Università di Catania

DIEEI
Dipartimento di Ingegneria Elettrica Elettronica e Informatica

Relazione Elaborato TrainUp

Lorenzo Basile 1000055691

Antonio Santo Buzzone 1000055698

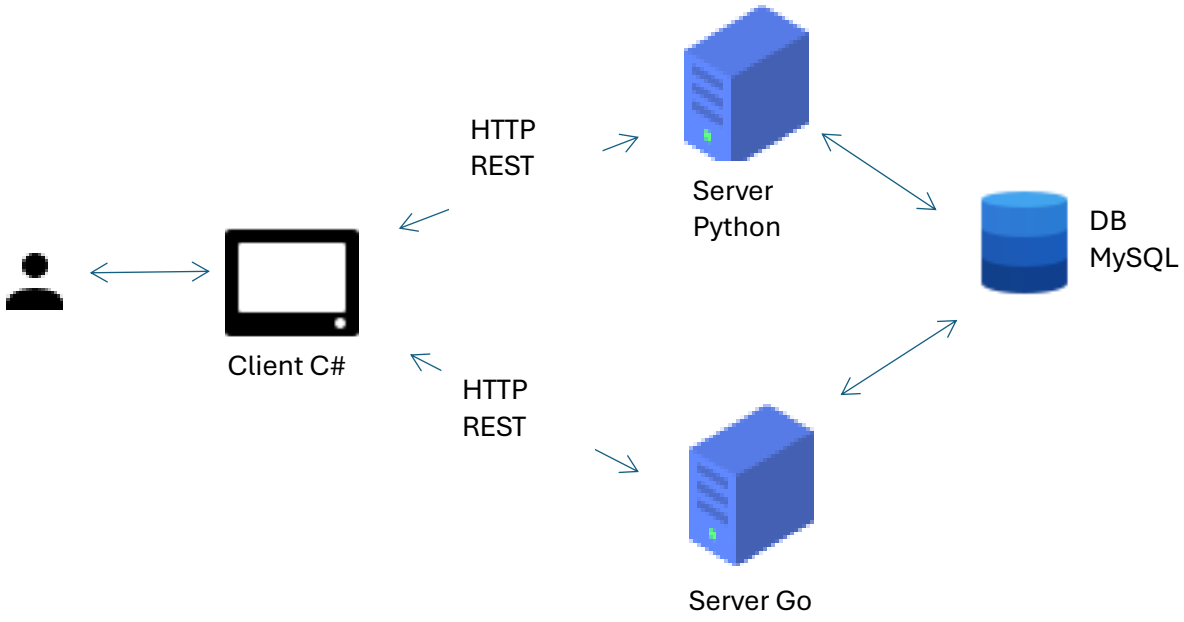
1. Introduzione

Il progetto prevede lo sviluppo di un'applicazione di gestione di programmi d'allenamento. L'obiettivo dell'applicazione di fitness è fornire agli utenti uno strumento completo e personalizzato per la gestione del proprio piano di allenamento.

Di seguito sono elencate le principali funzionalità previste:

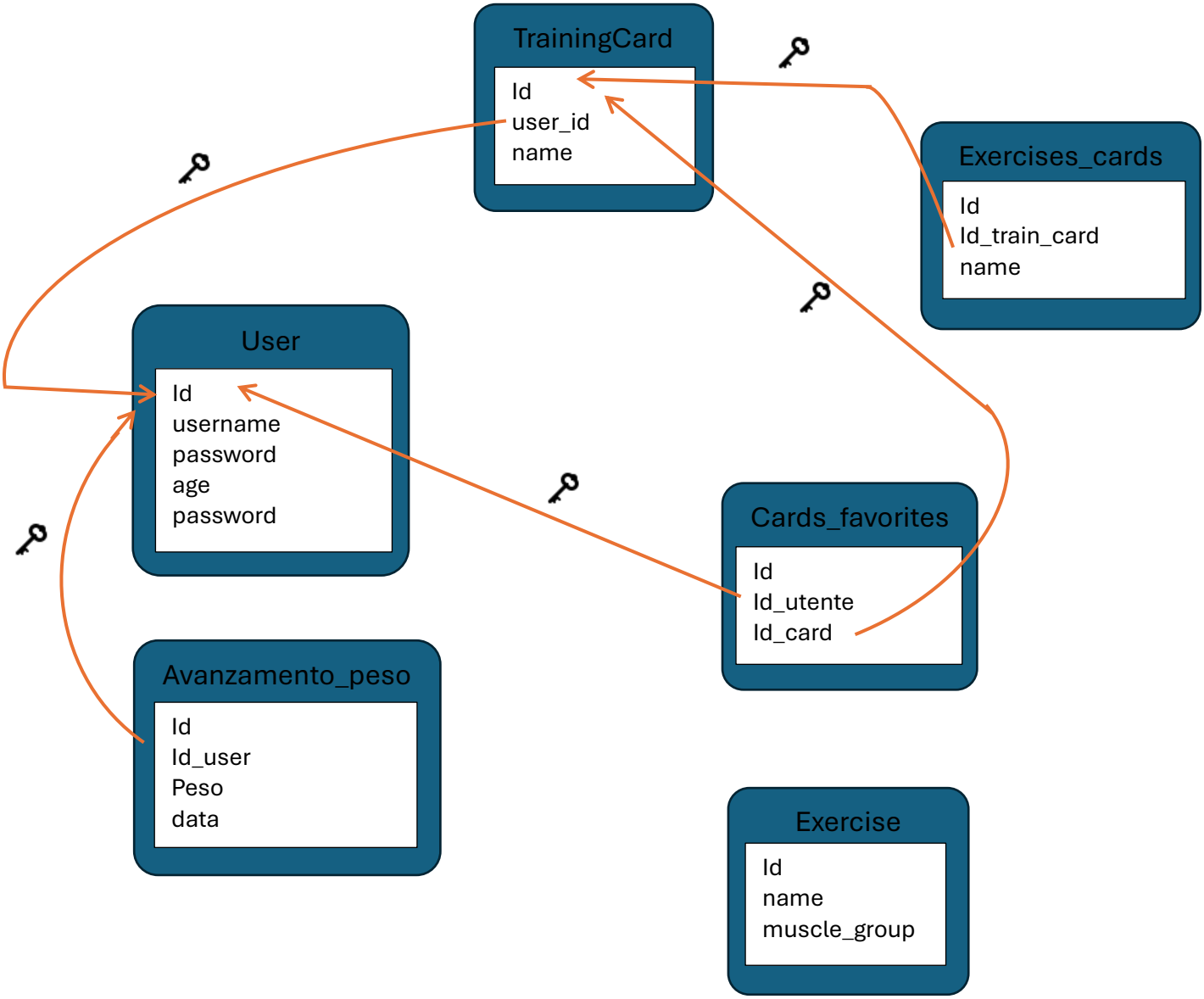
- Registrazione e Profilazione Utente
- Una lista completa di esercizi suddivisi per gruppo muscolare
- Schede di allenamento precompilate che l'utente potrà visualizzare
- Possibilità di creare delle schede di allenamento personalizzabili. L'utente potrà definire i giorni in cui allenarsi e gli esercizi specifici per i giorni scelti.
- Analisi e Grafico: gli utenti potranno inserire dati relativi al proprio stato di salute come peso ecc. e potranno visualizzare grafici che mostrano l'avanzamento del tempo di tali parametri e grafici che mostrano altre analisi.
- Meccanismo dei preferiti: sarà possibile scegliere delle proprie schede o di altri utenti ed inserirli in una lista preferiti in cui possono essere visualizzati separatamente.

2. Composizione applicazione



3. Database

Il database utilizzato è MySQL. Di seguito le tabelle create con le relative relazioni.



Descrizione tabelle

User:

Rappresenta gli utenti del sistema.

Contiene i campi id (chiave primaria), username, password, age e weight (peso attuale).

Exercise:

Memorizza gli esercizi caricati staticamente.

Ogni esercizio ha un id, nome dell'esercizio, e un muscle_group (gruppo muscolare coinvolto).

TrainingCard_:

Rappresenta una scheda di allenamento personalizzata per un utente.

Ha un id, un user_id che fa riferimento all'utente, nome della scheda di allenamento.

ExercisesCards_:

Associa esercizi alle schede di allenamento.

Contiene l'id, il riferimento alla scheda di allenamento tramite id_train_card, il nome dell'esercizio, il numero di sets (serie), di reps (ripetizioni), e il giorno in cui svolgere l'esercizio.

cardsFavorites:

Memorizza le schede di allenamento favorite di un utente.

Contiene l'id_favorite l'id_utente (utente che ha messo la scheda nei preferiti), e l'id_card (ID della scheda di allenamento preferita).

AvanzamentoPeso:

Tiene traccia del progresso di peso di un utente.

Ogni record ha un id, l'id_user (utente a cui appartiene il progresso), il peso (peso registrato) e una data (data di registrazione del peso).

4. Server Python

Utilizzeremo il framework Flask

Gestione di tutte le richieste per la selezione di esercizi e la creazione/personalizzazione di una scheda utente. Gestione del meccanismo di aggiunta e rimozione ai preferiti. Creazione di grafici che rappresentano statistiche, per esempio: scheda più usata, media età utenti che utilizzano l'app e grafici sul miglioramento della salute degli utenti.

Di seguito una breve descrizione dei file.py.

User.py

Questo file mostra l'implementazione delle tabelle tramite ORM (Object-Relational Mapping), specificamente attraverso SQLAlchemy, che consente di mappare le tabelle di un database relazionale a classi Python. In questo modo, è possibile interagire con il database usando il linguaggio Python anziché SQL.

TrainingCard.py

La classe TrainingCard rappresenta un oggetto scheda di allenamento. Ha un ID, un ID utente e una lista di esercizi. Può aggiungere esercizi specificando nome, serie, ripetizioni e giorno, e può restituire i dettagli completi della scheda.

FitnessApp.py

La classe FitnessApp gestisce schede di allenamento personalizzate per diversi utenti. La classe segue il **design pattern Singleton** garantendo che ci sia una sola istanza della classe durante l'intera esecuzione del programma. Il metodo __new__ assicura che venga creata una sola istanza della classe FitnessApp. Se un'istanza esiste già, restituisce quella, mantenendo la stessa lista di schede di allenamento.

App.py

All'interno di questo file sono definite una serie di route per l'applicazione Flask.

Home.py

homeCardDisplayer(): La funzione gestisce la richiesta di visualizzazione delle schede caricate staticamente.

Load_exercise(): La funzione gestisce la richiesta di visualizzazione degli esercizi caricati staticamente.

LoadCardFromDb: La funzione gestisce la richiesta di visualizzazione delle schede caricate dinamicamente, ovvero le schede create dagli utenti.

NewCard.py

addExercise(): La funzione add_exercise aggiunge un esercizio a una lista temporanea dopo aver decodificato un token utente e validato i dati dell'esercizio.

Confirm_creation_card(): La funzione conferma la creazione di una nuova scheda di allenamento per l'utente. Ottiene l'ID utente, controlla se il nome della scheda è valido, e recupera l'ultimo ID scheda dal database. Se non ci sono schede, ne crea una nuova, assegna un nuovo ID. Aggiunge la scheda e gli esercizi associati al database e svuota la lista temporanea degli esercizi dopo averli inseriti.

deleteTrainingCard(): la funzione elimina una scheda di allenamento e tutti gli esercizi associati, previa verifica delle credenziali dell'utente. Se la scheda e l'utente sono validi, gli esercizi e la scheda vengono rimossi dal database e viene restituito uno stato di successo.

Statistiche.py

I metodi servono per generare e restituire grafici relativi ai dati dell'applicazione:

cards_most_used(): Esegue una query per ottenere quante volte ogni scheda di allenamento è stata salvata tra i preferiti dagli utenti e genera un grafico a barre che mostra le schede più popolari.

numberExe(): Esegue una query per contare il numero di esercizi in ciascuna scheda di allenamento e genera un grafico a barre che mostra il numero di esercizi per scheda.

averageAge(): Calcola la media delle età degli utenti registrati e genera un istogramma che mostra la distribuzione delle età, con una linea che indica la media.

Avanzamento(): Il metodo genera un grafico dell'andamento del peso di un utente nel tempo, estraendo i dati dal database e restituendolo come immagine in formato PNG.

Favorites.py

addFavoritesCard(): La funzione add_favorite_card aggiunge una carta ai preferiti di un utente. Decodifica il token JWT per ottenere l'ID dell'utente, recupera l'ID della carta, e salva questa associazione nel database, restituendo un JSON di conferma.

remove_favorite_card(): La funzione remove_favorite_card rimuove una carta dai preferiti di un utente. Decodifica il token JWT per ottenere l'ID dell'utente, trova e elimina l'associazione con la carta nel database, e restituisce una conferma in JSON.

5. Server Go

Utilizzeremo il framework Echo
Gestione delle richieste di registrazione, valida i dati e memorizza le informazioni dell'utente nel database. Gestione delle richieste di aggiornamento dati dell'utente (password, peso, età).

Di seguito una breve descrizione dei file.go.

models.go

Il file definisce un tipo User con campi per ID, nome utente, password, età e peso. La funzione init crea o aggiorna la tabelle nel database, a condizione che il database sia inizializzato. Se il database non è inizializzato, viene registrato un messaggio di errore.

app.go

Il pacchetto configura un'applicazione web usando il framework Echo e GORM per gestire un database MySQL. All'avvio, stabilisce una connessione al database. Definisce le rotte per l'accesso utente, registrazione, disconnessione, e aggiornamento dei dati e della password.

routes.go

Questo file definisce diverse funzioni di gestione delle richieste HTTP per un'applicazione web utilizzando Echo. Ecco una breve descrizione di ciascuna:

userLogin: Gestisce le richieste di login. Riceve i dati di login, li verifica e restituisce un token se il login è riuscito, altrimenti un errore di autorizzazione.

userRegister: Gestisce le richieste di registrazione. Riceve i dati di registrazione, li elabora e restituisce un token se la registrazione ha successo, altrimenti un errore interno del server.

userLogout: Gestisce le richieste di logout. Riceve i dati di logout e invalida il token se il logout ha successo.

updateData: Gestisce le richieste di aggiornamento dei dati utente. Riceve i dati da aggiornare e applica le modifiche. Restituisce lo stato dell'operazione.

updatePassword: Gestisce le richieste di cambio password. Riceve i dati necessari e applica il cambiamento. Restituisce lo stato dell'operazione.

auth.go

login(): verifica le credenziali dell'utente e, se valide, genera un token JWT. Se l'utente è già stato invalidato, lo rimuove dalla lista dei token invalidati.

register(): registra un nuovo utente, verifica che tutti i campi siano corretti e genera un token JWT per il nuovo utente

logout(): invalida il token JWT fornito, aggiungendolo alla lista dei token invalidati, per disconnettere l'utente.

updateUserData(): aggiorna i dati dell'utente (età o peso) basandosi sul token JWT fornito e sul campo di aggiornamento specificato.

changeUserPassword(): cambia la password dell'utente se la vecchia password è corretta e la nuova password è confermata. Restituisce uno stato che indica se l'operazione è riuscita.

6. Client C#

Fornisce un'interfaccia per l'utente per tutte le possibili funzioni derivanti da entrambi i server.

Page0.xaml.cs: questa pagina WPF gestisce una logica di login. Quando l'utente inserisce username e password e clicca su "Submit", viene eseguita una richiesta HTTP POST a un server locale, inviando le credenziali in formato JSON. Se il server risponde con successo (stato "success"), l'utente viene reindirizzato a una nuova pagina (Page1), passando il token di autenticazione. Se invece lo stato è "fault", viene mostrato un messaggio di errore.

Page1.xaml.cs: questo file gestisce la pagina di home in cui si arriva dopo il login, visualizzando schede di allenamento sia predefinite che create dagli utenti.

Page2.xaml.cs: questo file gestisce la pagina di un'applicazione WPF, che visualizza una scheda di allenamento selezionata dalla Home (ovvero da page1.xaml.cs). Raggruppa e mostra gli esercizi per giorno. L'utente può tornare alla Home tramite un pulsante, mantenendo il token di autenticazione per la navigazione.

Page3.xaml.cs: questa pagina consente agli utenti di registrarsi inviando dati al server e gestisce le risposte di successo o errore, navigando verso la pagina di login o verso la Home a seconda del risultato.

Page4.xaml.cs: questa pagina gestisce l'aggiornamento dei dati utente, la modifica della password e la cancellazione delle schede di allenamento. Permette di aggiornare età e peso, modificare la password, e rimuovere schede, comunicando con il server.

Page5.xaml.cs: la classe "Page5" mostra un menù verso diverse pagine che mostrano esercizi classificati per categorie specifiche (come spalle, torace, addome, ecc.).

creaScheda.xaml.cs: la classe creaScheda gestisce la creazione di una scheda di allenamento. All'inizio, carica gli esercizi disponibili da un database e li visualizza in una ListBox. Gli utenti possono selezionare esercizi, specificare il numero di set e ripetizioni, e aggiungere questi esercizi a una scheda. Inoltre, è possibile confermare la creazione della scheda e navigare indietro alla pagina principale.

statistiche.xaml.cs: la classe Statistiche gestisce la visualizzazione delle statistiche in un'applicazione WPF. Permette di navigare alla pagina principale e di aprire una finestra per visualizzare immagini statistiche tramite diversi pulsanti.

Una serie di file dove il nome inizia con **stampaEs**, come ad esempio **StampEs_addome.xaml.cs**, **StampaEs_spalle.xaml.cs** ecc... sono state utilizzate per la visualizzazione degli esercizi dal menu di Page5, sia in un'unica lista che suddivisi per gruppo muscolare.