

UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTHENOPE

DIPARTIMENTO DI INGEGNERIA



**CORSO DI LAUREA IN INGEGNERIA E SCIENZE INFORMATICHE PER LA
CYBERSECURITY**

Tema selezionato

**Progettazione e implementazione di un sistema informativo per la gestione della
sezione Schede di allenamento di una palestra**

DOCENTE

PROF. Daniele Granata
Lucia,0334000092

STUDENTE

Vincenzo De

Traccia d'Esame – Progetto di Sistema Informativo per la gestione della sezione Schede di allenamento di una palestra

Il progetto si propone di sviluppare un **sistema informativo avanzato** con l'obiettivo di **digitalizzare e ottimizzare la gestione delle schede d'allenamento** all'interno di contesti sportivi quali palestre commerciali, private o studi di personal training. L'iniziativa nasce dall'esigenza di superare le inefficienze legate all'utilizzo di metodi tradizionali basati su supporti cartacei, spesso soggetti a deterioramento, smarrimento o difficoltà di aggiornamento, oltre che caratterizzati da un'elevata lentezza nelle comunicazioni tra trainer e cliente.

Il sistema sarà realizzato sotto forma di **applicazione web full-stack**, accessibile da qualsiasi dispositivo dotato di browser, con un'interfaccia utente progettata per garantire **usabilità, responsività e accesso immediato alle informazioni**. Gli utenti principali del sistema saranno i Personal Trainer e i clienti iscritti.

Il cuore logico e strutturale del sistema sarà costituito da un **database relazionale basato su SQLite**, scelto per la sua leggerezza, semplicità d'implementazione e adeguatezza in contesti embedded o a basso carico. Il database sarà progettato seguendo i principi della **progettazione concettuale, logica e fisica dei dati**, con l'obiettivo di garantire **integrità referenziale** e prestazioni ottimizzate nelle operazioni CRUD.

Le principali entità coinvolte nel sistema saranno:

- **Scheda d'allenamento:** documento digitale strutturato in sessioni, esercizi e parametri associati (serie, ripetizioni, carichi, tempi di recupero, ecc.), collegato sia al cliente sia al Personal Trainer che l'ha redatta.
- **Esercizio:** catalogo dettagliato degli esercizi disponibili, comprensivo di descrizioni tecniche, muscoli coinvolti, livello di difficoltà, materiale richiesto ed eventuali note esecutive.
- **Cliente:** anagrafica degli iscritti, con informazioni personali, obiettivi di allenamento, stato fisico attuale, patologie eventualmente dichiarate, e cronologia delle schede ricevute.
- **Esigenze e supplementazione:** raccolta dati sugli eventuali integratori assunti regolarmente (proteine, creatina, vitamine, ecc.) e note utili ai fini della personalizzazione delle schede.
- **Personal Trainer:** profilo professionale del tecnico abilitato alla creazione e alla gestione delle schede, con possibilità di monitorare l'andamento dei propri clienti e capirne l'andazzo in modo più diretto e oggettivo.

L'applicativo fornirà funzionalità avanzate quali:

- **Generazione assistita delle schede** tramite interfacce guidate
- **Ricerca e filtraggio esercizi** per categoria o gruppo muscolare
- **Possibilità di visualizzare** anche le schede passate
- **Gestione degli integratori** presi dall'atleta
- **Gestione autenticazione e autorizzazioni**, con ruoli distinti per Trainer e Clienti

L'intero progetto sarà sviluppato con un'architettura modulare e scalabile, per garantire manutenibilità e separazione delle responsabilità. Sono previsti futuri sviluppi che potrebbero

includere il supporto a notifiche push, integrazione con dispositivi wearable (es. fitness tracker), e dashboard di monitoraggio dei progressi mediante grafici dinamici.

Il sistema informativo proposto implementa una **gestione a ruoli** che distingue due categorie principali di utenti: **Clienti** e **Personal Trainer**. Questa suddivisione logica consente una **gestione differenziata dei permessi e delle funzionalità disponibili**, garantita da un sistema di **autenticazione e autorizzazione sicuro**, basato su credenziali univoche e controllo dei privilegi a livello di sessione utente.

1. Clienti (Utenti standard)

Gli utenti classificati come **Clienti** rappresentano il nucleo dell'utenza finale. A ciascun cliente è assegnato un profilo personale dal quale può accedere a una serie di funzionalità pensate per accompagnare e monitorare il proprio percorso di allenamento. Tra queste si annoverano:

- **Visualizzazione delle schede di allenamento:** i clienti possono consultare in tempo reale le schede personalizzate redatte dal proprio Personal Trainer, organizzate per data e ciclo di allenamento, con dettagli su esercizi, set, ripetizioni e carichi.
- **Gestione supplementazione:** ogni cliente può creare una propria **"scheda integratori"** personalizzata, elencando gli integratori alimentari che assume nel corso del proprio piano di allenamento, con indicazioni su posologia, marca, obiettivo e durata del ciclo. Questa funzione ha lo scopo di favorire una **visione integrata tra allenamento e supporto nutrizionale**.

2. Personal Trainer (Utenti avanzati)

Gli utenti con ruolo di **Personal Trainer** dispongono di un accesso privilegiato al sistema, che consente loro di **creare e gestire le schede di allenamento dei clienti affidati**. Le principali funzionalità a loro disposizione includono:

- **Creazione delle schede:** attraverso un'interfaccia dedicata, i Trainer possono generare nuove schede d'allenamento personalizzate, selezionando esercizi dal database, configurando parametri e allegando eventuali note. Ogni scheda viene poi assegnata a uno o più clienti.
- **Upload delle certificazioni professionali:** per incrementare la propria credibilità e professionalità, i Trainer possono caricare documenti attestanti la loro qualifica (es. lauree, corsi, attestati), che saranno visibili nel loro profilo. Ciò consente una **valorizzazione del merito** all'interno della piattaforma.

La gestione dei ruoli è implementata attraverso un sistema di **autenticazione utente** (login) con verifica delle credenziali e **autorizzazione basata su ruoli**, assicurando che ogni utente possa accedere esclusivamente alle funzioni previste per la propria tipologia. Tale impostazione favorisce l'organizzazione e la sicurezza del sistema, evitando sovrapposizioni funzionali o violazioni dei dati.

Oltre alle funzionalità di base, il sistema informativo prevede una serie di **funzionalità avanzate lato utente (cliente)**, volte a migliorare l'esperienza d'uso e a offrire un **maggiore controllo e**

consapevolezza sul proprio percorso di allenamento. Queste funzionalità, perfettamente integrate nell'interfaccia utente e supportate da opportune interrogazioni sul database relazionale, contribuiscono a rendere il sistema un vero e proprio strumento digitale di supporto continuo.

- **Accesso allo storico delle schede**
Ogni cliente potrà accedere a una schermata riepilogativa in cui saranno visualizzate **tutte le schede d'allenamento precedentemente ricevute**.
- **Preferiti e personalizzazione dell'esperienza**
Il sistema prevede una funzionalità che consente all'utente di **salvare determinati esercizi come preferiti**, per accedervi rapidamente in futuro o richiederne l'inserimento nelle schede successive. Gli esercizi preferiti saranno visualizzabili in un'apposita sezione personale.
- **Classificazione e filtro degli esercizi per gruppo muscolare**
Ogni esercizio inserito nel database è **etichettato in base al gruppo muscolare principale coinvolto** (es. pettorali, dorsali, gambe, addome, spalle, ecc.). L'interfaccia permetterà di navigare nel catalogo esercizi attraverso un sistema di filtri dinamici, così da agevolare la ricerca mirata di contenuti, sia per i clienti (per studio o curiosità personale), sia per i Personal Trainer (in fase di creazione scheda).
- **Schedulazione dell'allenamento e generazione dinamica delle liste esercizi**
Uno degli elementi chiave del sistema è la possibilità di associare a ciascun cliente una **schedulazione temporale dell'allenamento** (es. lunedì – push, mercoledì – gambe, venerdì – pull). In base a questa struttura, il sistema sarà in grado di eseguire **query mirate sul database** per restituire, giorno per giorno, l'elenco completo degli esercizi da svolgere.

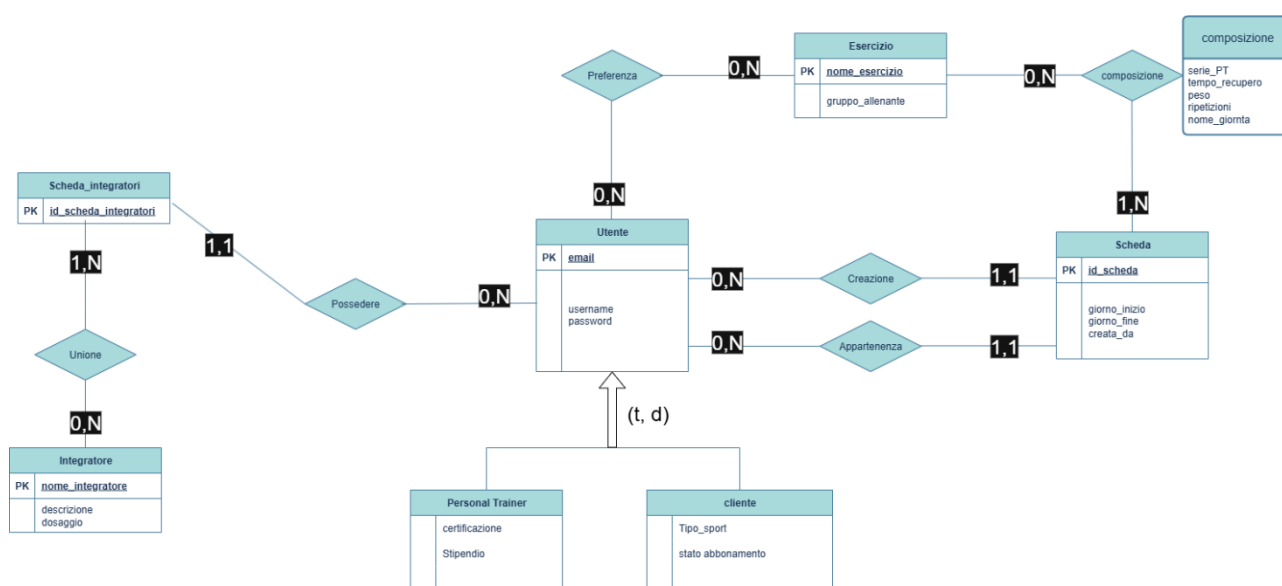
Obbiettivi del Progetto

Il progetto si inserisce in un contesto di **ampia rilevanza socio-economica**. In Italia, infatti, si stima che **oltre un terzo della popolazione pratici regolarmente attività fisica all'interno di palestre o centri sportivi**, il che rende il settore uno dei più dinamici e promettenti in termini di digitalizzazione dei servizi e ottimizzazione gestionale.

Alla luce di questi dati, il sistema proposto ha l'obiettivo di rivolgersi non solo a piccole strutture o personal trainer indipendenti, ma anche a **palestre multifunzionali di medie e grandi dimensioni**, sia di natura **pubblica** (es. centri sportivi comunali, impianti universitari) che **privata** (palestre commerciali, boutique gym, studi personalizzati).

L'obiettivo primario del sistema è quello di **migliorare l'esperienza dell'utente finale**, ma anche di **alleggerire il carico organizzativo** dello staff tecnico e amministrativo delle strutture ospitanti.

1. Analisi e progettazione concettuale: Modello Entità/Relazione



Prototipo del modello E/R con generalizzazione del tipo **totale e disgiunta** per l'entità "Utente" al fine di organizzare meglio i ruoli degli utenti e separarne le funzionalità.

La generalizzazione di *Utente* definisce due sottotipi: Personal Trainer e Cliente.

- **Totale**: ogni istanza dell'entità Utente deve appartenere obbligatoriamente ad una delle due specializzazioni figlie.
- **Disgiunta**: un utente può essere o un Personal Trainer o un Cliente, ma non entrambi contemporaneamente.

Possibili soluzioni della generalizzazione

- 1) Scenario 1 – **"Tutto nel padre"**: la generalizzazione ingloba completamente le specializzazioni fra ruoli assorbendo gli attributi delle entità figlie "Personal Trainer" e "Cliente" in un'unica tabella "Utente" che comprende tutti gli attributi. Gli attributi specifici di ciascun sottotipo vengono lasciati **NULL** nell'altro (perché sono attributi che non gli appartengono). È utile usare questo tipo di strategia quando le sottoclassi hanno molti

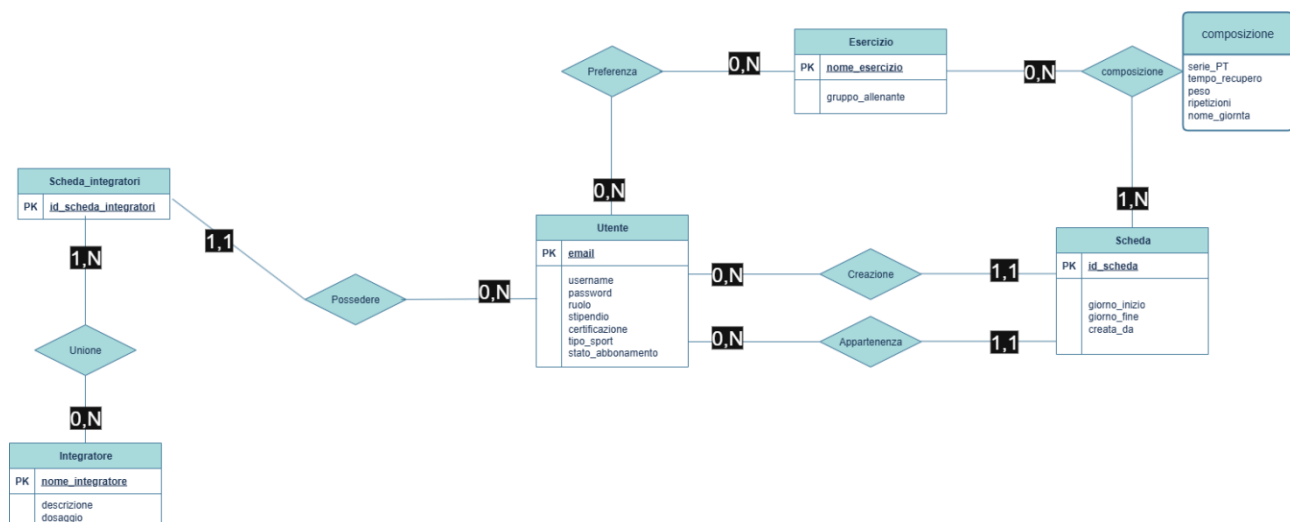
attributi in comune (evita ridondanze), il numero di sottoclassi è limitato e stabile e i dati sono usati insieme frequentemente. Garantisce una struttura più semplice e riduce il numero di tabelle. Tuttavia, usando questa strategia, ci saranno attributi che saranno NULL.

- Vantaggi: Garantisce una struttura più **semplice**, **riduce** il numero di tabelle, **performance** migliori in lettura, gestione centralizzata degli utenti.
- Svantaggi: presenza di molti valori **NULL**, query più complesse.

2) Scenario 2 – “**Tutto nelle figlie**”: non esiste una tabella superclasse, ma ogni sottoclasse viene mappata in una tabella diversa. Le specializzazioni ereditano gli attributi della generalizzazione, che vengono replicati in ciascuna figlia. Utile quando le sottoclassi hanno vincoli forti e diversi tra di loro.

- Vantaggi: tabelle **compatte** e **specifiche** e facile applicazione di vincoli
- Svantaggi: più tabelle da gestire, **ridondanze** negli attributi comuni, aumento della complessità applicativa, **scarsa scalabilità** in termini di estensione del modello.

Modello E/R ottimizzato



Passando al modello relazionale usato per il progetto, l’approccio scelto prevede l’unificazione delle entità figlia in un’unica super-entità. Questo comporta:

- Una sola tabella, “Utente”, unica per tutti gli utenti, con attributi opzionali specifici che possono essere nulli e con l’aggiunta di un attributo **ruolo** per distinguerli.

È stata presa questa scelta per:

- **garantire una maggiore scalabilità** in futuro, cioè se si vorranno aggiungere altri ruoli come proprietario della palestra o altri tipi di istruttori sarà più facile farlo.
- consente una **gestione migliore** del database.
- garantisce **query più semplici**: meno join, infatti non ci saranno join con le tabelle *cliente* i *personal trainer* ma solo con *Utente*.
- **Evitare** di avere **attributi duplicati**

Gli attributi delle entità figlie “Personal Trainer” e “Cliente” vengono inglobati all’interno della classe padre “Utente”. Gli attributi che non sono pertinenti a *Personal Trainer* assumeranno il valore NULL e viceversa.

- Utente: email, nome, password
 - Personal trainer: certificazione, stipendio
 - Cliente: tipo_sport, stato_abbonamento

Nel modello E/R questi attributi sono visualizzati come facenti parte tutti della stessa tabella, ma vincolati dalla specializzazione.

Discussione Cardinalità

- **Utente(0:N) → Creazione ← Scheda (1:1):**
 - Un Utente (ruolo = Personal Trainer) può creare da 0 a N schede d’allenamento
 - Una Scheda d’allenamento può essere creata da uno e un solo Personal Trainer
- **Utente(0:N) → Appartenenza ← Scheda (1:1):**
 - Un Utente (ruolo = Cliente) può avere 0 (nel caso si sia appena iscritto) o più schede d’allenamento
 - Una scheda d’allenamento può essere posseduta da uno e un solo utente
- **Utente(0:N) → Preferenza ← Esercizio (0:N):**
 - Un Utente può avere minimo 0 e massimo N esercizi nei preferiti
 - Un Esercizio può essere messo nei preferiti da minimo 0 utenti e massimo N
- **Scheda(1:N) → Composizione ← Esercizio (0:N):**
 - Una scheda per essere valida deve avere minimo 1 esercizio e massimo N all’interno di essa
 - Un esercizio deve essere in minimo 0 massimo N schede
- **Utente(0:N) → Possedere ← Scheda_integratori (1:1):**
 - Un utente può possedere 0 (se la sua dieta non ne necessita) o più schede di integratori
 - Una scheda di integratori può essere posseduta da uno e un solo utente
- **Scheda_integratori(1:N) → Unione ← Integratore (0:N):**
 - In una scheda di integrazione ci possono essere minimo 1 integratore, per essere valida, massimo N
 - Un integratore può far parte di 0 o di N schede di integrazione

2. Progettazione Logica

La progettazione logica del database è stata realizzata sulla base del modello E/R precedentemente descritto.

Utente: (e-mail, username (UNIQUE), password, ruolo(PT, cliente), stipendio(NULLABLE, PT), certificazione(NULLABLE, PT), tipo_sport(NULLABLE, cliente), stato_abbonamento(NULLABLE, cliente))

- La tabella Utente contiene le informazioni di profilazione di ciascun utente della piattaforma, PT o cliente semplice. Si usa come chiave primaria l'email perché è unica e non ne esistono due uguali; mentre l'attributo *username* è impostato come univoco per garantire evitare ambiguità nel sistema. Gli attributi "**stipendio**" e "**certificazione**" sono considerabili **NULLABLE** per i **Clienti**. Gli attributi "**tipo_sport**" e "**stato_abbonamento**", invece, sono considerabili **NULLABLE** per i **Personal Trainer**.

Esercizio: (nome, serie_PT, gruppo_allenante)

- L'entità Esercizio contiene le informazioni principali dei diversi esercizi che il cliente può avere in scheda con **nome** come chiave primaria perché si assume che non esistono due esercizi diversi ma con lo stesso nome.

Preferenza: (nome_esercizio: Esercizio, e-mail: Utente)

- Relazione N:N fra *Utente* ed *Esercizio*, utile per tenere traccia degli esercizi che un utente mette nei preferiti.

Scheda: (id_scheda, giorno_inizio, giorno_fine, creata_da, email: Utente)

- Scheda è la traduzione in tabella della classica scheda da palestra. Il campo *creata_da* funge come da "firma" per il PT. Si assume che una scheda intera sia divisa in più giorni, giornata dedicata al petto, una dedicata alle braccia e così via. Quindi ogni programmazione (ogni scheda intera) ha il suo *id_scheda* univoco e al suo interno è formata da tante giornate quante ha deciso il PT. Inoltre, poiché Ci sono due relazioni **1,N** fra *Utente* e Scheda è stata inserita la chiave primaria di *Utente* in Scheda.

Composizione: (nome_esercizio: Esercizio, id_scheda: Scheda, serie_PT, tempo_recupero, peso, ripetizioni, nome_giornata)

- Relazione N:N fra *Esercizio* e *Scheda* molto utile quando si devono fare query nel database per far visualizzare all'utente quali esercizi gli spettano in una certa giornata di allenamento e per la creazione effettiva delle schede personalizzare per ciascun atleta.

Scheda_integratori: (id_scheda_integratori, email: Utente)

- In questa tabella saranno presenti tutti gli integratori che un Cliente prende durante i suoi cicli di allenamento. È legata alla tabella *Utente* tramite una relazione **0,N**, per questo all'interno della tabella è presente anche la chiave primaria di *Utente*.

Integratore: (nome_integratore, descrizione, dosaggio)

- La tabella integratore serve per tenere traccia nel database dei possibili supplementi che un atleta potrebbe prendere. Si assume che non esistano due integratori diversi ma con lo stesso nome. Grazie agli attributi dosaggio e descrizione si riesce ad avere una visione completa dell'integratore.

Unione: (id_scheda_integratori: Scheda_integratori, nome_integratore: Integratore)

- Relazione N:N fra *Integratore* e *Scheda_integratore*, utile quando si devono fare query per far visualizzare all'utente la sua intera scheda di supplementi.

*Entità e Relazioni***Utente**

Nome Campo	Descrizione	Tipo Dati	Lunghezza	Vincoli
Email	L'email con cui accede l'utente	VARCHAR	200	PK
Username	Username che sceglie l'utente	VARCHAR	100	UNIQUE, NOT NULL
Password	Password di accesso	VARCHAR	200	NOT NULL
Ruolo	Ruolo utente (Personal Trainer, Cliente)	ENUM		NOT NULL, scelta='Personal Trainer' o 'Cliente'
Stipendio	Stipendio del PT	DECIMAL(8,2)		NULLABLE, (riempire solo se ruolo='Personal Trainer'), DEFAULT=0.00
Certificazione	Certificazioni del PT	VARBINARY(MAX)		NULLABLE, (riempire solo se ruolo='Personal Trainer')
Tipo_sport	Tipo di sport che esegue il cliente	VARCHAR	100	NULLABLE(riempire solo se ruolo='Cliente')
Stato_abbonamento	Stato dell'abbonamento del cliente alla palestra	BOOLEAN		NULLABLE(riempire solo se ruolo='Cliente')

- **Si assume che gli utenti con email contenente il dominio "@pt.com" siano Personal Trainer certificati:** per semplicità di progettazione e di validazione a livello applicativo, si stabilisce che qualunque utente registrato con un indirizzo email che termina con @pt.com sia considerato un Personal Trainer abilitato. Questo è un **vincolo semantico di dominio** e sarà gestito lato applicativo.
- **Si assume che l'attributo certificazioni della tabella Utente,** accessibile solo per utenti con ruolo "Personal Trainer", **possa contenere un solo file cumulativo che racchiude tutte le certificazioni possedute dal trainer.**
Questo è un vincolo di tipo condizionale e sarà gestito lato sistema.

Esercizio

Nome Campo	Descrizione	Tipo Dati	Lunghezza	Vincoli
Nome	Nome dell'esercizio	VARCHAR	100	PK

Gruppo_allenante	Muscolo allenato	ENUM		CHECK
------------------	------------------	------	--	-------

Preferenza

Nome Campo	Descrizione	Tipo Dati	Lunghezza	Vincoli
Nome_Esercizio	FK Esercizio	VARCHAR	100	PK, FK -> Esercizio
email	FK Utente	VARCHAR	200	PK, FK -> Utente

Scheda

Nome Campo	Descrizione	Tipo Dati	Lunghezza	Vincoli
Id_scheda	Identificatore scheda	INT		PK, AUTO_INCREMENT
Giorno_inizio	Giorno di inizio scheda	DATE		NOT NULL
Giorno_fine	Giorno di fine scheda	DATE		NOT NULL
Creata_da	Da chi è stat fatta la scheda	VARCHAR	100	NOT NULL, valori ammessi = solo Utenti "Personal Trainer"
Email	FK Utente	VARCHAR	200	NOT NULL, FK -> Utente

- La data di fine validità di una scheda non può mai essere precedente alla data di inizio. Nella base di dati, questo vincolo viene implementato tramite un CHECK nella tabella Scheda, in modo da garantire la coerenza temporale di ogni piano di allenamento inserito.

Composizione

Nome Campo	Descrizione	Tipo Dati	Lunghezza	Vincoli
Nome_esercizio	FK Esercizio	VARCAHR	100	PK, FK -> Esercizio

Id_scheda	FK Scheda	INT		PK, FK -> Scheda
Nome_giornata	FK Scheda	VARCAHR	100	PK, FK -> Scheda
Serie_PT	Serie da fare, che inserisce il PT	INT		
Tempo_recupero	Tempo di recupero fra una serie e l'altra	INT		DEFAULT=60
Peso	Peso con cui si è fatta la serie	DECIMAL(5,2)		DEFAULT=0.00
Ripetizioni	Ripetizioni che compongono la serie	INT		
Nome_giornata	nome della giornata d'allenamento	VARCHAR	100	

Scheda_integratori

Nome Campo	Descrizione	Tipo Dati	Lunghezza	Vincoli
Id_scheda_integratori	Identificativo della scheda integratori	INT		PK, AUTO_INCREMENT
email	FK Utente	VARCHAR	200	FK -> Utente

Integratore

Nome Campo	Descrizione	Tipo Dati	Lunghezza	Vincoli
Nome_integratore	Nome dell'integratore	VARCHAR	100	PK
Descrizione	Descrizione dell'integratore	TEXT		
dosaggio	Dosaggio da prendere	DECIMAL(6,2)		NOT NULL, dosaggio in grammi

Unione

Nome Campo	Descrizione	Tipo Dati	Lunghezza	Vincoli
Nome_integratore	FK Integratore	VARCHAR	100	PK, FK -> Integratore

Id_scheda_integratori	FK Scheda_integratori	INT		PK, FK -> Scheda_integratori
-----------------------	--------------------------	-----	--	---------------------------------

Vincoli interrelazionali:

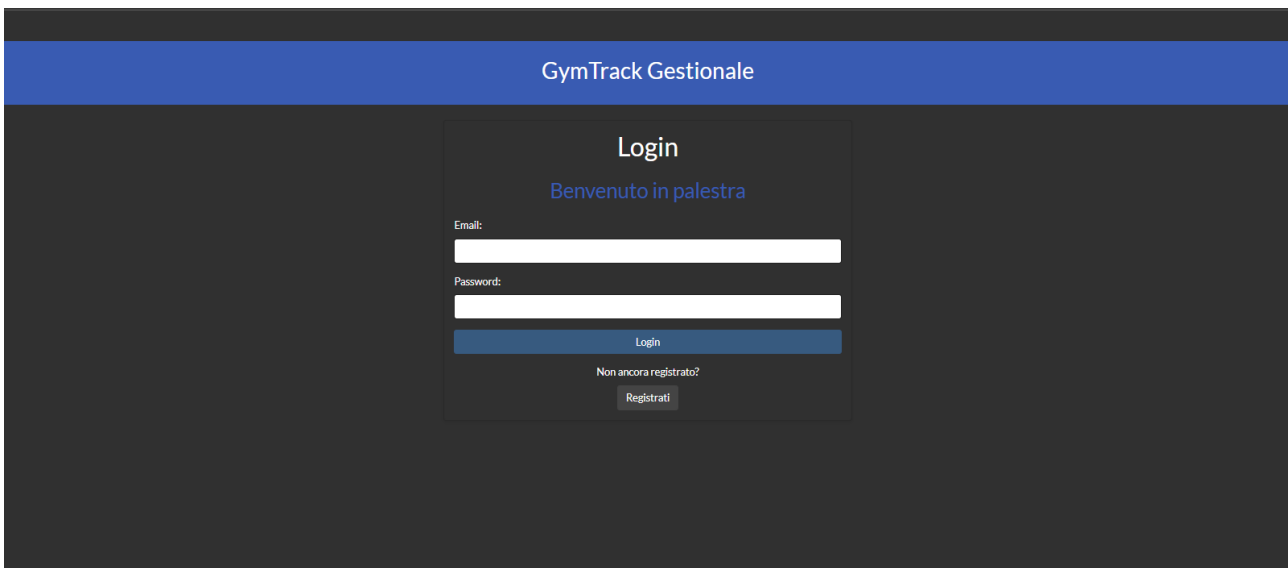
- Nella base di dati, il campo *creata_da* presente nella tabella Scheda deve riferirsi sempre a un utente registrato con ruolo “Personal Trainer”. Questo significa che esiste un vincolo di integrità referenziale verso la tabella Utente, che garantisce che l’utente creatore della scheda sia effettivamente presente nel sistema. Inoltre, a livello logico, è stabilito che solo i Personal Trainer possano creare nuove schede, così da attribuire correttamente le responsabilità all’interno della piattaforma.
- Ogni scheda è destinata esclusivamente a utenti che hanno ruolo “Cliente”. Nella base di dati, questo viene garantito attraverso un vincolo di chiave esterna sul campo *email* della scheda, che assicura che l’email inserita corrisponda a un utente esistente, permettendo di assegnare la scheda a un destinatario valido.
- La relazione tra Utente e Scheda_integratori è riservata ai soli utenti con ruolo Cliente. Questo significa che, nella base di dati, esiste un vincolo di integrità referenziale che collega la scheda integratori a un utente registrato, garantendo che ogni scheda integratori sia effettivamente associata a un cliente reale.
- La tabella Preferenza, che serve a memorizzare gli esercizi salvati come preferiti dagli utenti, è strutturata in modo tale da consentire l’inserimento sia per Clienti che per Personal Trainer. La base di dati mantiene l’integrità referenziale verso le tabelle Utente ed Esercizio, assicurando che vengano salvati solo esercizi presenti nel catalogo e che siano collegati a un utente esistente.
- Ogni esercizio utilizzato in una scheda o salvato come preferito deve necessariamente esistere nel catalogo ufficiale (cioè nella base di dati). Questo requisito viene garantito nel database attraverso vincoli di chiave esterna che collegano le tabelle delle schede e delle preferenze alla tabella Esercizio, impedendo l’utilizzo di esercizi non registrati.
- All’interno di una stessa scheda, per una determinata giornata di allenamento, uno stesso esercizio non può comparire più di una volta. Per assicurare questo vincolo, la base di dati implementa un vincolo UNIQUE su un gruppo di attributi composto da *scheda_id*, *nome_giornata* ed *esercizio_id*, così da evitare duplicazioni accidentali.
- Ogni integratore associato a una scheda integratori deve già essere presente nella tabella Integratore. Questo è garantito dalla presenza di una chiave esterna nella tabella Scheda_integratori, che assicura l’utilizzo esclusivo di prodotti realmente registrati nel database.
- Infine, ogni scheda d’allenamento deve essere assegnata a un cliente già registrato nel sistema. La base di dati garantisce l’esistenza di questo destinatario tramite la chiave esterna verso la tabella Utente, assicurando che nessuna scheda venga creata senza un cliente valido.

3. Implementazione Sistema Informativo

L'implementazione del sistema informativo è stata realizzata usando Django, un framework di Python, che permette di sviluppare applicativi web in modo rapido e ordinato, grazie a una struttura ben definita che offre moltissimi strumenti già integrati; inoltre, è altamente scalabile. Il sistema è stato progettato per gestire i dati secondo il modello logico precedentemente definito e per offrire un gruppo di funzionalità diverse e comode ai vari tipi di utenti.

In futuro contiamo che il sistema verrà aggiornato con altre funzionalità e altri tipi di utenti, il ciò giustificato dalle scelte fatte in precedenza sul modello entità/relazione utilizzato.

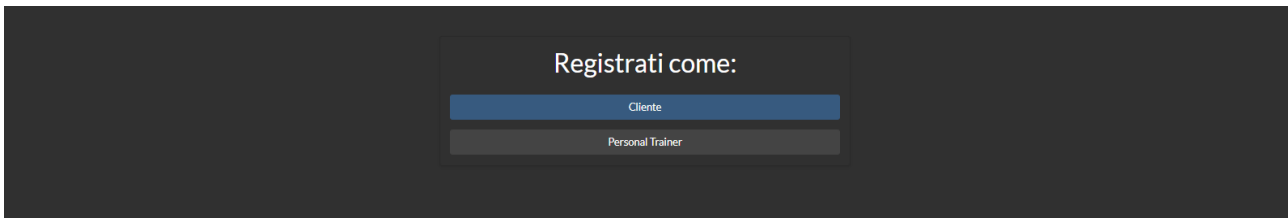
- **Login**



Appena si aprirà il sito la prima schermata consultabile sarà quella di login. La schermata di login di GymTrack riconosce da sola se l'utente che sta accedendo si tratta di un cliente o di un personal trainer. Si è pensato a questa soluzione così che l'accesso degli utenti sia il più semplice e veloce possibile.

Se l'utente non è ancora registrato cliccherà sul tasto "Registrati" e gli si aprirà un'altra schermata.

- **Scelta della registrazione**



Il cliente può registrarsi come Personal Trainer, se è autorizzato e possiede l'email col giusto dominio, oppure come Cliente semplicemente cliccando uno dei due pulsanti che li rimanderà alla vera e propria pagina di registrazione.

- **Registrazione degli utenti**

The image shows two registration forms side-by-side. The top form is titled 'Registrazione Cliente' and has fields for Email, Username, and Password, followed by a blue 'Registrati' button. The bottom form is titled 'Registrazione Personal Trainer' and has fields for Email (@PT.com), Username, and Password, followed by a blue 'Registrati' button. Both forms have a dark background and white text.

Le due schermate di registrazioni sono molto simili tra di loro, scelta fatta per mantenere linearità e coerenza nel sistema.

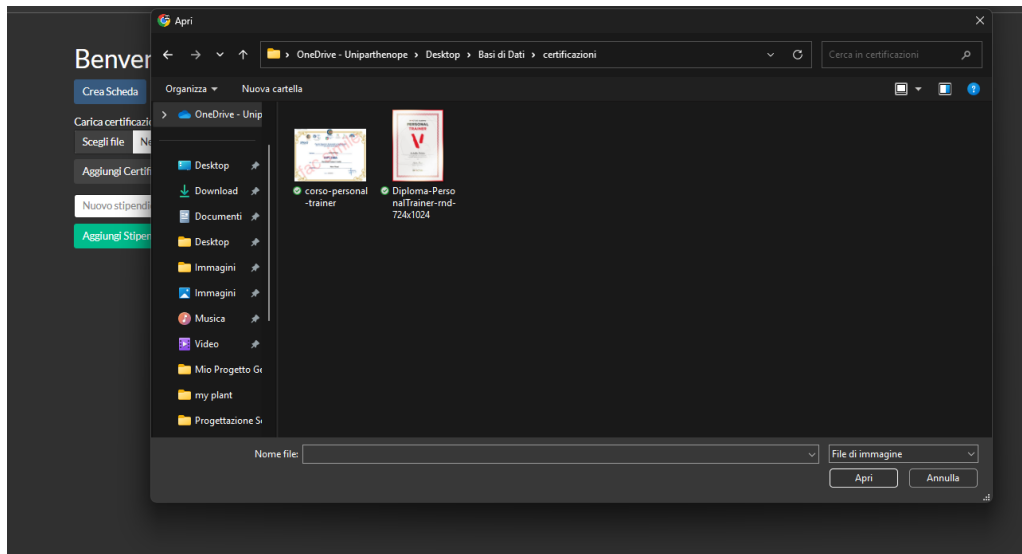
Una volta messe tutte le credenziali richieste e cliccato il pulsante blu “Registrati” l’utente verrà indirizzato alla pagina di login, dove inserirà di nuovo le sue credenziali e poi potrà accedere comodamente al sito.

Le credenziali verranno salvati all’interno della base di dati hashate con SHA256, così da garantire la sicurezza dell’utente.

- **Sistema lato Personal Trainer**

The image shows a dashboard for a Personal Trainer. It has a dark background with white text. At the top, it says 'Benvenuto Trainer, pierPT!'. Below this, there are three sections: 'Crea Scheda' (Create Schedule), 'Carica certificazione (foto):' (Upload certification (photo):) with a file selection button and a message 'Nessun file selezionato', and 'Nuovo stipendio' (New salary) with an 'Aggiungi Stipendio' (Add Salary) button.

Dopo l'accesso il Personal Trainer vedrà questa schermata dove poter inserire facilmente stipendio e certificazioni, sezione che gli consentirà di tenere i suoi attestati a portata di mano e che, se vorrà, gli farà guadagnare credibilità nel suo campo.



Questa è la schermata che apparirà quando il trainer cliccherà su “Scegli file”, successivamente potrà cercare fra le cartelle del suo dispositivo e selezionare l’attestato.

Cliccando il pulsante “Crea Scheda” al PT si aprirà un’altra pagina dove poter creare una scheda per un cliente selezionato.

- **Creazione schede**

La pagina di creazione delle schede è fatta in questo modo, si è cercato di fare una schermata intuitiva e facile da usare.

The screenshot shows a web form titled "Crea una nuova scheda d'allenamento". It includes input fields for "Data inizio:" and "Data fine:" with date pickers, and an "Email utente:" field with an email icon. Below these are two sections for adding workouts. The first section has a "Nome giornata:" field with "petto e bicipiti" entered, and a table with columns "Selezione esercizio:", "Serie:", "Ripetizioni:", and "Recupero (sec:)", containing "Panca Piana", "5", "5", and "180" respectively. The second section is identical but empty. Each section has an "Aggiungi esercizio" button. At the bottom, there is an "Aggiungi nome giornata" button and a green "Crea Scheda" button.

Infatti, ogni volta che il PT cliccherà su “aggiungi esercizio” o su “aggiungi giornata” si creerà subito sotto una nuova casella per aggiungere il prossimo esercizio o una nuova casella per aggiungere un’altra giornata d’allenamento. Se le nuove caselle non vengono riempite, restano vuote e non verranno salvate, così che il cliente non avrà spazi vuoti inutili all’interno della sua scheda d’allenamento; inoltre avranno tutte un pulsante vicino per essere eliminate nel caso in cui il PT abbia sbagliato ad inserirle.

Quando il pulsante verde “Crea Scheda” verrà premuto la scheda sarà salvata ed associata correttamente all’utente tramite la sua email.

- **Sistema lato Cliente**

The screenshot shows a client interface with a dark background. A white box contains the text "Ciao vi.delucia" in a large font. Below it is a dropdown menu showing "Scheda #2" with a downward arrow. To the right of the dropdown are two blue buttons: "Visualizza" and "Aggiungi integratore".

Dopo aver fatto l’accesso al cliente verrà mostrata questa schermata in cui potrà scegliere una fra le schede a lui associate e successivamente premere il pulsante “Visualizza”.

Ciao vi.delucia

Scheda #2

Visualizza

Aggiungi integratore

Esercizi della scheda 2

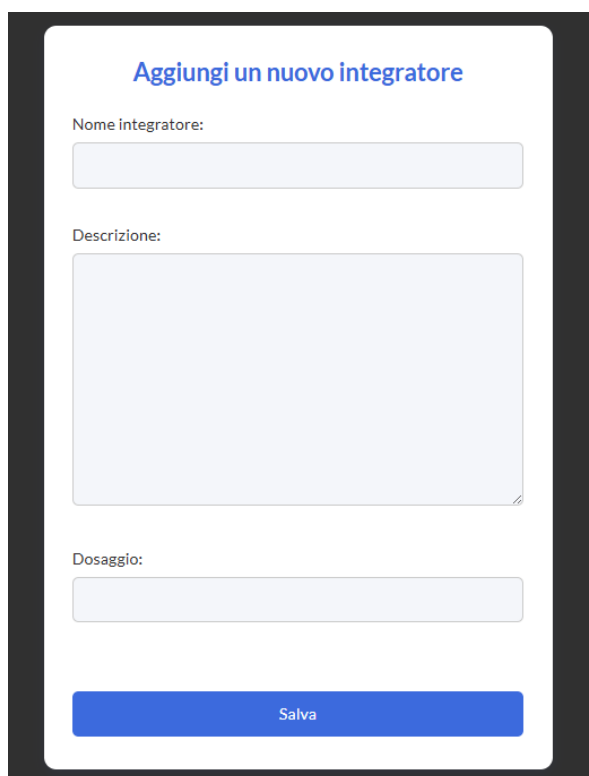
Giornata	Esercizio	Serie	Ripetizioni	Recupero (sec)
petto e bicipiti	Panca Piana	5	5	180
petto e bicipiti	Chest press	3	8	90
petto e bicipiti	Push up	3	12	60
petto e bicipiti	Hammer curl	4	8	120
schiena e tricipiti (richiamo spalle)	Rematore con manubrio	5	5	180
schiena e tricipiti (richiamo spalle)	Lat machine	3	8	90
schiena e tricipiti (richiamo spalle)	Scrollate con manubri	3	10	60
schiena e tricipiti (richiamo spalle)	French press	4	8	120
gambe, spalle e addome	Squat	5	5	180
gambe, spalle e addome	Stacco da terra	5	5	180
gambe, spalle e addome	Affondi	3	10	60
gambe, spalle e addome	Alzate laterali	3	10	60
gambe, spalle e addome	Crunch	3	20	60

pierPT

Una volta premuto il pulsante il cliente potrà visualizzare la sua scheda completa con tutte le informazioni. Inoltre, in fondo alla scheda si trova l'username del Personal Trainer che l'ha creata, per tenere traccia di chi ha fatto cosa e per far sapere al cliente a chi rivolgersi in palestra in caso di dubbi.

Infine, premendo il tasto "Aggiungi integratore" il cliente verrà rimandato ad un'altra pagina del sistema.

- **Aggiungere un integratore**



The form is titled "Aggiungi un nuovo integratore" in blue text. It contains three input fields: "Nome integratore:" (a single-line text box), "Descrizione:" (a large multi-line text area), and "Dosaggio:" (a single-line text box). At the bottom, there is a blue button labeled "Salva".

Il cliente in questa pagina potrà aggiungere un integratore con tanto di dosaggio da lui consumato, così che possa tenere sotto controllo anche i suoi supplementi.

4. Gestione delle sessioni

Nel progetto **GymTrack** è stato implementato il sistema di sessioni integrato di Django per garantire la persistenza dello stato tra richieste HTTP stateless. In un'applicazione web tradizionale, ogni richiesta inviata al server è indipendente dalle altre, e il server non mantiene informazioni sulle richieste precedenti. Questo rappresenta un limite quando si desidera realizzare funzionalità come l'autenticazione, che richiedono di riconoscere un utente come già loggato durante la navigazione.

Per risolvere questo problema, GymTrack utilizza le **sessioni**, che consentono di: Identificare l'utente in modo univoco garantire la sicurezza, gestire informazioni temporanee.

In Django, le sessioni si basano su un middleware dedicato e possono utilizzare diversi backend. In questo progetto è stato scelto il backend predefinito basato su database SQLite. L'integrazione delle sessioni è risultata fondamentale per implementare:

- Il **meccanismo di login/logout**, garantendo l'accesso alle aree riservate solo a utenti autenticati.
- La **personalizzazione dell'interfaccia**, mostrando funzionalità diverse a seconda del ruolo utente (Cliente o Personal Trainer).
- La **comunicazione utente-sistema**, tramite la gestione di messaggi temporanei salvati in sessione e mostrati successivamente nelle pagine di redirect.

L'engine utilizzato da django è `django.contrib.sessions.backends.db` per memorizzare i dati di sessione all'interno del database.

La gestione è abilitata tramite:

- L'app `django.contrib.sessions` inserita in `INSTALLED_APPS`.
- Il middleware `SessionMiddleware` incluso nella lista dei middleware attivi.

Login

Quando un utente effettua il login con credenziali corrette, il sistema salva nella sessione i seguenti dati:

```
request.session['user_email'] = utente.email
request.session['username'] = utente.username
```

Queste informazioni vengono utilizzate per:

- Identificare l'utente nelle pagine successive senza dover ripetere l'autenticazione.
- Mostrare contenuti personalizzati in base al ruolo dell'utente (ad esempio differenziare le pagine visibili per Clienti e Personal Trainer).

Accesso alle pagine riservate

Per ogni view che richiede un utente autenticato, viene effettuato un controllo preliminare verificando la presenza di *username* nella sessione:

```
username = request.session.get('username')
if not username:
    return redirect('login')
```

Se la chiave non è presente, l'utente viene reindirizzato alla pagina di login per effettuare nuovamente l'autenticazione.

Logout

La funzione di logout elimina tutti i dati salvati nella sessione con il metodo:

```
request.session.flush()
```

In questo modo, alla disconnessione, la sessione viene completamente cancellata e l'utente dovrà autenticarsi di nuovo per accedere alle sezioni protette.

Messaggi temporanei

In alcune pagine (ad esempio la sezione di creazione schede) vengono utilizzate variabili di sessione per mostrare messaggi temporanei di conferma o errore dopo un'operazione.

Il sistema salva nella sessione messaggi come:

```
request.session['success_message'] = "Certificazione aggiunta con successo!"
```

E li recupera nella view successiva con:

```
success_message = request.session.pop('success_message', None)
```

In questo modo i messaggi vengono mostrati una sola volta e poi eliminati dalla sessione.

5. Difesa agli attacchi

La **SQL Injection** è una vulnerabilità di sicurezza, tipica delle applicazioni web, che si verifica quando un input fornito dall'utente viene inserito all'interno di una query SQL senza un'adeguata validazione o parametrizzazione. In questo modo, un attaccante può manipolare la query ed eseguire comandi SQL arbitrari sul database, accedendo, modificando o eliminando dati sensibili. Questo tipo di attacco può compromettere la riservatezza, l'integrità e la disponibilità delle informazioni gestite dall'applicazione web.

Codice vulnerabile:

```
import sqlite3

def authenticated_vulnerabile(email, password):
    conn = sqlite3.connect('db.sqlite3')
    cursor = conn.cursor()

    query = f"SELECT * FROM Utente WHERE email = '{email}' AND password = '{password}'"
    cursor.execute(query)
    result = cursor.fetchone()

    conn.close()
    return result is not None
```

In questo caso, la query viene costruita **concatenando direttamente le stringhe dell'input utente**. Questo significa che se un utente inserisce un input malevolo, può alterare completamente la logica della query.

query = f"SELECT * FROM Utente WHERE email = '{email}' AND password = '{password}'"

la vulnerabilità sta proprio in questa riga di codice.

Immaginiamo che un utente inserisca come email: **' OR 1=1** – insieme ad una password qualsiasi, la query SQL finale costruita dal sistema sarà: **SELECT * FROM Utente WHERE email = " OR 1=1 --' AND password = '...' .** Possiamo notare come il frammento **OR 1=1** rende la condizione sempre vera, anche senza l'inserimento di una password, indipendentemente dai dati presenti nel database; inoltre, il simbolo **--** indica in SQL l'inizio di un commento. Quindi, tutto ciò che segue viene ignorato, inclusa la parte della query che controlla la password.

In pratica, **l'utente ottiene l'accesso senza conoscere email o password valide**. Questo è il tipico esempio di SQL Injection, in cui si sfrutta la concatenazione diretta di input utente alla query per aggirare i controlli di sicurezza.

Come è protetto il sistema

Nel progetto questa vulnerabilità è coperta grazie all'utilizzo dell'ORM di Django, come nella riga di codice: **utente = Utente.objects.get(email=email)**

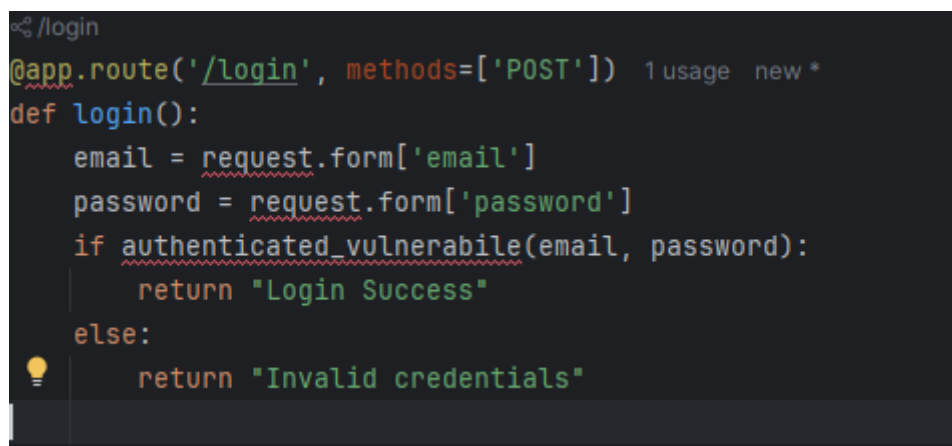
- L'ORM di Django, quando costruisce le query, utilizza **query parametrizzate**.
- I parametri vengono passati separatamente al database, che li interpreta come **valori** e non come parte della struttura della query SQL.
- Anche se un utente dovesse inserire input malevolo, verrebbe trattato come una normale stringa di testo, senza alcun effetto sulla logica della query.

Quando utilizzi metodi come **get()** (o anche **filter()** ad esempio) dell'ORM, Django genera automaticamente una query parametrizzata simile a questa: **SELECT * FROM Utente WHERE email = %s AND password = %s**, i valori reali di email e password vengono poi passati al database separatamente e sostituiti solo in fase di esecuzione, senza mai diventare parte della query SQL come codice eseguibile.

Simulazione SQL Injection con SQLmap

sqlmap è uno strumento open source di penetration testing, scritto in Python, progettato per automatizzare il processo di rilevamento e sfruttamento delle vulnerabilità di SQL Injection all'interno delle applicazioni web. È uno degli strumenti più utilizzati dagli ethical hacker e dai security analyst per valutare la sicurezza dei database esposti attraverso interfacce web. Nel progetto verrà usato proprio per il **rilevamento delle vulnerabilità di SQL Injection**.

Per usare correttamente sqlmap è stata scritta un'altra funzione di login che usi la funzione vulnerabile di prima, *authenticated_vulnerable*



```
#!/login
@app.route('/login', methods=['POST']) 1 usage new *
def login():
    email = request.form['email']
    password = request.form['password']
    if authenticated_vulnerable(email, password):
        return "Login Success"
    else:
        return "Invalid credentials"
```

In questo caso, sqlmap può attaccare il parametro **email** o **password**.

Dalla macchina dell'attaccante, sqlmap viene eseguito con: **sqlmap -u "http://http://127.0.0.1:8000/login" --data "email=prova&password=test" --batch --risk=3 --level=5**

- **-u "http:// http://127.0.0.1:8000/login"** specifica l'URL dell'endpoint da testare
- **--data "email=prova&password=test"** indica che è una POST con questi parametri
- **--batch** esegue senza chiedere conferme
- **--risk=3 --level=5** aumentano la profondità dei test e l'aggressività

Cosa fa sqlmap

Invia richieste modificate al parametro email e password, inserendo payload SQL per capire se la query è vulnerabile.

Se rileva la vulnerabilità, può:

- Leggere l'intero database (**--dump**)
- Ottenere i nomi di tutte le tabelle (**--tables**)
- Ottenere il nome del database in uso (**--dbs**)

Visto che *authenticated_vulnerable* inserisce l'input dell'utente direttamente nella query e non usa query parametrizzate sqlmap riesce ad iniettare direttamente comandi SQL

Esecuzione di un test con sqlmap

L'output che ci uscirà usando il comando spiegato prima sarà:

[INFO] testing for SQL injection on parameter 'email'

- Inizia i test di SQL Injection sul parametro email.

[INFO] testing 'Boolean-based blind - WHERE or HAVING clause'

[PAYLOAD] email=prova' AND 1=1 -- &password=test

[PAYLOAD] email=prova' AND 1=2 -- &password=test

[RESULT] response contents differ => VULNERABLE

- sqlmap invia due richieste:
 - Una con AND 1=1 (sempre vera)
 - Una con AND 1=2 (sempre falsa)

Poiché le risposte differiscono, conferma la vulnerabilità **boolean-based blind SQL Injection**.

[INFO] testing 'Time-based blind - SQLite > 2.0.0'

[PAYLOAD] email=prova' AND (SELECT CASE WHEN (1=1) THEN randomblob(1000000000) ELSE 1 END) -- &password=test

[RESULT] response time increased by 5 seconds => VULNERABLE

- Usa un payload che genera un grande blob di dati in SQLite per rallentare la risposta. Il tempo di risposta aumenta, confermando la vulnerabilità **time-based blind SQL Injection**.

[INFO] testing 'Error-based - SQLite inline queries'

[PAYLOAD] email=prova' AND (SELECT 1/0) -- &password=test

[RESULT] database returned division by zero error => VULNERABLE

- Forza un errore di divisione per zero nel DB. La ricezione dell'errore conferma la vulnerabilità **error-based SQL Injection**, utile per estrarre dati attraverso messaggi di errore.

[INFO] testing 'UNION query (NULL) - 1 to 10 columns'

[PAYLOAD] email=prova' UNION SELECT NULL, sqlite_version(), NULL -- &password=test

[RESULT] database version string detected => SQLite 3.39.4

- Usa una query UNION per leggere informazioni extra (qui la versione di SQLite). Questo dimostra la vulnerabilità **union-based SQL Injection** e la capacità di estrarre dati arbitrari.

[INFO] testing 'Stacked queries'

[PAYLOAD] email=prova'; DROP TABLE Utente; -- &password=test

[RESULT] stacked queries supported (table dropped)

- Testa se il database accetta **stacked queries** (più query in una richiesta). Qui sqlmap elimina la tabella Utente, mostrando l'estrema gravità della vulnerabilità.

[INFO] fetching database names

available databases [1]:

[*] 'db.sqlite3'

- Estrae i nomi dei database presenti

[INFO] fetching tables for database 'db.sqlite3'

- Elenca le tabelle presenti nel database Utente.

[INFO] fetching data from table 'Utente'

- Dumps i dati reali contenuti nella tabella Utente, esponendo tutte le credenziali.

Questa simulazione dimostra come sqlmap:

1. Prova diverse tecniche di SQL Injection: boolean-based, time-based, error-based, union-based, stacked queries, out-of-band.
2. Individua rapidamente la vulnerabilità nella funzione authenticated_vulnerabile.

3. Può estrarre **tutti i dati del database** o eseguire comandi dannosi.

L'uso dell'ORM di Django (o di query parametrizzate) **protegge completamente** da questi attacchi, evitando che l'input utente venga interpretato come parte della query SQL.

Difesa agli attacchi di forza bruta

Gli attacchi brute force sono una delle tecniche più basilari, ma anche più diffuse, per violare la sicurezza informatica di sistemi e account. Il termine *brute force* significa “forza bruta” e descrive bene il metodo utilizzato: consiste infatti nel provare sistematicamente **tutte le combinazioni possibili** fino a trovare quella corretta.

Nel sistema gli attacchi brute force vengono prevenuti grazie a varie funzionalità, che introduce Django, usate nel login. Per utilizzare queste funzionalità è necessario importare la libreria *cache* scrivendo: **from django.core.cache import cache**

```
def login(request): 1 usage 2 De Lucia092 *
    if request.method == 'POST':
        email = request.POST.get('username')
        password = request.POST.get('password')
        if not email or not password:
            return render(request, template_name='index.html', context={'error_message': "Compila tutti i campi"})

        # Recupera tentativi e blocco dalla cache
        failed_attempts = cache.get(key=f'failed_{email}', default=0)
        block_until = cache.get(f'block_{email}')

        now = timezone.now()
        if block_until and now < block_until:
            return render(request, template_name='index.html', context={
                'error_message': "Troppi tentativi falliti. Riprova tra 2 minuti."
            })

        hashed_pass = hashlib.sha256(password.encode()).hexdigest()
        if authenticated(email, hashed_pass):
            cache.delete(f'failed_{email}')
            cache.delete(f'block_{email}')
            utente = Utente.objects.get(email=email)
            request.session['user_email'] = utente.email
            request.session['username'] = utente.username
            if email.lower().endswith('@pt.com'):
                return render(request, template_name='creazione_schede.html', context={'username': utente.username})
            return redirect('mie_schede')
        else:
            failed_attempts += 1
            cache.set(key=f'failed_{email}', failed_attempts, timeout=120)
            if failed_attempts >= 3:
                block_time = now + timezone.timedelta(minutes=2)
                cache.set(key=f'block_{email}', block_time, timeout=120)
                cache.set(key=f'failed_{email}', value=0, timeout=120)
                return render(request, template_name='index.html', context={
                    'error_message': "Troppi tentativi falliti. Riprova tra 2 minuti."
                })
            return render(request, template_name='index.html', context={'error_message': "Credenziali non valide, riprova"})
    else:
        return render(request, template_name='index.html', context={'error_message': "Metodo non valido"})
```

- La funzione usa un contatore di tentativi falliti: **failed_attempts = cache.get(f'failed_{email}', 0)** e ne memorizza il numero, ogni volta che l'utente sbaglia la password il contatore viene incrementato.
- Blocca temporaneamente l'utente per 2 minuti dopo 3 tentativi falliti grazie a questo controllo:

```
if failed_attempts >= 3:  
    block_time = now + timezone.timedelta(minutes=2)  
    cache.set(f'block_{email}', block_time, timeout=120)  
    cache.set(f'failed_{email}', 0, timeout=120)
```

- Ogni volta che un utente prova ad accedere, viene controllato se esiste un blocco attivo, se esiste allora l'utente **non può effettuare altri tentativi** fino alla scadenza del blocco.
- Se il nuovo login, dopo i 2 minuti, va a buon fine, il contatore di tentativi falliti e il blocco vengono eliminati:
cache.delete(f'failed_{email}')
cache.delete(f'block_{email}')

6. Utenti

vincydelucia04@gmail.com | Parthenope2025!

pierpaolo@pt.com | Parthenope2025!