinamicamente calorie, dieta e attività fisica attraverso l'inserimento quotidiano di dati.

Il sistema integra inoltre la presenza di una wiki, pensata per guidare l'utente con varie forme di informazione.

L'inserimento opzionale dei dati anagrafici, dati biometrici e obiettivo scelto in base alle preferenze dell'utente, saranno poi necessari all'applicazione per consigliare una dieta e un piano di allenamento adatto a raggiungere l'obiettivo. Se l'utente decide di non voler inserire i propri dati, l'applicazione userà dei dati preimpostati che si riferiscono a un utente medio.

Attraverso il monitoraggio alimentare invece, il programma si occupa della gestione dei pasti quotidiani, calcolando automaticamente la quantità di nutrienti necessari da assumere nei pasti della giornata.

Workit-out quindi, grazie ai dati, confronta l'andamento attuale con quello prefissato inizialmente dall'utente; le informazioni biometriche sono usate per la creazione della scheda personalizzata con esercizi e per il dispendio energetico.

L'applicazione prevede inoltre la presenza di una wiki (un'enciclopedia) che raccoglie varie informazioni sull'allenamento e che, anche tramite la presenza di video, fornisce una guida all'utente per permettere anche ai principianti di allenarsi responsabilmente e in sicurezza.

Requisiti funzionali

- Inserimento dei dati fisiologici dell'utente (età, altezza, peso, sesso, livello di attività) e di un obiettivo da raggiungere (perdita peso, mantenimento, aumento massa, ipertrofia).

- Preparazione di una scheda di allenamento settimanale con esercizi personalizzati in base all'obiettivo prefissato, con distinzione fra esercizi di forza e cardio.

- Gestione del diario alimentare giornaliero con calcolo dinamico dei macronutrienti (calorie, proteine, carboidrati, grassi) in relazione ai dati fisiologici.
- Consultazione di una wiki informativa con articoli e video relativi ad allenamento e alimentazione, con suggerimenti contestuali basati sul profilo dell'utente.

Requisiti non funzionali

- **Persistenza dei dati:** i dati dell'utente, lo storico alimentare e i piani di allenamento vengono salvati localmente in formato JSON e CSV, garantendo la permanenza delle informazioni tra sessioni successive.
- **Sistema di autenticazione:** tramite password per la protezione dell'account (tale funzionalità non è stata realizzata nella prima versione e sarà oggetto di futuri sviluppi).
- **Integrazione con la geolocalizzazione:** per il conteggio dei passi e della distanza percorsa (tale funzionalità richiede studi specifici sulle API di sistema e non è stata inclusa nella prima versione).

1.2 Modello del Dominio

Il dominio Workit-out è incentrato sulla figura dell'utente, caratterizzata da propri dati personali e dati biometrici che descrivono il suo stato fisico.

L'utente si pone un obiettivo specifico da raggiungere. Per monitorare il progresso verso tale fine, l'Utente compila un diario giornaliero, il quale aggrega i diversi pasti consumati durante il corso della giornata; ogni pasto è a sua volta composto da vari alimenti (`food`) contenenti diversi nutrienti (`nutrients`).

Parallelamente alla nutrizione, il dominio comprende anche la parte di attività fisica organizzata in piani di allenamento, composti da esercizi pianificati e loro consumo calorico basati sull'obiettivo.

Infine, l'Utente può consultare un'Enciclopedia (Wiki), intesa come un grande archivio di contenuti educativi quali articoli e video. Tali risorse potranno essere correlate con gli elementi del diario giornaliero e del piano di allenamento, fungendo da supporto per le attività dell'utente.

Gli elementi costitutivi del problema sono sintetizzati in Figura 1.1. Data la complessità del dominio la prima versione non comprenderà ricette o funzionalità di "Live Workout" con timer per i recuperi e spunta degli esercizi in tempo reale.

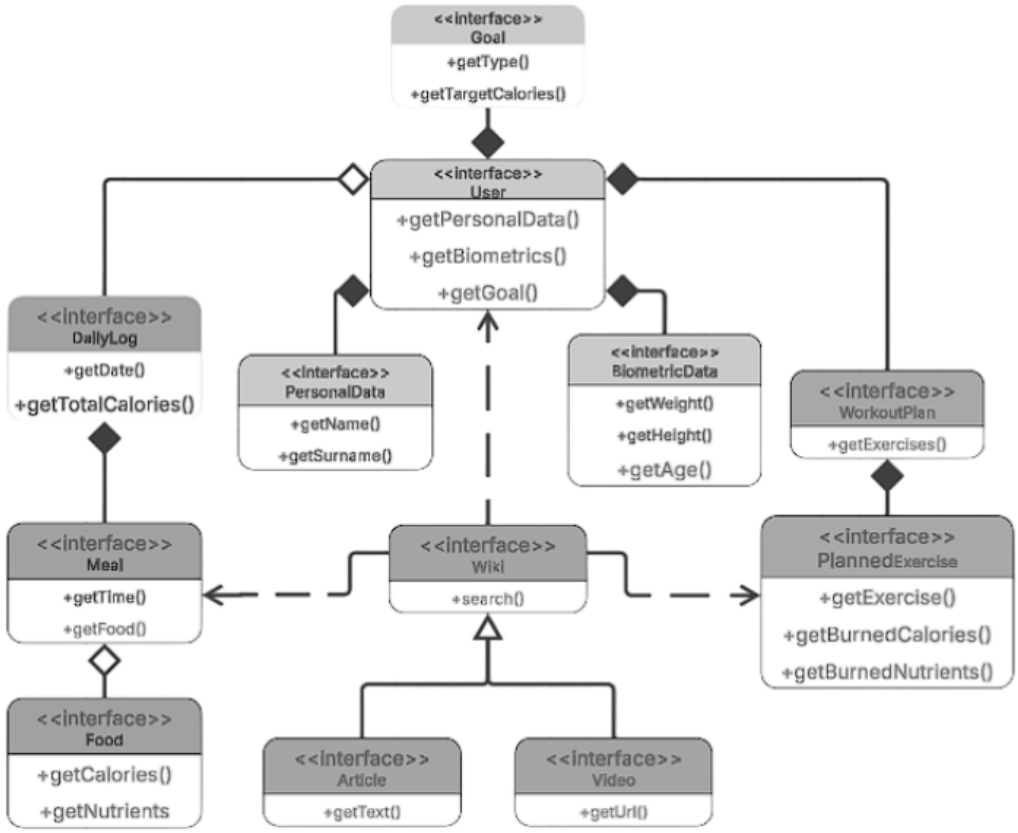

Figura 1.1: Schema UML dell'analisi del dominio, con rappresentate le entità principali ed i rapporti fra loro.

Capitolo 2

Design

2.1 Architettura

L'architettura di Workit-out segue il pattern architetturale **Model-View-Controller (MVC)**, con l'obiettivo di separare nettamente la logica applicativa dalla presentazione.

- Il **Model** rappresenta il nucleo logico dell'applicazione e contiene lo stato dei dati e la business logic: gestione dell'utente con calcolo di BMR e TDEE, repository degli alimenti e diario giornaliero con calcolo dei macronutrienti, generazione dei piani di allenamento, e wiki con suggerimenti contestuali. Il Model è totalmente indipendente dalla View, permettendone il riutilizzo in contesti differenti.
- Il **Controller** funge da intermediario fra View e Model. Il `MainController` svolge il ruolo di mediatore centrale, coordinando la comunicazione fra i controller dei singoli moduli (`UserProfileController`, `NutritionController`, `UserExerciseController`, `WikiController`). Ad esempio, le calorie bruciate dal modulo allenamento e i nutrienti registrati dal modulo nutrizione transitano entrambi per il `MainController`, che li inoltra al controller del profilo utente. Questa scelta evita dipendenze dirette fra i controller dei singoli moduli.
- La **View** costituisce l'interfaccia visualizzata dall'utente ed è priva di logica interna. Le view sono definite tramite interfacce (`NutritionView`, `UserProfileView`, `WikiView`, `PlanViewer`), il che consente la sostituzione delle implementazioni concrete senza che Controller o Model debbano subire alcuna modifica.

In Figura 2.1 è esemplificato il diagramma UML architetturale.

Sostituibilità della View: grazie alla separazione operata tramite interfacce, la View è definita come contratto; le implementazioni concrete possono essere sostituite senza che Controller o Model debbano subire alcuna modifica. L'unico punto in cui si potrebbe riscontrare un legame è l'uso di classi di utilità condivise per la navigazione tra le views, ma queste non contengono logica e quindi rispettano il pattern architetturale.

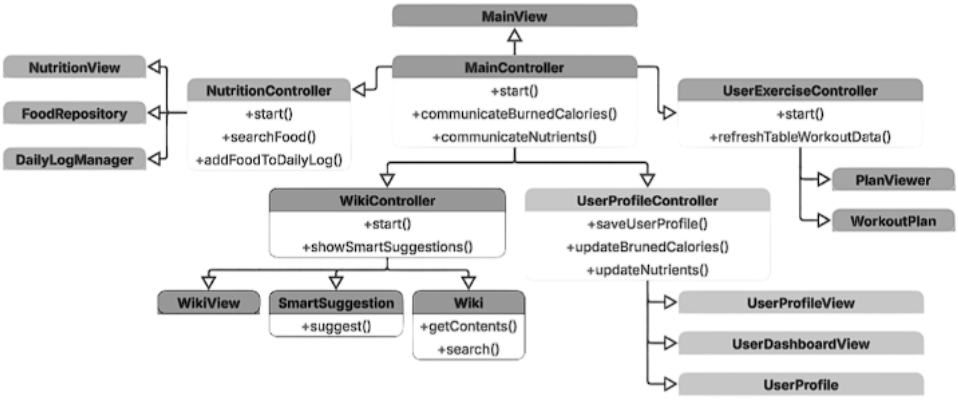

Figura 2.1: Schema UML architetturale di Workit-out, con le interfacce principali di Model, View e Controller e le relazioni fra loro.

2.2 Design dettagliato

In questa sezione si descrivono le scelte di design più significative, organizzate per problema affrontato e soluzione adottata.

2.2.1 Suggerimenti contestuali per la parte di wiki (Salvatore)

Problema: La wiki contiene numerosi contenuti (articoli e video). È necessario proporre all'utente quelli più rilevanti in base al suo obiettivo, agli esercizi del piano di allenamento e agli alimenti registrati nel diario.

Soluzione: È stato applicato il pattern **Strategy**: l'interfaccia funzionale `SmartSuggestion` definisce il contratto con un unico metodo `suggest(Wiki, UserProfile, List<Exercise>, Meal)`, e il `WikiControllerImpl` delega ad essa la logica di filtraggio senza conoscerne l'implementazione concreta. L'implementazione `SmartSuggestionImpl` estrae un insieme di tag dall'obiettivo dell'utente, dai nomi degli esercizi e dagli alimenti, quindi calcola uno score di rilevanza per ciascun contenuto della wiki e li restituisce ordinati per score decrescente.

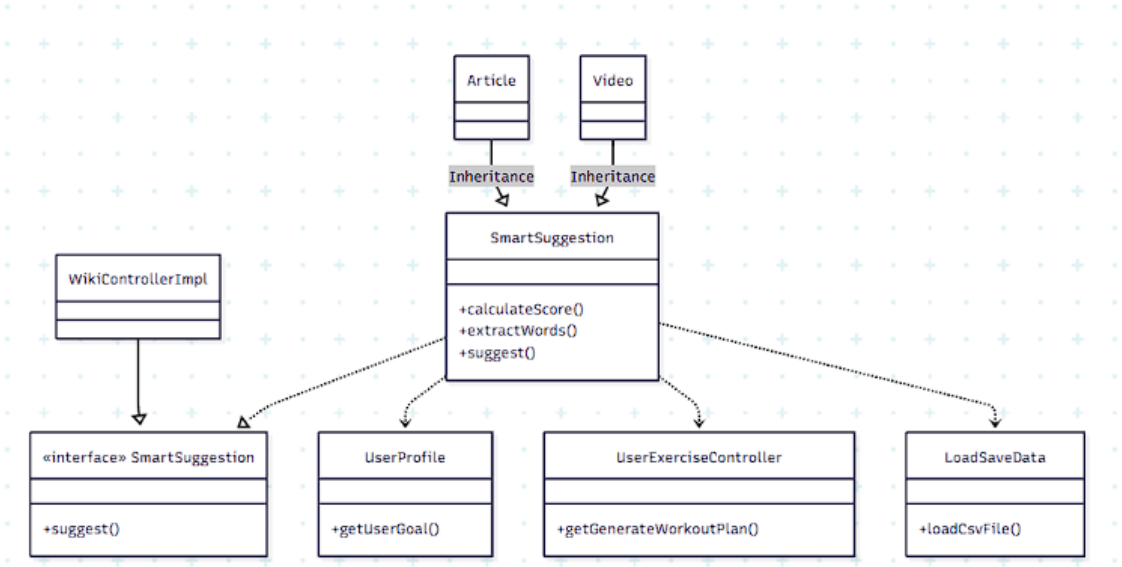

Figura 2.1: Rappresentazione UML del pattern Strategy per i suggerimenti contestuali della wiki.

2.2.2 Modellazione tipologia degli esercizi (Lorenzo)

Problema: Un sistema di allenamento deve poter gestire esercizi diversi tra loro, sia nella stessa categoria che non (come Strength e Cardio). Ognuno ha parametri personali (come serie/ripetizioni e durata/distanza), ma il sistema deve poterli trattare in modo uniforme per visualizzarli in una scheda o calcolare dati ad essi associati.

Soluzione: È stato implementato il pattern **Template Method**. La classe `AbstractPlannedExercise` definisce la struttura comune (nome, minuti, stato completamento), mentre il metodo astratto `getVolume()` viene implementato specificamente dalle sottoclassi: `StrengthPlannedExerciseImpl` utilizza `VolumeCalculator` per calcolare il volume (serie × ripetizioni × peso), mentre `CardioPlannedExerciseImpl` restituisce la distanza percorsa come valore di volume.

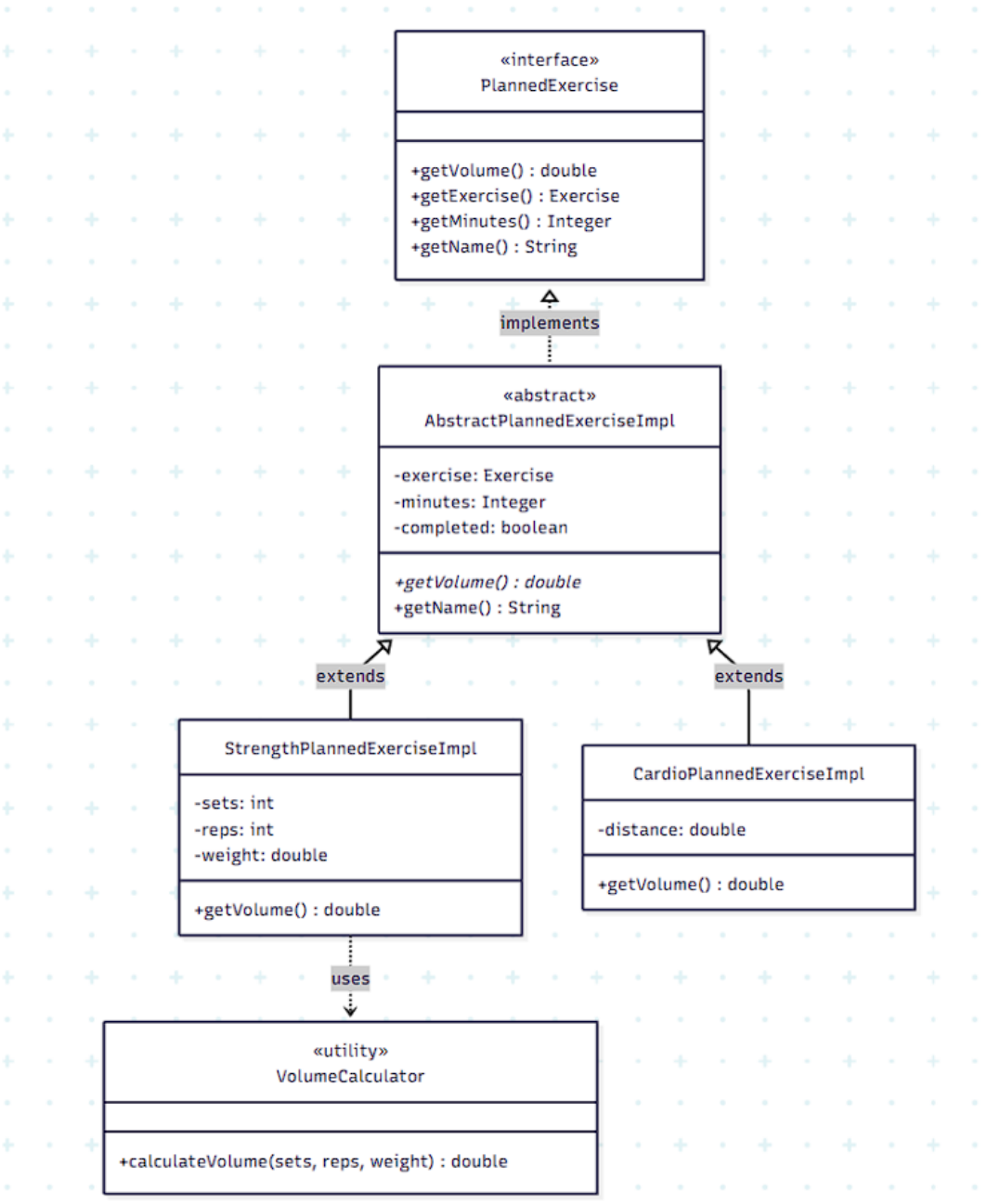

Figura 2.2: Rappresentazione UML del pattern Template Method per la gerarchia degli esercizi.

2.2.3 Astrazione della Gestione Nomi via Ereditarietà (Lorenzo)

Problema: Diversi componenti del sistema (`WorkoutPlan`, `WorkoutSheet`) necessitano di una gestione uniforme di un identificativo testuale (nome).

Soluzione: È stata creata la classe `NameFunction` che centralizza la gestione della stringa name. Facendo ereditare `WorkoutPlanImpl` e `WorkoutSheetImpl` da questa classe, si è garantita coerenza e ridotto la duplicazione di codice nel Model.

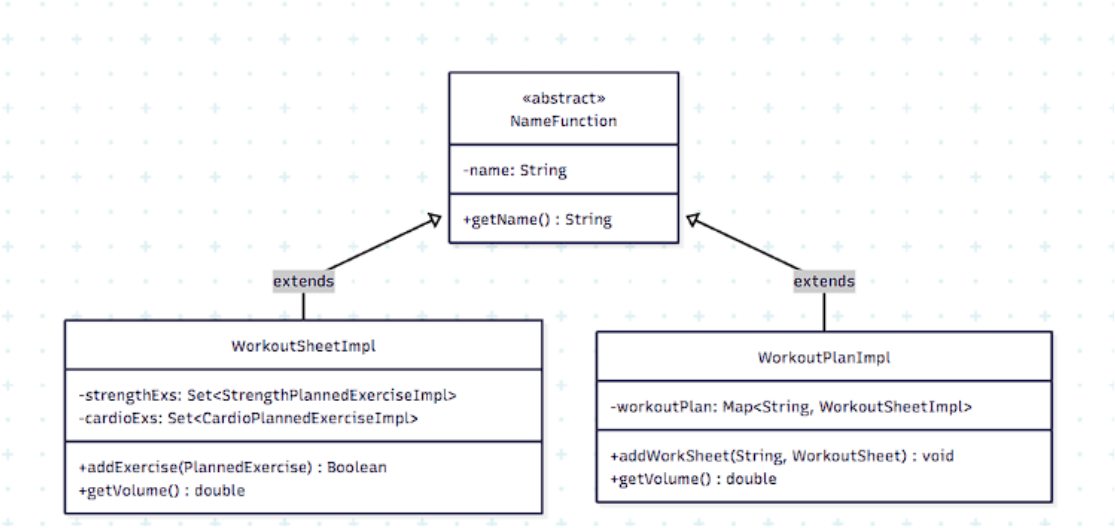

Figura 2.3: Rappresentazione UML della classe astratta NameFunction e delle classi che la estendono.

2.2.4 Creazione del piano (Lorenzo)

Problema: La creazione di un piano di allenamento completo è un'operazione che coinvolge diversi livelli: l'aggiunta di singoli esercizi (`PlannedExercise`) a una scheda giornaliera (`WorkoutSheet`), che a sua volta deve essere aggiunta in un programma settimanale (`WorkoutPlan`).

Soluzione: Con l'interfaccia `WorkoutCreator` (e la sua implementazione `WorkoutCreatorImpl`) che funge da coordinatore per la costruzione del modello. Questa classe permette di incapsulare la logica di creazione della gerarchia e garantire che ogni esercizio sia aggiunto al "foglio" (sheet) corretto prima che quest'ultimo venga inserito nel piano generale.

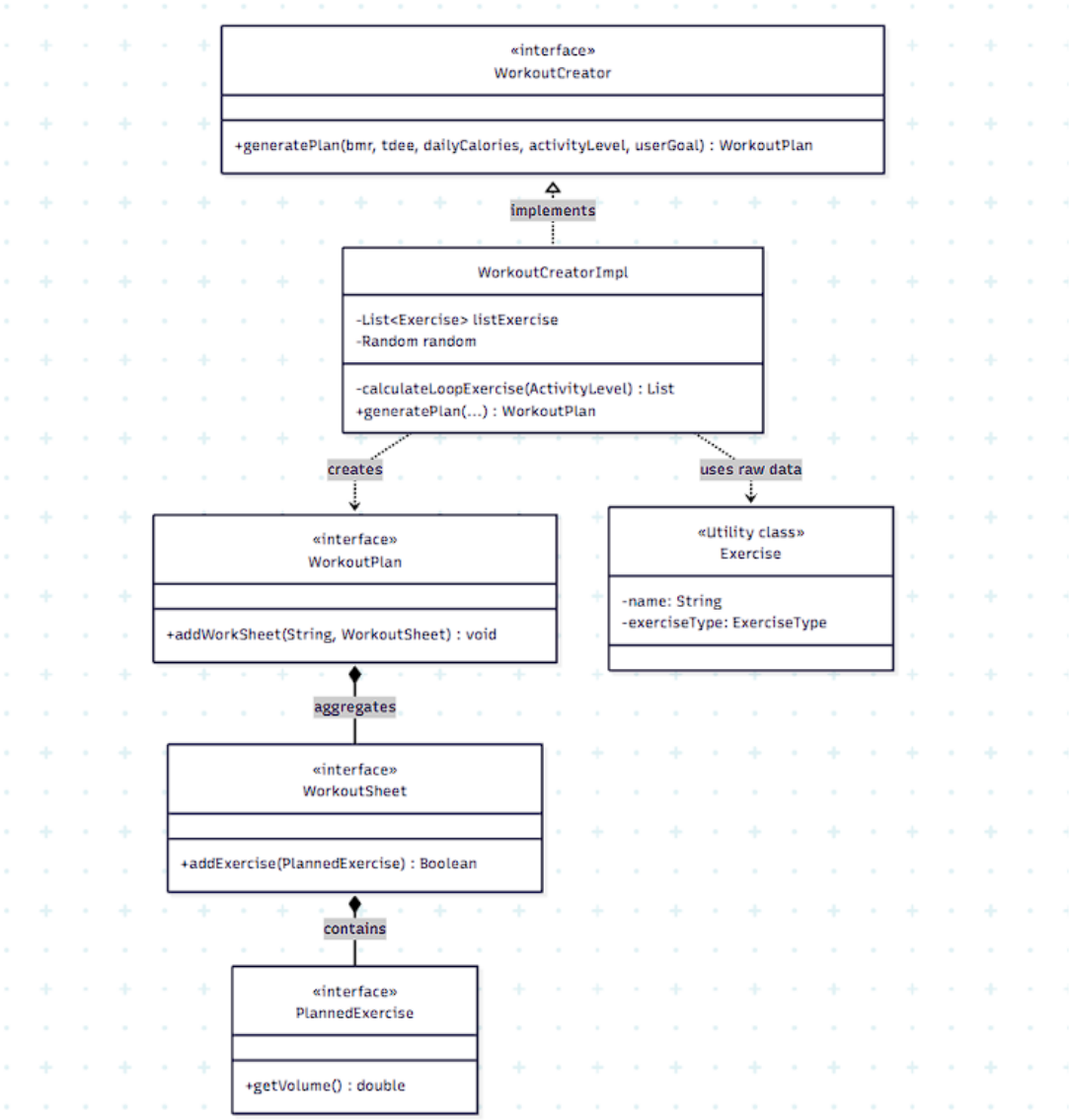

Figura 2.4: Rappresentazione UML della creazione del piano di allenamento tramite WorkoutCreator.

2.2.5 Utilizzo del Pattern Singleton (Lorenzo)

Problema: La comunicazione tra il `MainController` e la view del modulo esercizi necessita di un unico punto di accesso al controller.

Soluzione: È stato applicato il pattern **Singleton** alla classe `UserExerciseControllerImpl`, garantendo così una centralizzazione delle comunicazioni. Si riconosce che questa scelta comporta un accoppiamento più forte e limita la testabilità: un'alternativa sarebbe stata la dependency injection del controller, che avrebbe reso il sistema più flessibile.

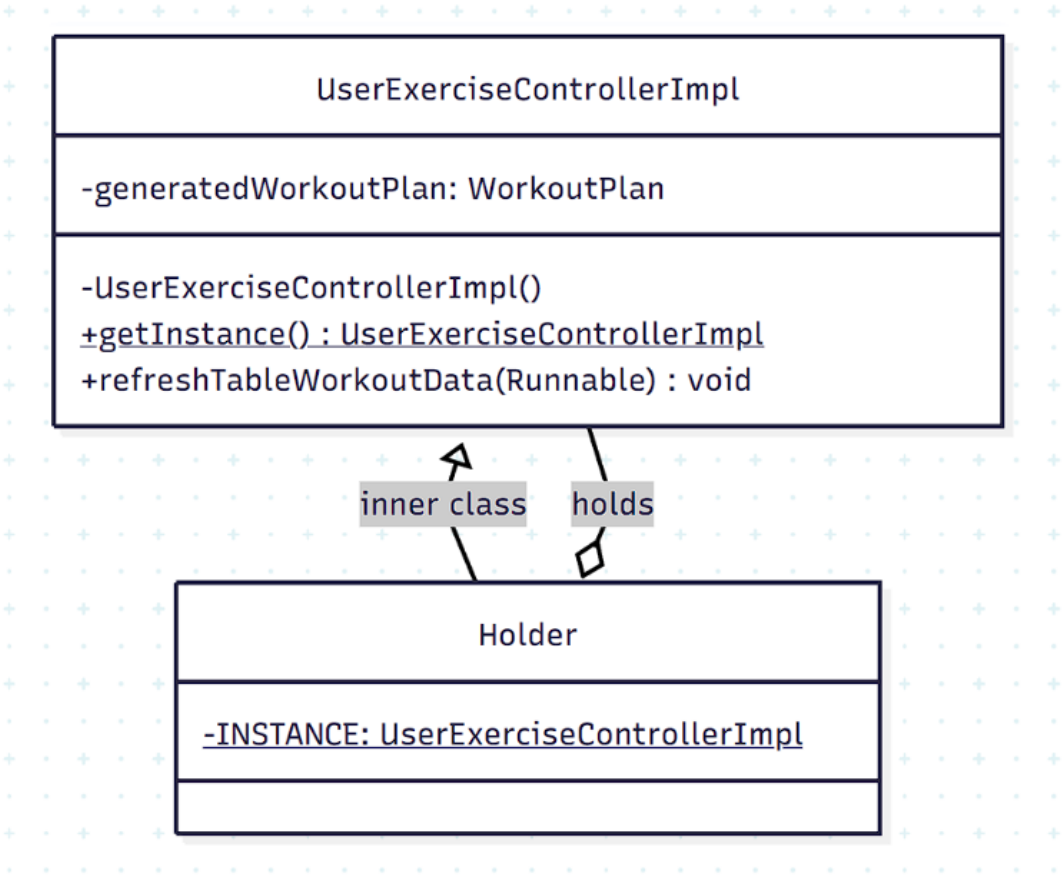

Figura 2.5: Rappresentazione UML del pattern Singleton applicato a UserExerciseControllerImpl.

2.2.6 Gestione delle formule per il BMR (Diego)

Problema: Il calcolo del Basal Metabolic Rate (BMR) può essere effettuato tramite diverse formule scientifiche (es. Mifflin-St Jeor, Harris-Benedict) che hanno parametri e precisione diverse. Scrivere queste formule direttamente dentro la classe `UserProfile` avrebbe reso il codice meno comprensibile e più difficile da gestire un'eventuale nuova formula in futuro.

Soluzione: Per risolvere il problema è stato usato il pattern **Strategy**: l'operazione di calcolo è definita all'interno dell'interfaccia funzionale `BMRCalculatorStrategy`, le due classi `MifflinStJeorStrategy` e `HarrisBenedictStrategy` implementano questa interfaccia e contengono le relative formule per il calcolo.

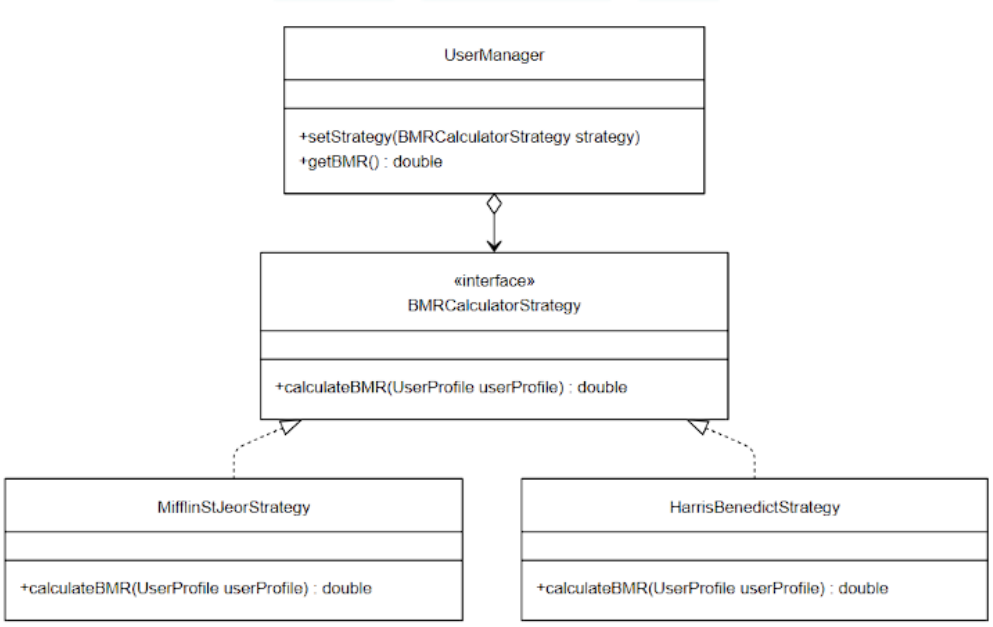

Figura 2.6: Rappresentazione UML del pattern Strategy per il calcolo del BMR.

2.2.7 Incapsulamento di dati e logica nelle enumerazioni (Diego)

Problema: Ogni utente ha i propri parametri specifici, come il livello di attività fisica e l'obiettivo a lungo termine per il peso, che corrispondono a precisi valori necessari per i calcoli del TDEE e del BMR. Affidare la gestione di questi dati a strutture di controllo (come switch o lunghi blocchi if-else nel manager) avrebbe violato il principio di incapsulamento e avrebbe reso il codice più complicato da modificare.

Soluzione: Per risolvere il problema sono state utilizzate delle enumerazioni che contengono al loro interno tutti i dati e la logica necessaria: `ActivityLevel`, dove ogni costante è associata al proprio moltiplicatore numerico accessibile tramite il metodo `getMultiplier()`, e `BMRStrategyChoice`, dove ogni costante crea e contiene direttamente l'istanza della strategia di calcolo associata.

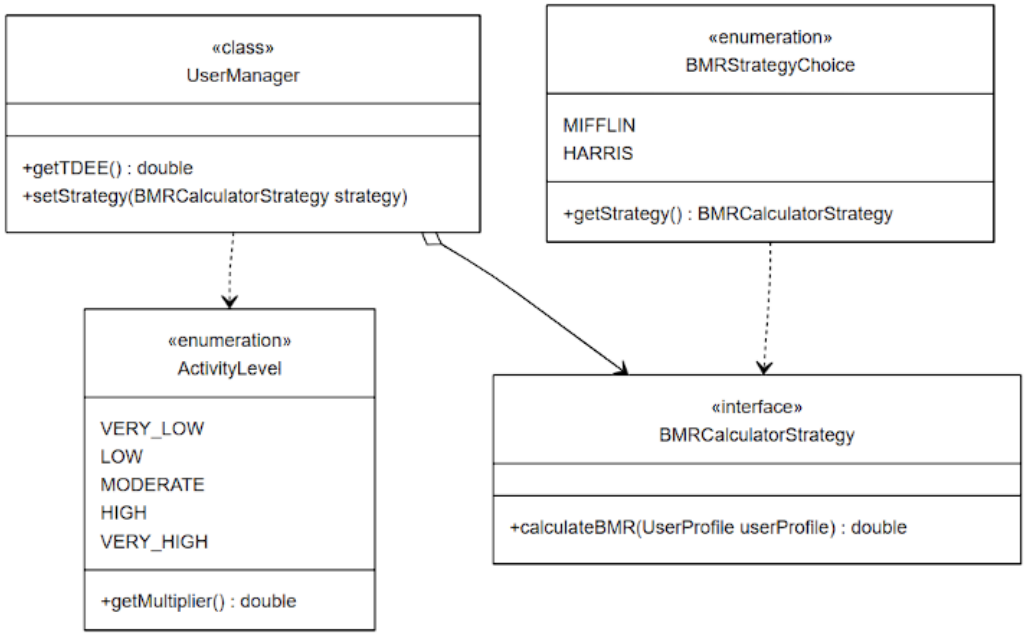

Figura 2.7: Rappresentazione UML del pattern Strategy per il BMR con le enumerazioni Sex e il contesto UserProfile.

2.2.8 Gestione e filtraggio del catalogo per la parte di nutrizione (Leonardo)

Problema: L'applicazione necessita di gestire un catalogo potenzialmente ampio di alimenti, permettendo ricerche rapide, filtraggi basati su criteri nutrizionali e la possibilità di mappare queste informazioni nel log giornaliero dell'utente. Il problema principale è disaccoppiare la logica dalla modalità con cui i dati vengono effettivamente memorizzati e recuperati, evitando che il resto del sistema debba conoscere i dettagli dell'I/O.

Soluzione: È stata scelta un'architettura che separa nettamente la definizione dell'entità (`Food`) dalla sua gestione logica. La soluzione si basa su un'interfaccia `FoodRepository` che funge da strato di mediazione. L'implementazione di un Repository dedicato permette di testare la logica di filtraggio indipendentemente dalla UI e nasconde la complessità del parsing del file CSV dietro metodi puliti.

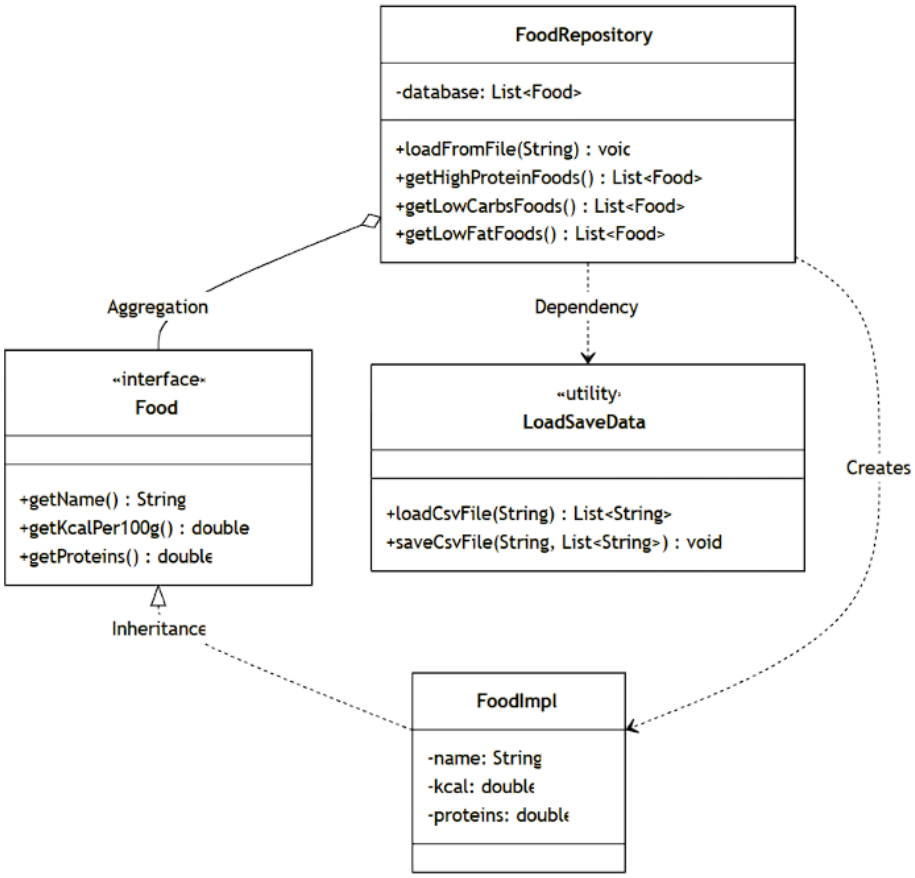

Figura 2.8: Rappresentazione UML della gestione del catalogo alimentare tramite FoodRepository.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Il testing automatizzato del progetto è stato realizzato come da richiesta tramite JUnit. I test verificano il corretto funzionamento delle componenti del Model, in particolare:

- **WikiTest:** controllo che gli articoli e video vengano correttamente caricati e che le funzioni di ricerca base funzionino.
- **SmartSuggestionTest:** controllo delle funzionalità dei suggerimenti intelligenti (`SmartSuggestion`) in base ai dati dell'utente come obiettivo, esercizi da fare e cibo registrato.
- **BMRStrategyTest:** verifica la correttezza delle formule matematiche implementate dalle strategie (Harris-Benedict e Mifflin-St Jeor), confrontando i risultati calcolati con quelli attesi per diverse tipologie di profilo utente.
- **UserManagerTest:** controlla la logica legata all'utente, verificando il calcolo del TDEE (Total Daily Energy Expenditure), la modifica del target calorico giornaliero in base all'obiettivo selezionato e la corretta ripartizione dei macronutrienti.
- **UserProfileTest:** assicura la coerenza dei dati del profilo e il corretto incapsulamento, testando i metodi di accesso e modifica e che vengano lanciate le eccezioni corrette in caso di valori non validi.
- **AbstractPlannedExerciseTest:** test unificato per le classi strength e cardio, verificando le funzionalità in comune e le singole di ogni classe derivata.
- **ExerciseTest:** verifica le funzionalità di base di un esercizio (restituzione del nome, della tipologia, del calcolo delle calorie, dell'obiettivo associato).
- **WorkoutCreatorTest:** testa la logica di creazione in base all'obiettivo e controlla se il safety check sia coerente quando l'utente è in deficit.
- **WorkoutPlanTest:** verifica che il piano di allenamento contenga gli esercizi assegnati e che le schede siano effettivamente presenti.

- **WorkoutSheetTest:** verifica che la gestione di una singola scheda funzioni correttamente (aggiunta e rimozione).
- **DailyLogTest:** verifica la corretta aggregazione dei dati nutrizionali giornalieri.
- **FoodRepositoryTest:** testa la persistenza e il recupero dei dati dal database CSV, verificando il corretto caricamento e il funzionamento dei filtri di ricerca.
- **FoodTest:** verifica l'integrità dell'oggetto Food e la correttezza dei valori nutrizionali.

I test sono completamente automatici e riproducibili senza intervento manuale.

3.2 Note di sviluppo

3.2.1 Salvatore

- **Uso di Stream API e Comparator complessi:** Utilizzato per filtrare e ordinare i suggerimenti intelligenti in `SmartSuggestionImpl`.
  https://github.com/LorenzoBelletti05/Workit-out/blob/7fb1b3ff/src/main/java/it/unibo/workitout/model/wiki/impl/SmartSuggestionImpl.java#L75-L138
- **Uso di lambda expressions e interfacce funzionali (Consumer, Runnable):** Utilizzate in `WikiControllerImpl` e `WikiViewImpl` per registrare listener e gestire callback di navigazione. L'approccio è stato ispirato da un progetto Java trovato online, di cui non è stato possibile recuperare il riferimento preciso.
  Permalink (controller): https://github.com/LorenzoBelletti05/Workit-out/blob/7fb1b3ff/src/main/java/it/unibo/workitout/controller/wiki/impl/WikiControllerImpl.java#L85-L116
  Permalink (view): https://github.com/LorenzoBelletti05/Workit-out/blob/7fb1b3ff/src/main/java/it/unibo/workitout/view/wiki/impl/WikiViewImpl.java#L220-L329
- **Utilizzo della libreria Google Gson (gson.fromJson()):** Usata in `WikiRepositoryImpl` per deserializzare i file JSON della wiki in oggetti Java.
  https://github.com/LorenzoBelletti05/Workit-out/blob/7fb1b3ff/src/main/java/it/unibo/workitout/model/wiki/impl/WikiRepositoryImpl.java#L40

3.2.2 Lorenzo

- **Utilizzo di Reflection e generics:** Gestione del caricamento dei dati e il filtraggio degli esercizi (`LoadFromResources` e `LoadSavedDataFrom`) utilizza la reflection per deserializzare correttamente i JSON. `WorkoutPlanImpl` contiene metodi con generics per operare su sottotipi specifici di `PlannedExercise`.

https://github.com/LorenzoBelletti05/00P25-Workit-out/blob/91d6c3c4/src/main/java/it/unibo/workitout/model/main/datamanipulation/LoadSaveData.java#L78
https://github.com/LorenzoBelletti05/00P25-Workit-out/blob/91d6c3c4/src/main/java/it/unibo/workitout/model/workout/impl/WorkoutPlanImpl.java#L70

- **Utilizzo degli Stream e lambda expression:** Per gestire collezioni di esercizi e calcolo dei dati in `WorkoutSheetImpl` e `WorkoutPlanImpl`.
  https://github.com/LorenzoBelletti05/00P25-Workit-out/blob/91d6c3c4/src/main/java/it/unibo/workitout/model/workout/impl/WorkoutSheetImpl.java#L40
- **Utilizzo degli Optional:** In `WorkoutSheetImpl` e `UserExerciseControllerImpl`.
  https://github.com/LorenzoBelletti05/00P25-Workit-out/blob/91d6c3c4/src/main/java/it/unibo/workitout/model/workout/impl/WorkoutSheetImpl.java#L70
- **Utilizzo della libreria JSON (Gson):** Per la lettura dei file JSON in `LoadSaveData` e `WorkoutCreatorImpl`.
  https://github.com/LorenzoBelletti05/00P25-Workit-out/blob/91d6c3c4/src/main/java/it/unibo/workitout/model/main/datamanipulation/LoadSaveData.java#L1
- **Programmazione funzionale e Method references:** In `WorkoutPlanImpl`.
  https://github.com/LorenzoBelletti05/00P25-Workit-out/blob/91d6c3c4/src/main/java/it/unibo/workitout/model/workout/impl/WorkoutPlanImpl.java#L44

3.2.3 Leonardo

- **Uso di Java Stream API e Predicati:** Utilizzate all'interno di `FoodRepository` per implementare il filtraggio avanzato degli alimenti.
  https://github.com/LorenzoBelletti05/Workit-out/blob/f3ebe6b9/src/main/java/it/unibo/workitout/model/food/impl/FoodRepository.java#L127-L133
- **Utilizzo di Lambda Expressions e Case-Insensitivity:** In `sortByName`, ricerca testuale robusta con normalizzazione tramite `toLowerCase(Locale.ROOT)`.
  https://github.com/LorenzoBelletti05/Workit-out/blob/f3ebe6b9/src/main/java/it/unibo/workitout/model/food/impl/FoodRepository.java#L94-L98
- **Utilizzo di Java Time API (LocalDate):** Il modulo `DailyLog` utilizza `java.time.LocalDate` per la gestione temporale dello storico alimentare.
  https://github.com/LorenzoBelletti05/Workit-out/blob/f3ebe6b9/src/main/java/it/unibo/workitout/model/food/impl/DailyLogImpl.java#L31-L33

3.2.4 Diego

- **Uso di Lambda expression:** Utilizzate all'interno di `UserProfileControllerImpl` per definire gli `ActionListener` della view.
  https://github.com/LorenzoBelletti05/Workit-out/blob/e19b3fb5/src/main/java/it/unibo/workitout/controller/user/impl/UserProfileControllerImpl.java#L46-L52
- **Definizione di interfacce funzionali:** Utilizzo dell'annotazione `@FunctionalInterface` per l'interfaccia `BMRCalculatorStrategy`.
  https://github.com/LorenzoBelletti05/Workit-out/blob/cb8e3bd3/src/main/java/it/unibo/workitout/model/user/model/contracts/BMRCalculatorStrategy.java
- **Utilizzo di Java Time API:** Uso della classe `java.time.LocalDate` nel modello `UserProfile` per la gestione della data di ultimo accesso e la logica di `dailyReset()`.
  https://github.com/LorenzoBelletti05/Workit-out/blob/f6ed594f/src/main/java/it/unibo/workitout/model/user/model/impl/UserProfile.java#L251-L267

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

Diego: Mi sono occupato del modulo User, sviluppando l'intera logica del profilo utente, la relativa interfaccia grafica di login, dashboard e del calcolo dei fabbisogni energetici. Un punto di forza nella mia parte è l'uso del pattern Strategy per il calcolo del metabolismo basale (BMR), questa scelta ha conferito all'utente di decidere con quale formula scientifica preferisce il calcolo. Inoltre la classe `UserManager` ha permesso di intercettare i dati del profilo e di effettuare i vari calcoli in base alle preferenze dell'utente. Tra le debolezze segnalo una rigidità nel trattare l'interfaccia grafica Swing, le view in determinate sezioni presentano dimensioni fisse e layout che ne rendono complessa la manutenzione. In aggiunta, avrei potuto usare i dati giornalieri per creare uno storico dei progressi dell'utente che avrebbe permesso di tenere traccia dei progressi nel tempo.

Lorenzo: Mi sono occupato del modulo Workout, creando la logica di creazione degli esercizi e dell'intera struttura che caratterizza schede e piani di allenamento (compresi gli esercizi stessi) nonché la loro visualizzazione e la coordinazione tra quest'ultima e la logica interna attraverso il controller. I punti di forza sono la suddivisione delle tipologie degli esercizi (STRENGTH e CARDIO) garantendo così una migliore gestione in due macro-categorie e l'uso del Singleton per il controller grazie al quale è possibile far comunicare la `PlanViewerImpl` e il `MainController`. Infine la logica di creazione che grazie ai vari moltiplicatori ricevuti dallo user module e dal calcolo diretto interno, in base a specifiche tipologie di esercizi è stato possibile creare dinamicamente dei piani di allenamento con esercizi ognuno molto diverso dall'altro, evitando ripetizione e garantendo diversità. Per quanto riguarda le debolezze, tendo a creare logiche più complesse di quando possano essere, ripetendo per esempio cicli. Avrei potuto usare più lambda mentre la gestione della logica creazionale avrebbe potuto essere ancora più articolata per una maggior completezza. La `PlanViewerImpl` nella lettura degli esercizi può non essere immediata in quanto alcuni campi sono stati uniti per compattezza, e al completamento dell'esercizio avrei potuto mantenere la lista fissa e nella schermata di modifica non sono stati aggiunti i limitatori dei dati consigliati per la modifica.

Leonardo: Mi sono occupato del modulo Food, gestendo la gerarchia delle entità alimentari e la logica di persistenza dei dati, oltre all'integrazione di questi con il

modulo User. Individuo sicuramente come punto di forza la robustezza della logica di calcolo: il sistema non si limita a un semplice conteggio calorico, ma gestisce l'aggregazione dei macronutrienti e la loro conversione in percentuali caloriche, fornendo dati pronti per essere visualizzati in grafici o riepiloghi. Un altro aspetto positivo è la modularità della logica alimentare. L'aver separato nettamente il calcolo dei macronutrienti dalla gestione del database ha permesso di creare una suite di test che isola la logica dai dati esterni, garantendo che il sistema rimanga estensibile anche in caso di modifiche future alla struttura degli alimenti. Una debolezza riscontrata riguarda la gestione dell'I/O dei file CSV. Inizialmente, la logica di caricamento dipendeva strettamente dalla struttura delle cartelle locali, causando errori durante i primi test di gruppo; ho risolto il problema implementando un sistema di fallback su risorse interne, ma una pianificazione iniziale più accurata sulla gestione dei percorsi avrebbe evitato rallentamenti e permesso una soluzione più pulita. Inoltre, la logica di calcolo dei macronutrienti presuppone una coerenza totale dei dati nel database: se un alimento nel CSV presenta valori nutrizionali incongruenti rispetto alle Kcal totali, il sistema non dispone attualmente di un meccanismo di validazione che segnali l'errore all'utente, limitandosi a eseguire il calcolo matematico.

Salvatore: Mi sono occupato del modulo Wiki (caricamento di articoli e video, ricerca, filtraggio e suggerimenti contestuali) e di alcune parti comuni, tra cui l'impostazione dell'architettura iniziale del progetto. Come punto di forza individuo proprio quest'ultima, che separa chiaramente i vari moduli tramite interfacce dedicate rendendo il sistema scalabile. Fra le debolezze riscontro la poca interconnessione tra il mio modulo e gli altri: ad esempio, il collegamento con il modulo alimentare avviene tramite lettura diretta dello storico invece che attraverso un metodo dedicato. Un'altra carenza è l'impossibilità di riprodurre i video direttamente nella view, dovendo ricorrere a un browser esterno, probabilmente a causa di limitazioni nelle policy di embedding di YouTube nelle webview. Infine, la mia organizzazione del lavoro non è stata costante, concentrandosi prevalentemente nei weekend, il che mi portava a dimenticare le feature già realizzate o a cambiare approccio a metà del progresso, ne è esempio il fatto che il sistema di `SmartSuggestion` è stato creato solo verso la fine del progetto.

Lavori futuri

Fra le possibili evoluzioni del progetto si individuano: l'implementazione di uno storico dei progressi dell'utente, un miglioramento della UI e una migliore standardizzazione del progetto.

4.2 Difficoltà incontrate e commenti per i docenti

Fra le difficoltà riscontrate, la gestione del lavoro di gruppo tramite Git è stata quella che ha richiesto più tempo di adattamento, essendo un argomento poco approfondito durante le lezioni. Anche la realizzazione di diagrammi UML a un livello di astrazione adeguato si è rivelata meno immediata del previsto. Infine, per la stesura della relazione sarebbe stato utile disporre di qualche esempio concreto di relazioni di anni passati, oltre alla meta-relazione, come riferimento pratico.

Appendice A

Guida utente

Al lancio dell'applicazione, si presenta la schermata di inserimento del profilo utente (se è il primo accesso) o direttamente la dashboard (se un profilo è stato già salvato).

- **Profilo utente:** L'utente inserisce nome, cognome, età, altezza (cm), peso (kg), sesso, livello di attività e obiettivo. È possibile scegliere la formula per il calcolo del metabolismo basale tra Mifflin-St Jeor e Harris-Benedict. Premendo il pulsante di conferma, i dati vengono salvati e si accede alla dashboard.
- **Dashboard:** La dashboard mostra un riepilogo del profilo (calorie giornaliere target, macronutrienti target, calorie consumate e bruciate). Da qui si accede ai moduli:
  - **Profilo:** per modificare i dati personali.
  - **Diario Alimentare:** per cercare alimenti, filtrarli (alto contenuto proteico, basso contenuto di grassi o carboidrati) e registrarne il consumo specificando i grammi (cliccare sull'alimento scelto).
  - **Piano di Allenamento:** mostra il piano settimanale generato automaticamente; l'utente può modificare l'intensità degli esercizi, e le calorie bruciate vengono aggiornate nel profilo (cliccare sull'esercizio che si è completato).
  - **Wiki:** per consultare articoli e video, con suggerimenti contestuali basati sul profilo e sulle attività recenti, più viene messa a disposizione la lista di esercizi per esplorare l'elenco completo degli esercizi disponibili.

  Il pulsante "Indietro" presente in ciascun modulo riporta alla dashboard.

Appendice B

Esercitazioni di laboratorio

salvatore.novelli@studio.unibo.it

- Laboratorio 12: https://virtuale.unibo.it/mod/forum/discuss.php?d=211539#p290612
- Laboratorio 11: https://virtuale.unibo.it/mod/forum/discuss.php?d=210617#p289549
- Laboratorio 10: https://virtuale.unibo.it/mod/forum/discuss.php?d=209589#p288284
- Laboratorio 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=208718#p287164
- Laboratorio 08: https://virtuale.unibo.it/mod/forum/discuss.php?d=207921#p286124

leonardo.castiglione@studio.unibo.it

- Non caricati.

lorenzo.belletti5@studio.unibo.it

- Laboratorio 11: https://virtuale.unibo.it/mod/forum/discuss.php?d=210617#p289683
- Laboratorio 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=208718#p287288
- Laboratorio 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=207193#p285053

diego.gessaroli@studio.unibo.it

- Laboratorio 11: https://virtuale.unibo.it/mod/forum/discuss.php?d=210617#p289676
- Laboratorio 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=208718#p287278
- Laboratorio 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=207193#p285052