1506
UNIVERSITÀ
DEGLI STUDI
DI URBINO
CARLO BO

Relazione per il Progetto del Corso di Basi di Dati

Diego Chiodi, Andrii Ursu

A.A. 2024/2025

1. Introduzione

1.1 Scopo del progetto

L’obiettivo di questo progetto è realizzare una semplice applicazione web chiamata **Meal Planner**, che permetta agli utenti di pianificare i pasti settimanali in modo personalizzato. Il sistema aiuta a organizzare colazione, pranzo, merenda e cena per ogni giorno della settimana, tenendo traccia dei piatti scelti e dei valori nutrizionali complessivi.

2. Analisi dei Requisiti

2.1 Definizione del documento di specifica

Il progetto consiste nello sviluppo di un **Meal Planner multiutente**, accessibile tramite login o registrazione, che consente di:

- Visualizzare e gestire piatti e ingredienti;
- Organizzare i pasti su una **griglia settimanale interattiva**;
- Calcolare i **valori nutrizionali complessivi** (proteine, carboidrati, calorie) per singolo giorno o per l'intera settimana;
- Creare nuovi piatti e ingredienti, assegnando la **visibilità** (pubblica o privata).

È stato sviluppato **sia il backend** che un interfaccia **frontend**, utilizzando:

- HTML, CSS, JavaScript per la parte client;
- Python per la logica applicativa;
- MySQL per la persistenza dei dati.

2.2 Decomposizione del testo in gruppi di fasi

Il progetto è stato suddiviso nelle seguenti fasi principali:

1. Autenticazione e gestione utenti

- Registrazione e login multiutente.
- Crittografia e verifica delle credenziali.
- Gestione della visibilità dei contenuti per utente.

2. Progettazione del database relazionale

- Modellazione entità come *piatti*, *ingredienti, piatti ingredienti, utenti*, *planner.*
- Introduzione di campi come **`validato`**, **`utente_id`** per la gestione fine della visibilità.

3. Backend applicativo (Python + SQL)

- API per creare, modificare e recuperare piatti e ingredienti.
- Calcolo dinamico dei valori nutrizionali per giorno/settimana.
- Associazione di piatti ai giorni tramite planner.

4. Frontend interattivo

- Griglia settimanale dinamica.
- Modal per la creazione di nuovi piatti/ingredienti.
- Funzionalità di drag & drop e aggiornamento in tempo reale via JavaScript.
- Visualizzazione di statistiche nutrizionali con pulsanti dedicati.

2.3 Definizione delle operazioni sui dati

Le principali operazioni eseguite dal sistema sui dati sono:

- **Creazione e gestione utenti**: inserimento di nuovi utenti e login con verifica.

- **Creazione piatti**: definizione di un piatto con uno o più ingredienti e loro quantità; visibilità pubblica o privata.

- **Creazione ingredienti**: inserimento di ingredienti con valori nutrizionali; visibilità pubblica o privata.

- **Gestione planner(griglia) settimanale**:

    - Inserimento dei piatti nel planner tramite drag & drop.
    - Salvataggio automatico nel database della relazione giorno-pasto-piatto-utente.

- **Calcoli nutrizionali**:

    - Somma dei valori nutrizionali per singolo giorno (query che aggrega gli ingredienti dei piatti pianificati).
    - Somma per l'intera settimana.

3. Progettazione Concettuale

3.1 Strategia di progettazione concettuale

Per la progettazione concettuale del sistema si è adottato un approccio **top-down**, partendo dall'analisi dei requisiti funzionali per poi individuare le entità principali, le loro relazioni e le operazioni che il sistema deve supportare.

L'obiettivo era modellare un sistema di **meal planning multiutente**, dove ogni utente potesse:

- gestire piatti e ingredienti propri o pubblici,
- pianificare i pasti settimanalmente su una griglia interattiva,
- visualizzare i valori nutrizionali giornalieri e settimanali.

Una volta definiti i casi d'uso principali, si è passati alla modellazione concettuale attraverso un **diagramma ER**, che ha evidenziato entità, attributi e relazioni cardine, come quella molti-a-molti tra piatti e ingredienti e la relazione tra utente, giorno e piatto nel planner.

3.2 Dizionario delle entità

Entità	**Descrizione**
Utenti	Rappresenta un utente registrato al sistema. Ogni utente ha username, email e password.
Piatti	Piatti salvati nel sistema, possono essere pubblici o privati. Ogni piatto è associato ad un utente (o admin come "null").
Ingredienti	Ingredienti utilizzabili nei piatti. Possono essere pubblici o privati. Ogni ingrediente contiene i valori nutrizionali base per unità di misura.
Planner	Rappresenta la pianificazione dei pasti di un utente in un giorno specifico. Ogni entry contiene il giorno, il tipo di pasto e il piatto associato.

3.3 Dizionario delle relazioni

Relazione	Descrizione
piatti_ingredienti	Relazione molti-a-molti tra piatti e ingredienti, con quantità specifica per ogni ingrediente usato in un piatto. Permette il calcolo nutrizionale aggregato.
planner	Relazione tra utente, data, tipo di pasto(colazione, pranzo, merenda, cena) e piatto assegnato. Costituisce il cuore del meal planner(griglia) settimanale.

3.3 Diagramma Entità-Relazioni

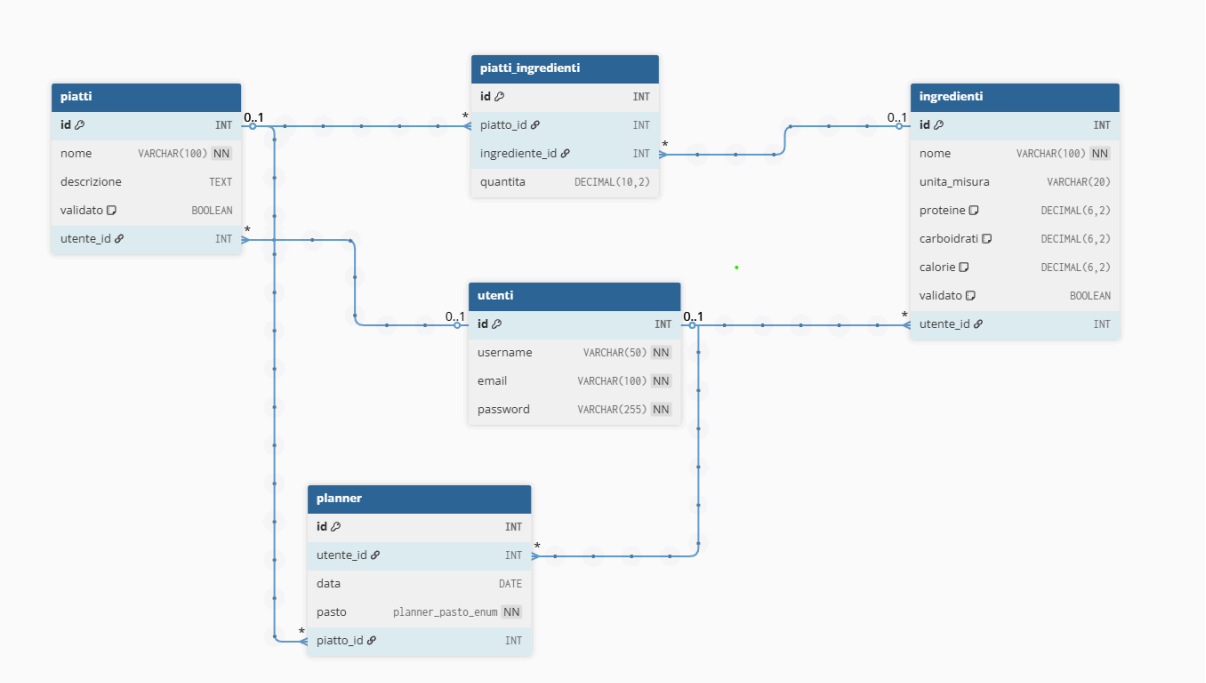


Tabella delle relazioni:

piatti **(0..1)**	planner **(N)**
utenti **(0..1)**	piatti **(N)**
utenti **(0..1)**	planner **(N)**
utenti **(0..1)**	ingredienti **(N)**
piatti **(N)**	ingredienti **(N)**
piatti_ingredienti	

Tabella dei vincoli di integrità(chiavi):

piatti(id) **pk**	planner(piatto_id) **fk**
utenti(id) **pk**	piatti(utente_id) **fk**
utenti(id) **pk**	planner(utente_id) **fk**
utenti(id) **pk**	ingredienti(utente_id) **fk**
piatti(id) **pk**	ingredienti(id) **fk**
piatto_id, ingrediente_id	

3.5 Business Rules

Il sistema implementa le seguenti regole di business:

- Ogni utente può creare piatti e ingredienti, che sono di default **privati**.
- Gli ingredienti creati dagli utenti vengono contrassegnati con **`validato = 0`**, mentre quelli pubblici hanno **`validato = 1`** e sono globalmente disponibili.
- I piatti, analogamente, possono essere pubblici o privati a seconda del campo **`validato`** e dell'associazione con l'utente.
- È consentita l'associazione di più ingredienti a un piatto e la specifica della quantità.
- Un utente può pianificare un piatto in una specifica data e pasto. Nello stesso giorno/pasto possono esserci più piatti.
- I valori nutrizionali giornalieri e settimanali sono **derivati dinamicamente** sommando le quantità degli ingredienti nei piatti pianificati.

3.5.1 Vincoli di integrità

- **Chiavi primarie:** su ogni tabella (id).
- **Chiavi esterne:** per collegare utenti con piatti e ingredienti, piatti con ingredienti e planner.
- **NOT NULL:** applicato solo dove necessario (es. email utente, nome piatto, data nel planner).
- **UNIQUE:** sull'email utente.
- **DEFAULT:** su campi come **`validato`**, **`proteine`**, **`calorie`** per fornire valori predefiniti coerenti.

3.5.2 Regole di derivazione

- **Calorie giornaliere:** somma delle calorie di tutti i piatti pianificati in un dato giorno, calcolate in base agli ingredienti e alle quantità.
- **Macronutrienti settimanali:** aggregazione delle proteine/carboidrati/calorie dei piatti pianificati tra due date.

4. Progettazione Logica

4.1 Ristrutturazione del modello concettuale

La ristrutturazione del modello concettuale è avvenuta per prepararlo alla fase di traduzione nel modello logico relazionale. In questa fase sono stati effettuati i seguenti interventi:

- **Normalizzazione delle entità**: tutte le entità sono in **terza forma normale (3NF)**, evitando duplicazioni e ridondanze.

- **Risoluzione delle relazioni molti-a-molti**: la relazione tra *piatti* e *ingredienti* è stata gestita mediante una **tabella intermedia (`piatti_ingredienti`)** che consente anche di memorizzare la quantità di ciascun ingrediente in ogni piatto.

- **Gestione della multiutenza**: ogni piatto e ogni ingrediente è associato a un utente creatore tramite una foreign key. Se il valore è **`NULL`**, l'elemento è stato creato da un "admin" e considerato pubblico.

- **Visibilità pubblica/privata**: per evitare una tabella separata, la visibilità è gestita direttamente tramite un attributo booleano **`validato`**, con valore `1` per gli elementi pubblici e `0` per quelli privati.

4.2 Analisi delle ridondanze

Nel modello logico finale sono state **minimizzate le ridondanze**, seguendo questi criteri:

- I **valori nutrizionali complessivi dei piatti o dei planner** non sono memorizzati, ma **calcolati dinamicamente** in base agli ingredienti e alle quantità, evitando così ridondanza e rischio di inconsistenza.

- Il campo **`utente_id`** presente sia in **`piatti`** che in **`ingredienti`** è necessario per la gestione della visibilità e non è ridondante.

- I nomi degli ingredienti o dei piatti non sono duplicati, perché ogni voce è identificata in modo univoco tramite un id.

- La relazione **`planner`** collega un solo piatto per riga (per un determinato giorno e pasto), quindi non serve una struttura nidificata.

Conclusione: non sono presenti ridondanze significative o non giustificate. Il modello è stato progettato per essere efficiente, facilmente estendibile e conforme ai principi della progettazione relazionale.

4.3 Traduzione nel modello logico

Di seguito viene riportata la **traduzione del modello concettuale nel modello logico**, espressa mediante **DDL SQL** :

```sql
-- Utenti
CREATE TABLE utenti (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(50) NOT NULL,
  email VARCHAR(100) NOT NULL UNIQUE,
  password VARCHAR(255) NOT NULL
);

-- Piatti disponibili : pubblici o privati
CREATE TABLE piatti (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nome VARCHAR(100) NOT NULL,
  descrizione TEXT,
  validato BOOLEAN DEFAULT 1, -- 0 = privato, 1 = pubblico
  utente_id INT, -- NULL se pubblico creato da admin
  FOREIGN KEY (utente_id) REFERENCES utenti(id) ON DELETE CASCADE
);

-- Ingredienti
CREATE TABLE ingredienti (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nome VARCHAR(100) NOT NULL,
  unita_misura VARCHAR(20),
  proteine DECIMAL(6,2) DEFAULT 0,   -- quantità in grammi per unità di misura
  carboidrati DECIMAL(6,2) DEFAULT 0,
  calorie DECIMAL(6,2) DEFAULT 0,
  validato BOOLEAN DEFAULT 1, -- 0 = privato, 1 = pubblico
  utente_id INT, -- NULL se pubblico creato da admin
  FOREIGN KEY (utente_id) REFERENCES utenti(id) ON DELETE CASCADE
);

-- Relazione piatti-ingredienti
CREATE TABLE piatti_ingredienti (
  id INT AUTO_INCREMENT PRIMARY KEY,
  piatto_id INT,
  ingrediente_id INT,
  quantita DECIMAL(10,2), -- quantità dell'ingrediente nel piatto
  FOREIGN KEY (piatto_id) REFERENCES piatti(id) ON DELETE CASCADE,
  FOREIGN KEY (ingrediente_id) REFERENCES ingredienti(id) ON DELETE CASCADE
);

-- Planner settimanale di un utente
CREATE TABLE planner (
  id INT AUTO_INCREMENT PRIMARY KEY,
  utente_id INT,
  data DATE,
  pasto ENUM('colazione', 'pranzo', 'merenda', 'cena') NOT NULL,
  piatto_id INT,
  FOREIGN KEY (utente_id) REFERENCES utenti(id) ON DELETE CASCADE,
  FOREIGN KEY (piatto_id) REFERENCES piatti(id) ON DELETE CASCADE
);
```

5. Implementazione in MySQL

5.1 Creazione del database e delle tabelle

Il database è stato realizzato utilizzando **MySQL**. La creazione delle tabelle è avvenuta tramite uno script **SQL** che definisce in modo esplicito le relazioni, i vincoli di integrità referenziale e i default sui campi chiave.

Le tabelle principali (`utenti`, `piatti`, `ingredienti`, `piatti_ingredienti`, `planner`) sono progettate per mantenere la coerenza del sistema e supportare tutte le funzionalità offerte dall'applicazione, tra cui la multiutenza, la visibilità pubblica/privata, la composizione dei piatti e la pianificazione settimanale.

Tutti i vincoli `NOT NULL` sono stati usati solo dove strettamente necessari, mentre i campi opzionali come `descrizione`, `unita_misura` o `utente_id` possono essere `NULL`. I default impostati (es. `validato = 1`, `proteine = 0`, ecc.) garantiscono un comportamento sensato anche in assenza di valori espliciti.

5.2 Popolamento del database

5.2.1 Strategia di popolamento

Per consentire il test immediato dell'applicazione, è stato previsto un **popolamento iniziale minimo ma funzionale**, consistente in:

- **14 ingredienti di base**, con valori nutrizionali realistici
- **5 piatti pubblici predefiniti**, composti da ingredienti combinati
- Assenza di dati iniziali nel planner, in modo che l'utente possa aggiungerli dinamicamente

Questo approccio offre una base realistica ma leggera, sufficiente per provare la creazione di piatti, la pianificazione giornaliera/settimanale e il calcolo automatico dei nutrienti.
Tutti i dati aggiuntivi vengono gestiti **direttamente tramite l'interfaccia utente** dell'applicazione, che si occupa dell'inserimento nel database tramite il **backend** Python

5.2.2 Script SQL per il popolamento

```sql
INSERT INTO ingredienti (nome, unita_misura, proteine, carboidrati, calorie) VALUES
  ('Tonno', 'grammi', 25.00, 0.00, 116.00),
  ('Insalata', 'grammi', 1.00, 2.00, 15.00),
  ('Olio di oliva', 'grammi', 0.00, 0.00, 900.00),
  ('Pasta', 'grammi', 10.00, 60.00, 350.00),
  ('Pesto', 'grammi', 3.00, 3.00, 1000.00),
  ('Parmigiano', 'grammi', 35.00, 0.00, 400.00),
  ('Uova', 'grammi', 11.00, 1.00, 155.00),
  ('Patate', 'grammi', 2.00, 20.00, 73.00),
  ('Pane integrale', 'grammi', 10.00, 50.00, 280.00),
  ('Prosciutto cotto', 'grammi', 20.00, 2.00, 200.00),
  ('Formaggio a fette', 'grammi', 20.00, 3.00, 350.00),
  ('Yogurt bianco', 'grammi', 3.50, 4.00, 60.00),
  ('Banana', 'grammi', 1.00, 23.00, 89.00),
  ('Fiocchi di avena', 'grammi', 13.00, 60.00, 380.00);

INSERT INTO piatti (nome, descrizione) VALUES
  ('Insalata di Tonno', 'Tonno con insalata e un filo d'olio d'oliva.'),
  ('Pasta al Pesto', 'Pasta condita con pesto e una spolverata di parmigiano.'),
  ('Uova e Patate', 'Uova strapazzate con contorno di patate e olio d'oliva.'),
  ('Toast con Prosciutto e Formaggio', 'Pane integrale tostato con prosciutto cotto e formaggio fuso.'),
  ('Yogurt e Frutta', 'Yogurt con banana a fette e fiocchi d'avena.');


INSERT INTO piatti_ingredienti (piatto_id, ingrediente_id, quantita) VALUES
  (1, 1, 100.00),  -- Tonno
  (1, 2, 50.00),   -- Insalata
  (1, 3, 10.00);   -- Olio d'oliva


INSERT INTO piatti_ingredienti (piatto_id, ingrediente_id, quantita) VALUES
  (2, 4, 80.00),   -- Pasta
  (2, 5, 30.00),   -- Pesto
  (2, 6, 10.00);   -- Parmigiano


INSERT INTO piatti_ingredienti (piatto_id, ingrediente_id, quantita) VALUES
  (3, 7, 120.00),  -- Uova
  (3, 8, 150.00),  -- Patate
  (3, 3, 10.00);   -- Olio d'oliva


INSERT INTO piatti_ingredienti (piatto_id, ingrediente_id, quantita) VALUES
  (4, 9, 60.00),   -- Pane integrale
  (4, 10, 50.00),  -- Prosciutto cotto
  (4, 11, 30.00);  -- Formaggio a fette


INSERT INTO piatti_ingredienti (piatto_id, ingrediente_id, quantita) VALUES
  (5, 12, 150.00), -- Yogurt bianco
  (5, 13, 100.00), -- Banana
  (5, 14, 30.00);  -- Fiocchi d'avena
```

5.3 Verifica dei vincoli di integrità, view e trigger

Vincoli di integrità
Il database implementa **vincoli di integrità referenziale** tramite chiavi esterne (**`FOREIGN KEY`**) e azioni **`ON DELETE CASCADE`**, in modo da:

- Garantire la coerenza tra piatti, ingredienti e planner
- Rimuovere automaticamente i dati figli alla cancellazione di un utente, piatto o ingrediente

Esempi:

- L'eliminazione di un piatto rimuove automaticamente tutte le sue associazioni con gli ingredienti (**`piatti_ingredienti`**)
- L'eliminazione di un utente comporta la rimozione di tutti i suoi piatti, ingredienti e pianificazioni

View e Trigger

Per garantire coerenza e semplicità, i valori nutrizionali sono calcolati dinamicamente tramite **VIEW**, evitando l'uso di **TRIGGER**. Questa scelta elimina la necessità di campi ridondanti (es. calorie_totali in **`piatti`**) e mantiene il modello normalizzato.

```sql
CREATE VIEW planner_nutritional_view AS
SELECT
  planner.utente_id,
  planner.data,
  planner.pasto,
  planner.piatto_id,
  ingredienti.id AS ingrediente_id,
  (ingredienti.proteine * piatti_ingredienti.quantita / 100) AS proteine,
  (ingredienti.carboidrati * piatti_ingredienti.quantita / 100) AS carboidrati,
  (ingredienti.calorie * piatti_ingredienti.quantita / 100) AS calorie
FROM planner
JOIN piatti ON planner.piatto_id = piatti.id
JOIN piatti_ingredienti ON piatti.id = piatti_ingredienti.piatto_id
JOIN ingredienti ON piatti_ingredienti.ingrediente_id = ingredienti.id;
```

6. Query SQL

6.1 Struttura generale

Il progetto utilizza query SQL **parametrizzate**, gestite tramite Python con **MySQL Connector**, per eseguire le principali operazioni sul database:

- Inserimento di dati (`INSERT`)
- Lettura e ricerca (`SELECT`)
- Modifica (`UPDATE`)
- Eliminazione (`DELETE`)

Il codice Python per l'accesso ai dati è organizzato in moduli all'interno della cartella `models`, con un file dedicato per ogni tabella principale (utenti, ingredienti, piatti, planner). Questa scelta rende il progetto modulare e facile da mantenere.

6.2 Query sugli utenti

File: `models/utenti.py`
Principali operazioni:

- **Verifica se un utente esiste già (username o email):**
  ```
  SELECT id FROM utenti WHERE username = %s OR email = %s
  ```

- **Recupero credenziali per login:**
  ```
  SELECT id, password FROM utenti WHERE email = %s
  ```

- **Inserimento di un nuovo utente:**
  ```
  INSERT INTO utenti (username, email, password) VALUES (%s, %s, %s)
  ```

6.3 Query sugli ingredienti

File: `models/ingredienti.py`
Principali operazioni:

- **Elenco ingredienti disponibili per l'utente:**
  ```
  SELECT id, nome, unita_misura, proteine, carboidrati, calorie, validato,
  utente_id
  FROM ingredienti
  WHERE validato = 1 OR utente_id = %s
  ```

- **Inserimento di un nuovo ingrediente:**
  ```
  INSERT INTO ingredienti (nome, unita_misura, proteine, carboidrati, calorie,
  validato, utente_id)
  VALUES (%s, %s, %s, %s, %s, %s, %s)
  ```

6.4 Query sui piatti

File: `models/piatti.py`
Principali operazioni:

- **Elenco piatti disponibili:**
```
SELECT id, nome, descrizione
FROM piatti
WHERE validato = 1 OR utente_id = %s
```

- **Inserimento di un nuovo piatto:**
```
INSERT INTO piatti (nome, descrizione, validato, utente_id)
VALUES (%s, %s, %s, %s)
```

- **Associazione ingredienti a un piatto:**
```
INSERT INTO piatti_ingredienti (piatto_id, ingrediente_id, quantita)
VALUES (%s, %s, %s)
```

6.5 Query sul planner settimanale

File: `models/planner.py`
Principali operazioni:

- **Recupero del planner dell'utente:**
```
SELECT data, pasto, piatto_id, piatti.nome
FROM planner
JOIN piatti ON planner.piatto_id = piatti.id
WHERE planner.utente_id = %s
```

- **Aggiunta o aggiornamento di un pasto:**
```
SELECT id FROM planner WHERE utente_id = %s AND data = %s AND pasto = %s
```

  Se già presente:
```
UPDATE planner SET piatto_id = %s WHERE id = %s
```

  Altrimenti:
```
INSERT INTO planner (utente_id, data, pasto, piatto_id)
VALUES (%s, %s, %s, %s)
```

- **Rimozione di un piatto dal planner:**
```
DELETE FROM planner
WHERE utente_id = %s AND data = %s AND pasto = %s AND piatto_id = %s
```

6.6 View per le statistiche nutrizionali

Per calcolare i valori nutrizionali giornalieri e settimanali in modo efficiente, è stata creata una **VIEW** chiamata **`planner_nutritional_view`**.
Questa view aggrega le informazioni su ingredienti, piatti e planner in un'unica tabella virtuale.

```
CREATE VIEW planner_nutritional_view AS
SELECT
   planner.utente_id,
   planner.data,
   planner.pasto,
   planner.piatto_id,
   ingredienti.id AS ingrediente_id,
   (ingredienti.proteine * piatti_ingredienti.quantita / 100) AS proteine,
   (ingredienti.carboidrati * piatti_ingredienti.quantita / 100) AS carboidrati,
   (ingredienti.calorie * piatti_ingredienti.quantita / 100) AS calorie
FROM planner
JOIN piatti ON planner.piatto_id = piatti.id
JOIN piatti_ingredienti ON piatti.id = piatti_ingredienti.piatto_id
JOIN ingredienti ON piatti_ingredienti.ingrediente_id = ingredienti.id;
```

Query di utilizzo della view:

- **Statistiche per un giorno:**
  ```
  SELECT
    COALESCE(SUM(proteine), 0) AS tot_proteine,
    COALESCE(SUM(carboidrati), 0) AS tot_carboidrati,
    COALESCE(SUM(calorie), 0) AS tot_calorie
  FROM planner_nutritional_view
  WHERE utente_id = %s AND DATE(data) = %s
  ```

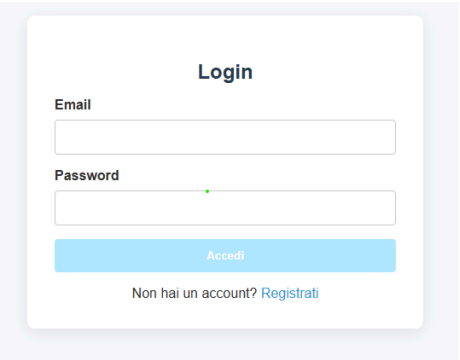
- **Statistiche per una settimana:**
  ```
  SELECT
    COALESCE(SUM(proteine), 0) AS tot_proteine,
    COALESCE(SUM(carboidrati), 0) AS tot_carboidrati,
    COALESCE(SUM(calorie), 0) AS tot_calorie
  FROM planner_nutritional_view
  WHERE utente_id = %s AND DATE(data) BETWEEN %s AND %s
  ```

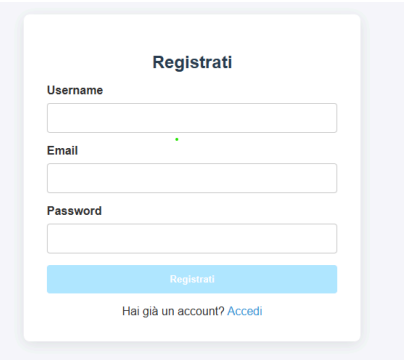
Nota su `COALESCE`:
La funzione `COALESCE` serve a **evitare valori NULL** quando la somma non restituisce righe(ad esempio se l'utente non ha piatti assegnati a quella data).
Se `SUM` restituisce `NULL`, `COALESCE` lo sostituisce con 0 per evitare errori nel calcolo o nei grafici

7. Testing

In questa sezione vengono riportati alcuni screenshot dell'applicazione funzionante.
Sotto ogni immagine è indicata la **route** Python responsabile della funzionalità, in modo da evidenziare l'integrazione tra **frontend** e **backend**.

7.1 Schermata di login e di registrazione utente

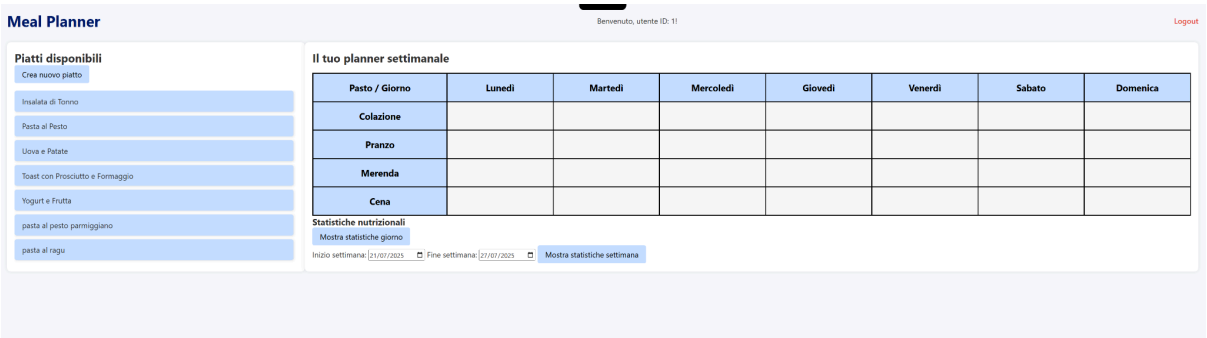




- **Route Python coinvolte:**

```
@app.route('/register', methods=['GET', 'POST'])
@app.route('/login', methods=['GET', 'POST'])
```

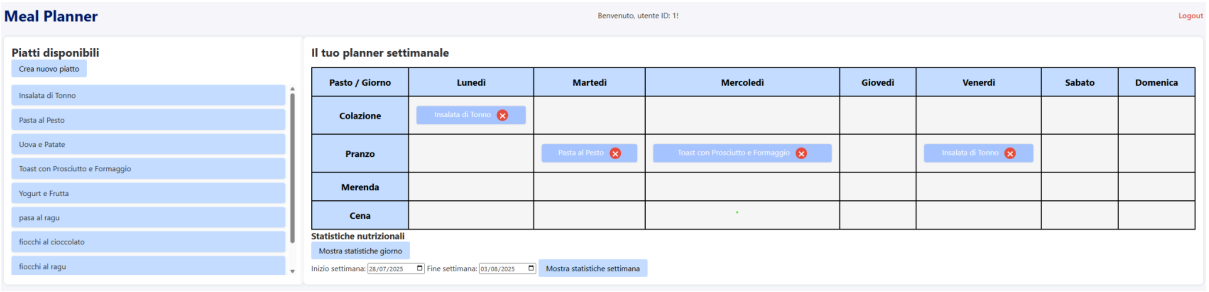
7.2 Schermata principale del meal planner



- **Route Python coinvolte:**

```
@app.route('/')
```

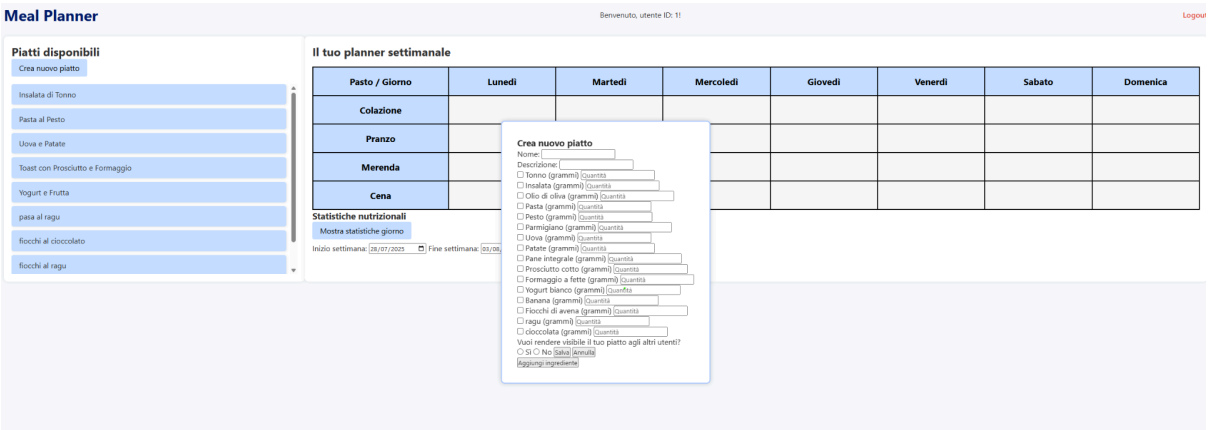
7.3 Inserimento di un piatto nel planner



- **Route Python coinvolte:**

```
@app.route('/add-to-planner', methods=['POST'])
@app.route('/remove-from-planner', methods=['POST'])
```

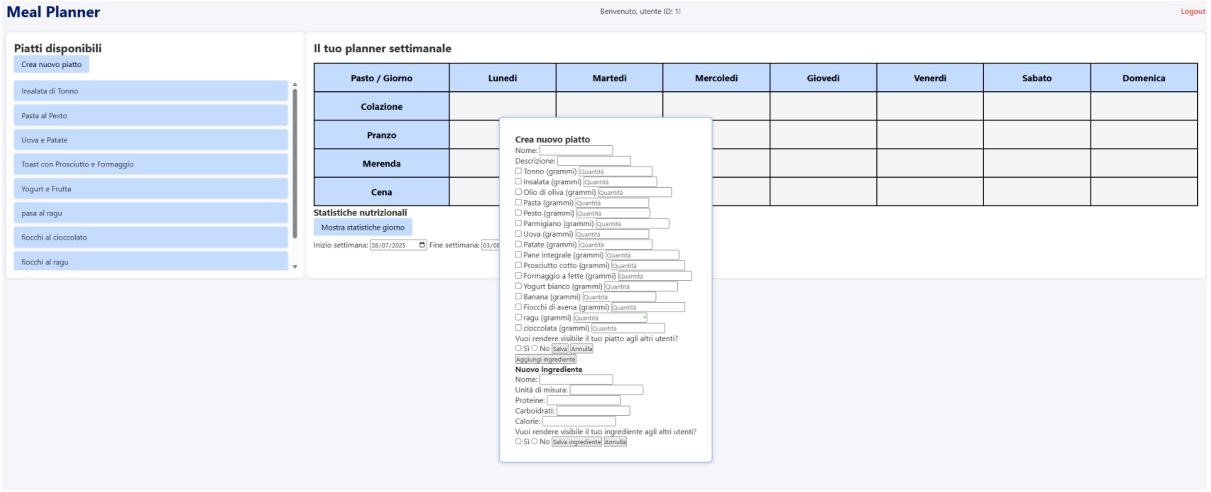
7.4 Creazione di un nuovo piatto



- **Route Python coinvolte:**

```
@app.route('/ingredienti-list')
@app.route('/crea-piatto', methods=['POST'])
```

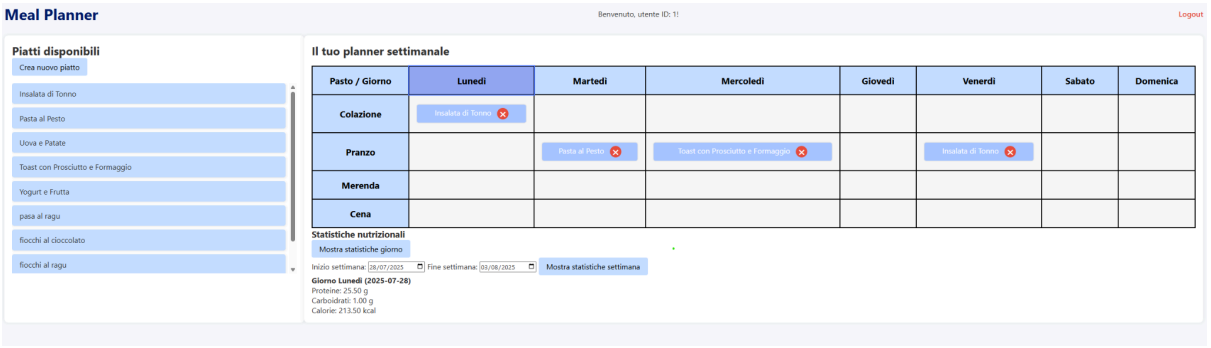
7.5 Creazione di un nuovo ingrediente



- **Route Python coinvolte:**

```
@app.route('/aggiungi-ingrediente', methods=['POST'])
```

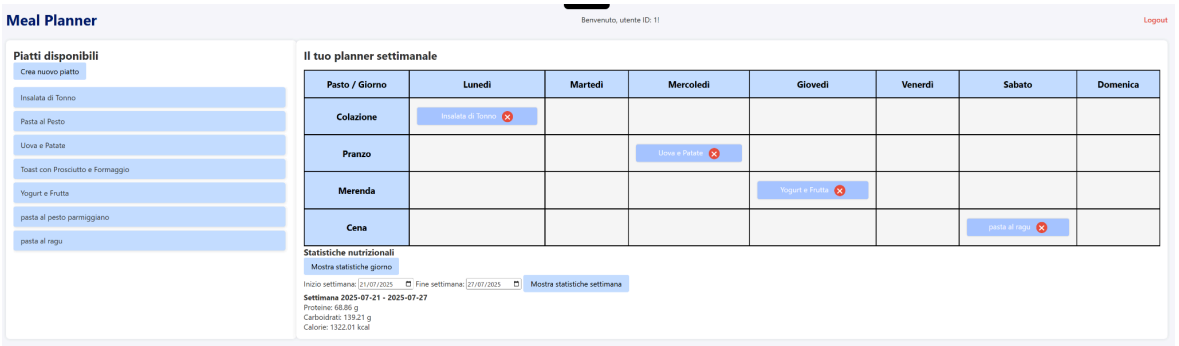
7.6 Visualizzazione delle statistiche giornaliere



- **Route Python coinvolte:**

```
@app.route('/stats-day', methods=['GET'])
```

7.7 Visualizzazione delle statistiche settimanali



- **Route Python coinvolte:**

```
@app.route('/stats-week', methods=['GET'])
```