



Document: FitCal_Sviluppo

Revision: 0.2

Progetto:

FitCal

Titolo del documento:

Documento di Sviluppo

Document Info:

Document name	D4-FitCal_Sviluppo	Document number	SV4
Description	Documento contenente gli user flow, resource ed api diagrams insieme alla documentazione della struttura del progetto e dello sviluppo dell'applicazione.		
Authors	Azemi Kevin Lollato Filippo Mattioli Michele		

INDICE

1. [Contenuti del documento](#)
2. [User Flow](#)
3. [Resource Diagram](#)
4. [Api Diagram](#)
 - 4.1. [API Diagrams User](#)
 - 4.2. [API Diagrams Meal](#)
 - 4.3. [API Diagrams Food](#)
5. [Sviluppo Applicazione](#)
 - 5.1. [Struttura progetto](#)
 - 5.2. [Dipendenze](#)
 - 5.3. [Modelli](#)
 - 5.3.1. [User](#)
 - 5.3.2. [Food](#)
 - 5.3.3. [Meal](#)
 - 5.4. [API](#)
 - 5.5. [API Documentation](#)
 - 5.6. [Frontend](#)
 - 5.7. [Testing](#)
6. [GitHub repository e Deployment](#)

Contenuti del documento

Il presente documento contiene tutte le informazioni relative lo sviluppo dell'applicazione FitCal.

Inizialmente, verrà presentato il diagramma di flusso utente, che descrive le azioni eseguibili all'interno dell'applicazione da parte di un utente.

Successivamente, sarà illustrato il diagramma delle risorse, ovvero la struttura dei dati all'interno del database, necessaria per garantire il corretto funzionamento del servizio.

In seguito, verranno descritti i diagrammi delle API e il processo di sviluppo dell'applicazione, elencando la struttura del progetto, le dipendenze, i modelli, API e la documentazione associata.

Successivamente, sarà presentato uno showcase del front-end basato sui mockup illustrati nel documento D1 e sulle relative funzionalità.

Il documento si concluderà con la fase di testing dell'applicazione, tramite apposito servizio, e una descrizione del deployment, comprensiva di un link a GitHub e al sito funzionante, per permettere la revisione senza dover disporre di un'istanza locale dell'applicazione.

User Flow

In questa sezione descriviamo il percorso che un utente esterno deve seguire per utilizzare una specifica funzionalità dell'applicazione FitCal. Attraverso il seguente diagramma di contesto, è possibile visualizzare come l'utente interagisce con il prodotto e ogni singola azione che deve completare, inclusi i possibili problemi che potrebbero presentarsi.

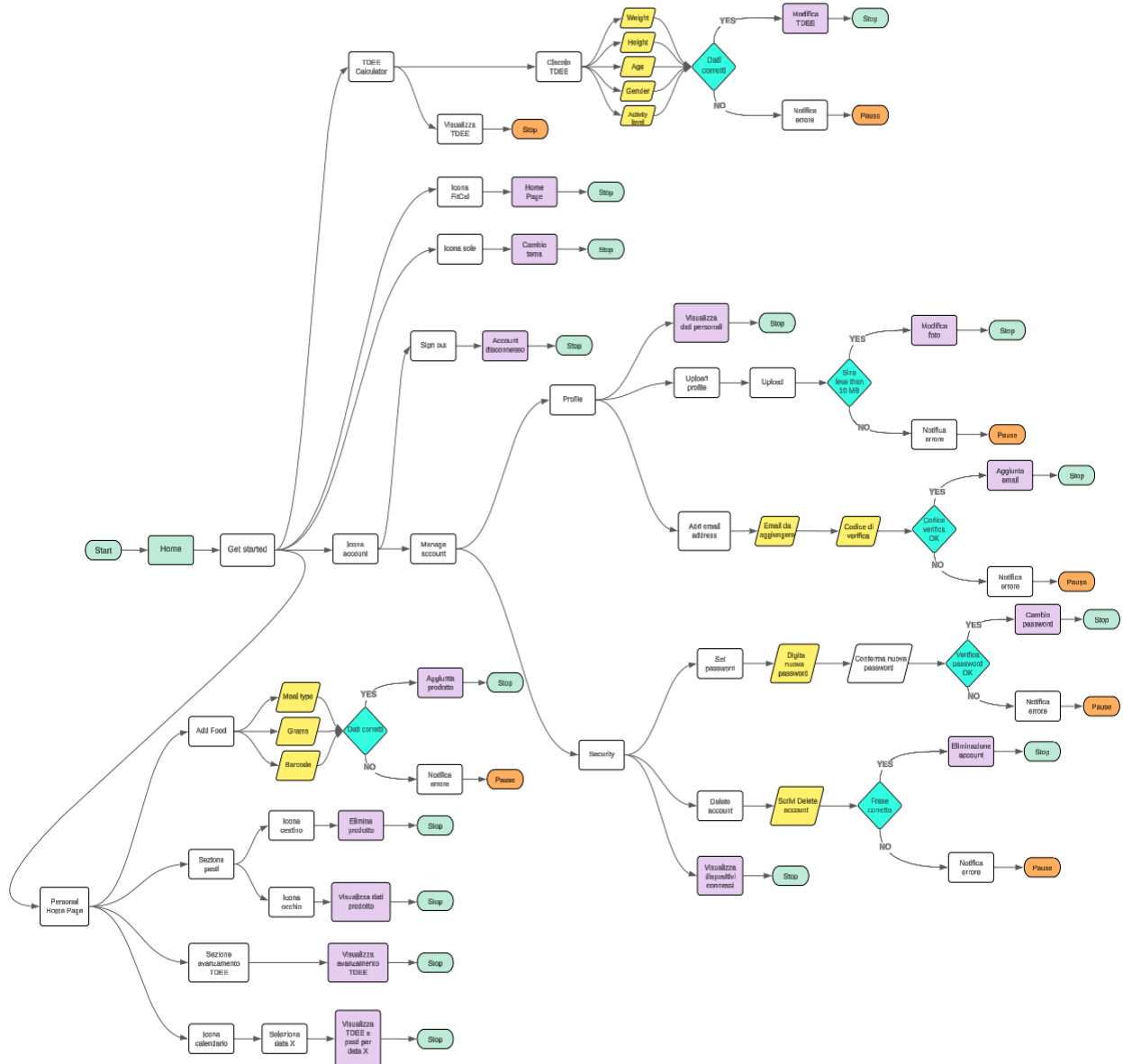


Fig. 1: User Flow.

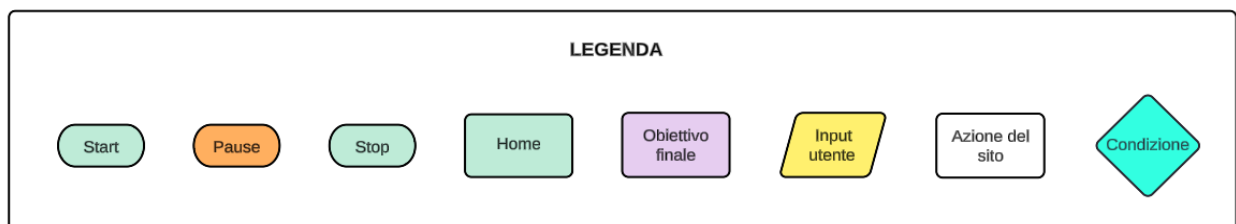


Fig. 2: Legenda.

Resource Diagram

Nella seguente sezione illustreremo il resource diagram dalla quale verranno in seguito estratti i modelli per l'applicazione.

Partendo dal class diagram presentato nel documento precedente sono state individuate tre risorse principali:

- User;
- Food;
- Meal.

A ciascuna risorsa saranno poi associate le rispettive richieste eseguibili per il corretto funzionamento dell'applicazione.

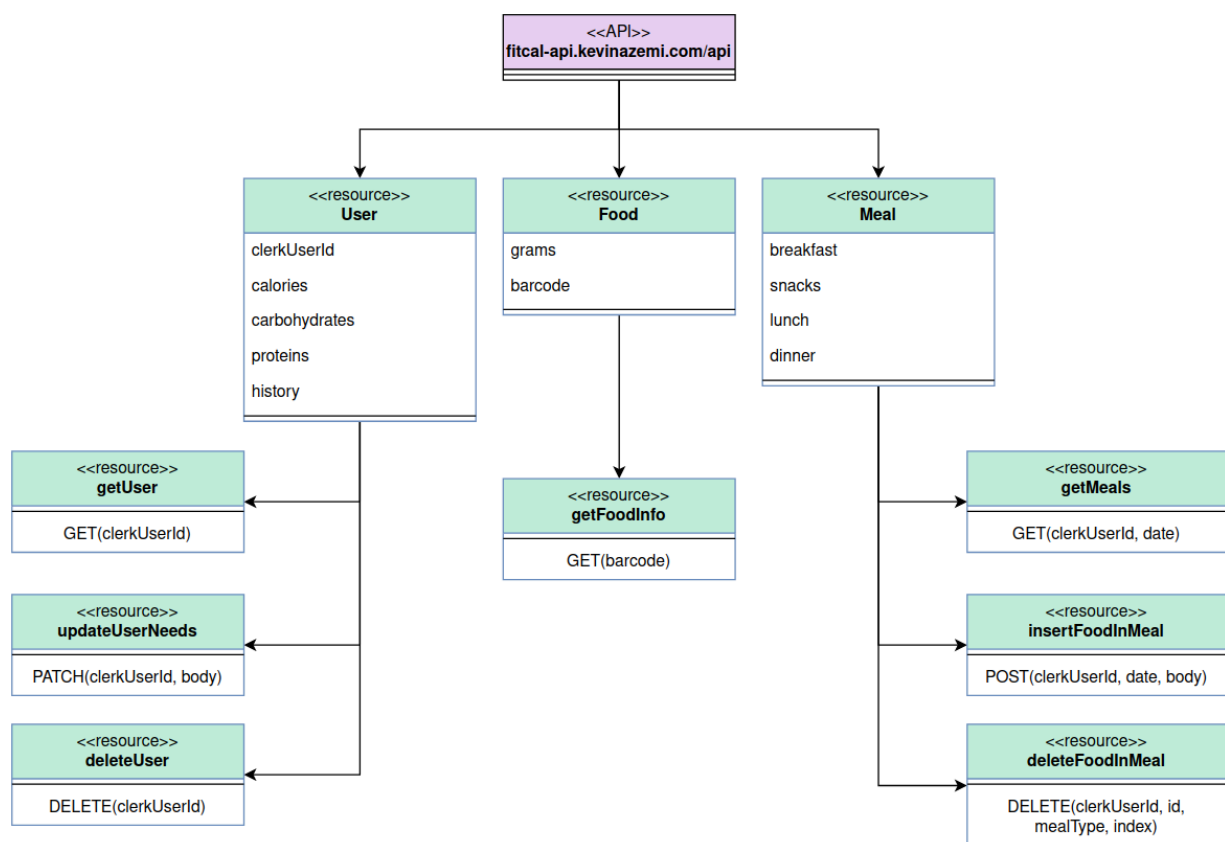


Fig. 3: Resource Diagram.

API Diagram

Nella seguente sezione verranno presentati gli API diagrams nella quale verranno elencate le API disponibili ed i dati che richiedono in ingresso e ritornano in uscita.

Verranno inoltre elencati i possibili errori riscontrabili nel caso di richieste sbagliate o dati non trovati.

API Diagrams User

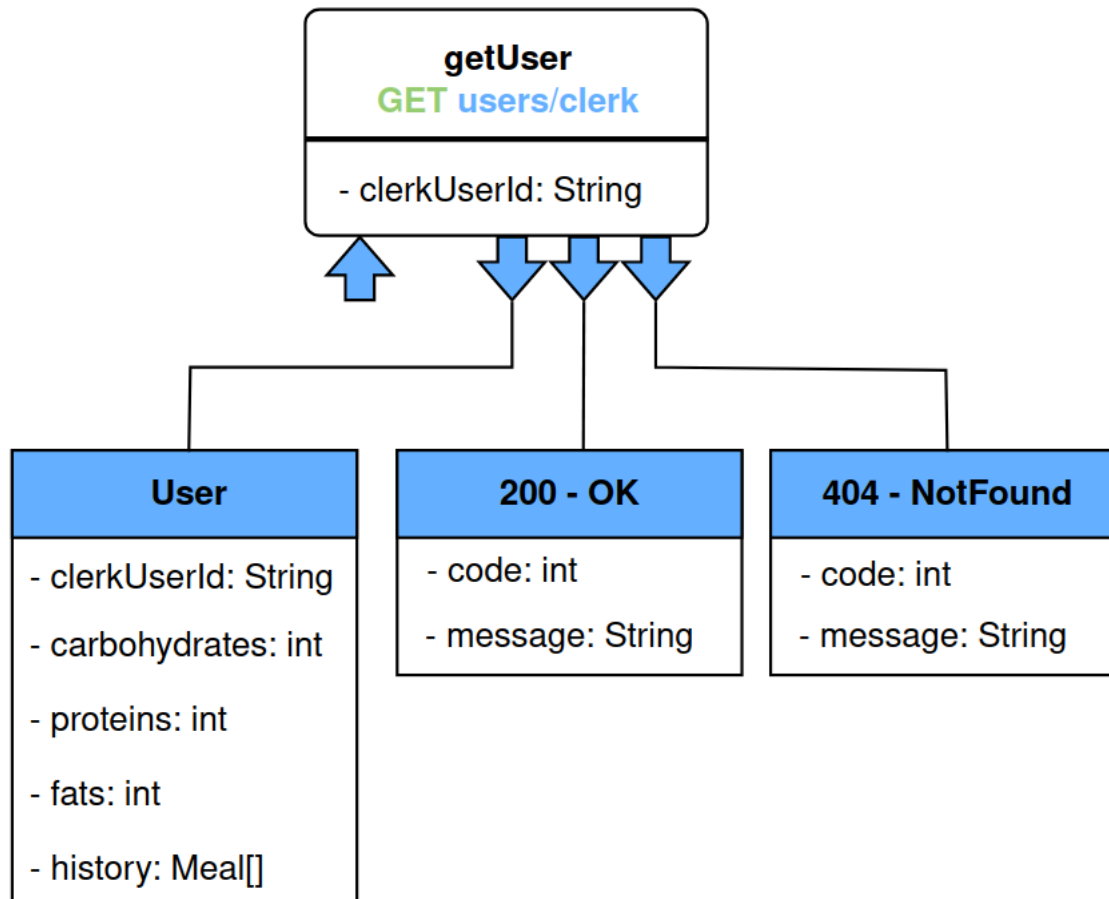


Fig. 4: getUser API Diagram.

La seguente api rappresenta la richiesta di un utente dato l'id e può ritornare l'utente in caso di successo oppure l'errore 404 in caso di utente non trovato.

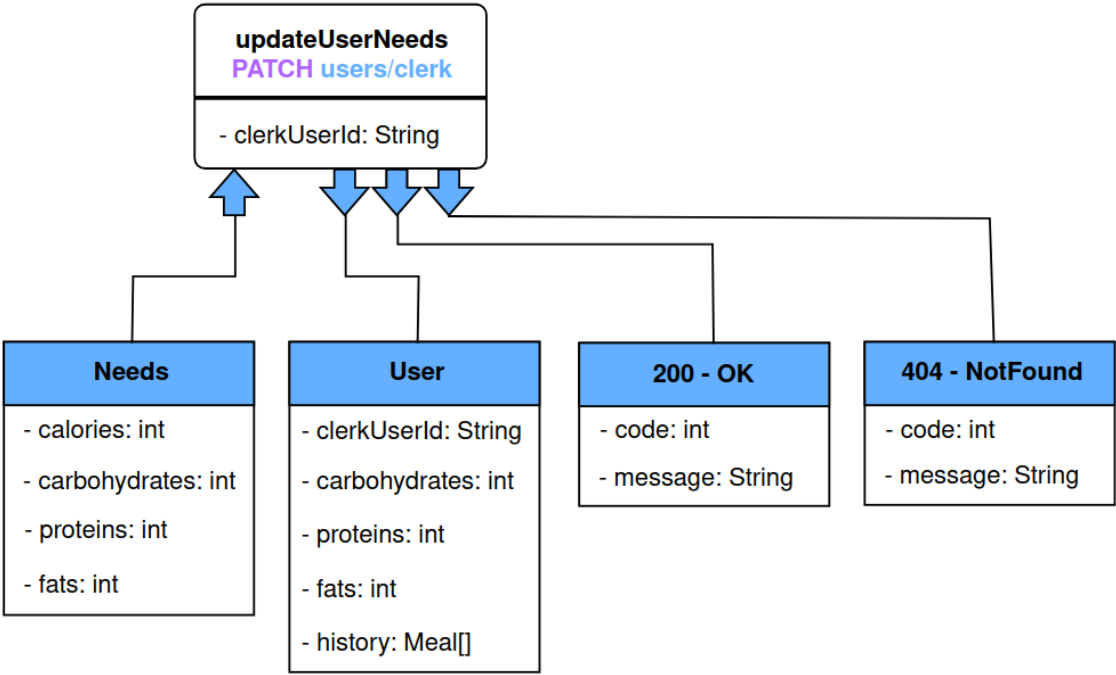


Fig. 5: updateUserNeeds API Diagram.

La seguente api rappresenta la richiesta di aggiornamento dei dati di fabbisogno di un utente dato l'id e può ritornare l'utente con i nuovi dati in caso di successo oppure l'errore 404 in caso di utente non trovato.

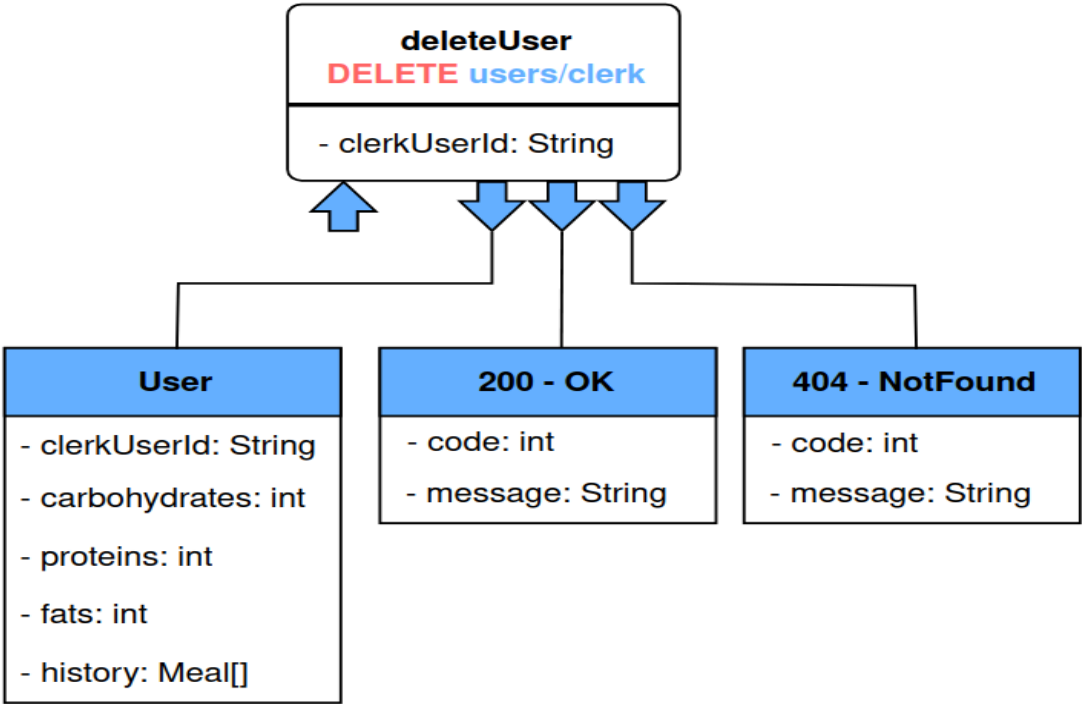


Fig. 6: deleteUser API Diagram.

La seguente api rappresenta la richiesta di eliminazione di un utente dato l'id e può ritornare l'utente appena rimosso dal database in caso di successo oppure l'errore 404 in caso di utente non trovato.

API Diagrams Meal

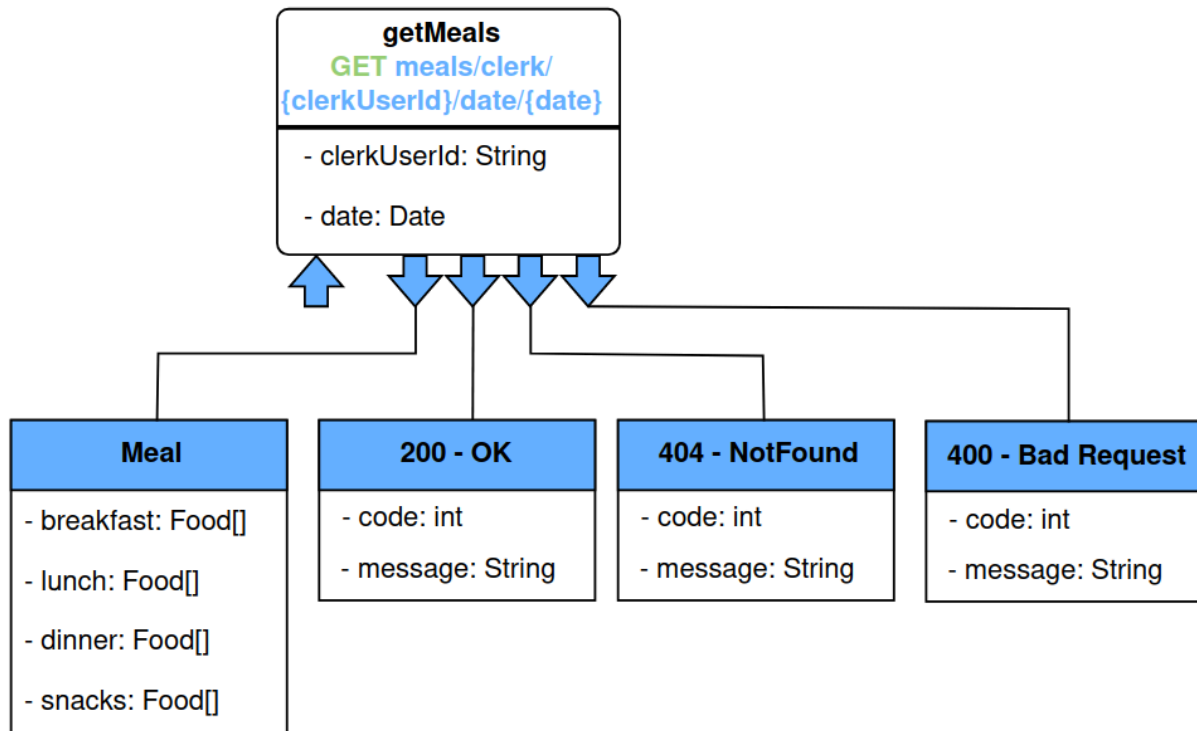


Fig. 7: getMeals API Diagram.

La seguente api rappresenta la richiesta di un pasto dato l'id di un utente ed una specifica data e può ritornare:

- i pasti dell'utente in caso di successo;
- l'errore 404 in caso di utente non trovato;
- l'errore 400 in caso di data mal formattata.

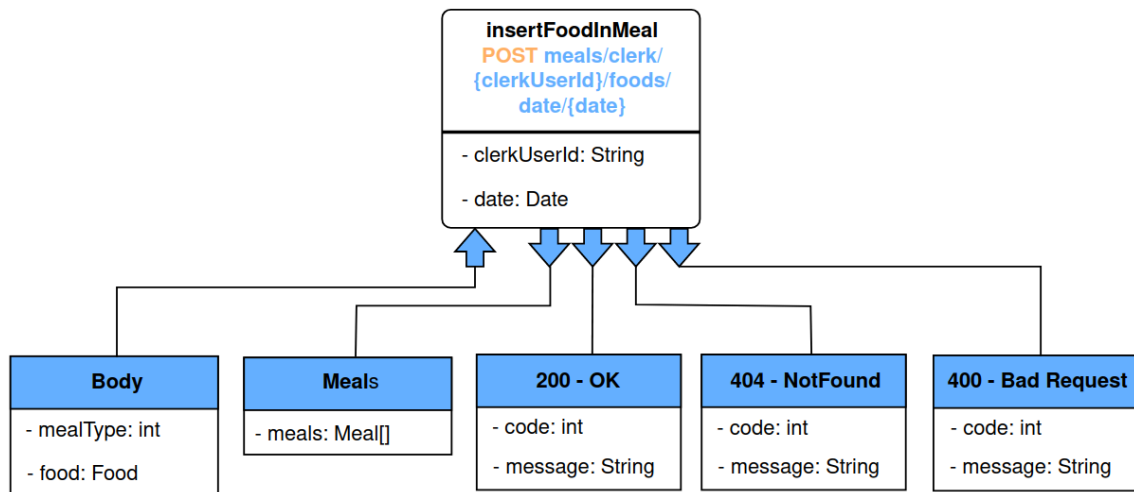


Fig. 7: insertFoodInMeal API Diagram.

La seguente api rappresenta la richiesta di inserimento di un cibo in un pasto dato l'id di un utente ed una specifica data e può ritornare:

- i nuovi pasti dell'utente in caso di successo;
- l'errore 404 in caso di utente non trovato;
- l'errore 400 in caso di data mal formattata.

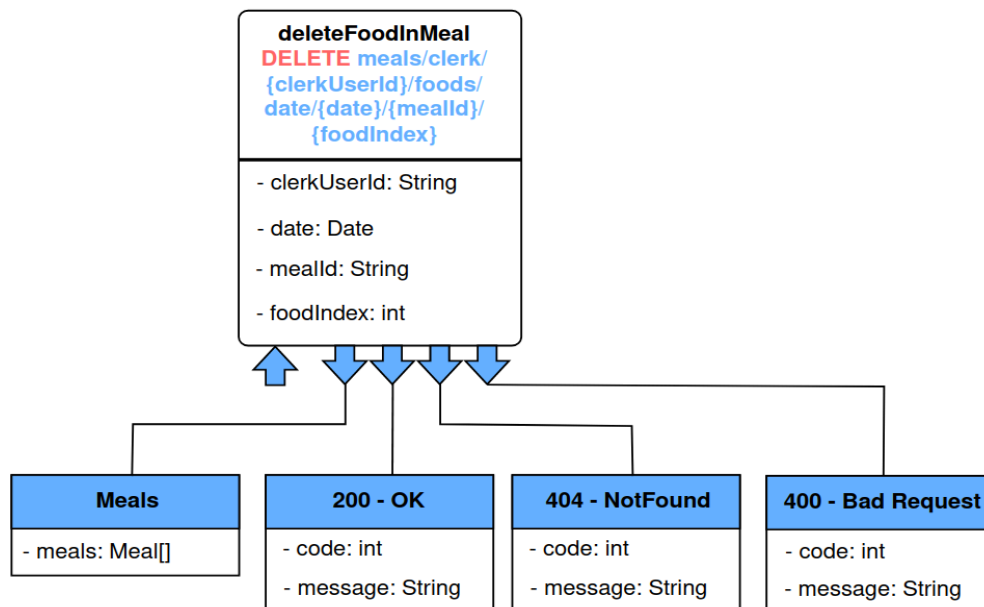


Fig. 8: insertFoodInMeal API Diagram.

La seguente api rappresenta la richiesta di eliminazione di un cibo da un pasto da dato l'id di un utente, una specifica data, il tipo di pasto e l'indice del cibo da eliminare e può ritornare:

- i nuovi pasti dell'utente in caso di successo;
- l'errore 404 in caso di utente non trovato;
- l'errore 400 in caso di data mal formattata, mealId o indice incorretto.

API Diagrams Food

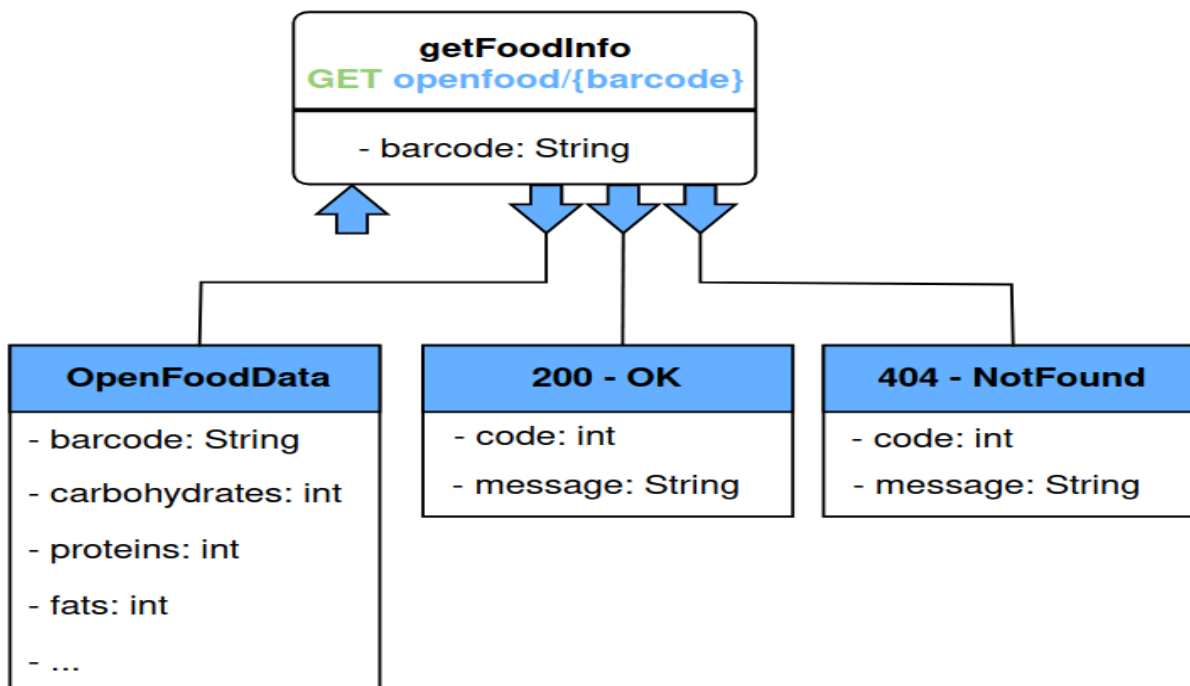


Fig. 9: insertFoodInMeal API Diagram.

La seguente api rappresenta la richiesta di informazioni di un specifico cibo dato il barcode e può ritornare:

- i dati del cibo presi direttamente dall'API OpenFoodFacts;
- l'errore 404 in caso di cibo non trovato.

Sviluppo Applicazione

In seguito verranno riportati gli step presi per lo sviluppo dell'applicazione, presentando in ordine la struttura del progetto, le dipendenze assunte, i modelli utilizzati all'interno del database, le API fornite con relativa documentazione, la descrizione del front-end e relativo testing dell'applicazione.

Struttura progetto

Nella seguente sezione sarà presentata la struttura del progetto e la suddivisione logica fra back-end e front-end.

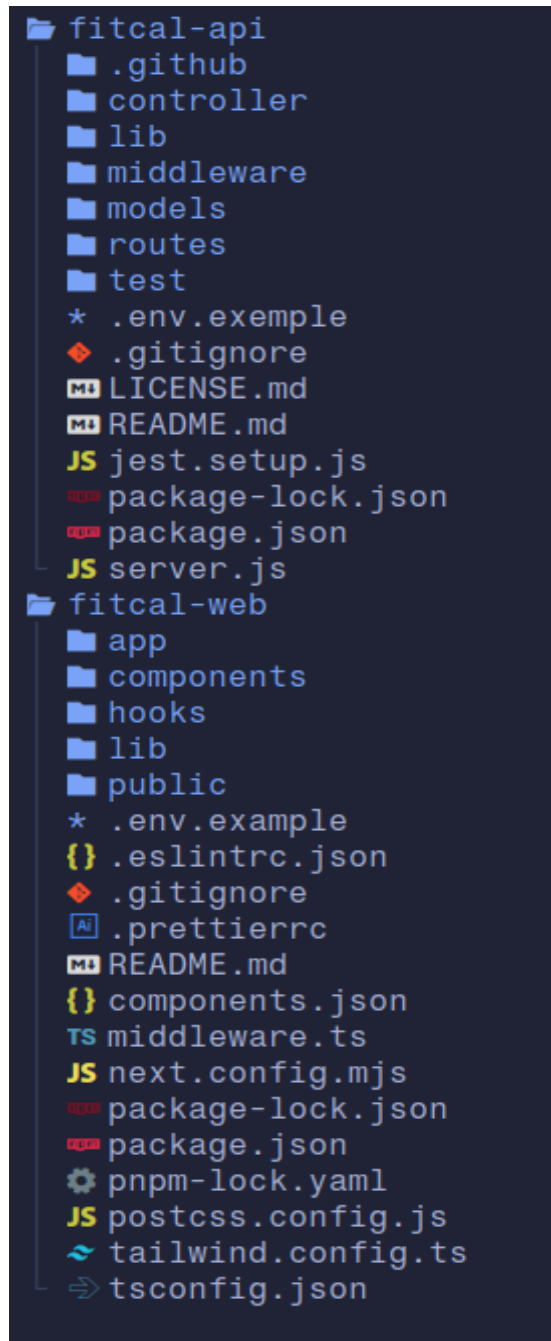


Fig. 10: Struttura Repository.

La struttura del progetto è suddivisa in due cartelle principali: *fitcal-api* e *fitcal-web*.

La cartella *fitcal-api* contiene il necessario per la gestione delle API dell'applicazione:

- controller: Implementa la logica delle API;
- middleware: Gestione richieste;
- models: Modelli delle risorse per interagire con il database;
- routes: Definizione delle rotte delle API;
- tests: Contiene i test per verificare il corretto funzionamento delle API.

La cartella *fitcal-web* invece contiene il lato front-end dell'applicazione:

- app: Pagine principali;
- components: Componenti che vanno a formare le singole pagine dell'applicazione;
- hooks: Custom hooks per la gestione dello stato ed effetti;
- public: risorse generali come immagini.

La seguente struttura è stata progettata per essere flessibile a future aggiunte di funzionalità e per facilitare le modifiche in caso di eventuali problemi.

Dipendenze

Saranno ora elencate le dipendenze esterne usate dall'applicazione, reperibili dal file *package.json* all'interno della cartella *fitcal-api*:

- clerk (4.13.7): Gestione utenti ed autenticazione;
- body-parser (1.20.2): Estrazione informazioni da una richiesta HTTP;
- cors (2.8.5): Accesso a risorse esterne al proprio dominio;
- dotenv (16.4.1): Gestione variabili d'ambiente del progetto;
- express (4.19.2): Backend framework;
- express-async-handler (1.2.0): Gestore chiamate asincrone per Express;
- install (0.13.0): Caricamento moduli;
- ioredis (5.4.1): Redis client;
- mongoose (8.1.1): Gestione dati con database;
- nodemon (3.0.3): Ripartire in automatico un nodo per facilitare lo sviluppo;
- npm (10.8.1): Package manager;
- redis (4.6.14): Database caching;
- svix (1.16.0): Gestione webhooks;
- jest (29.7.0): Test suite;
- nock (14.0.0-beta.7): Test suite;
- supertest (7.0.0): Test suite.

Modelli

Nella seguente sezione verranno elencati i modelli usati nell'applicazione estratti dal diagramma delle risorse.

User

```
const userSchema = mongoose.Schema(
  {
    clerkUserId: {
      type: String,
      required: true,
      unique: true
    },
    needs: {
      calories: {
        type: Number
      },
      carbohydrates: {
        type: Number
      },
      proteins: {
        type: Number
      },
      fats: {
        type: Number
      }
    },
    history: {
      type: [Meal.schema],
      default: []
    },
  },
  {
    timestamps: true
  }
)
```

Fig. 11: User Model.

Il seguente modello rappresenta i dati di un Utente della nostra applicazione, ed è denominato User.

I dati principali di questo modello includono:

- l'identificativo univoco dell'utente, denominato *clerkUserId*, usato per associare i dati nutrizionali all'utente corretto e ricavare dati aggiuntivi;
- gli attributi nutrizionali del fabbisogno giornaliero per comodità sono stati incapsulati in un oggetto per facilitare il trasferimento dati e contengono le calorie, carboidrati, proteine e grassi.

L'attributo *history* è un array che memorizza i record dei pasti consumati dall'utente, utilizzando lo schema *Meal*.

Food

```
const foodSchema = mongoose.Schema({
  {
    grams: {
      type: Number,
      required: [true, "Please enter the grams"]
    },
    barcode : {
      type: String,
      required: [true, "Please enter a barcode"]
    }
  },
  {
    timestamps: false
  }
})
```

Fig. 12: Food Model.

Il seguente modello rappresenta un prodotto alimentare all'interno della nostra applicazione, ed è denominato Food.

I dati principali di questo modello includono:

- la quantità di cibo consumata in grammi;
- il barcode del prodotto e permette la richiesta tramite API esterna delle informazioni.

Meal

```
const mealSchema = mongoose.Schema({
  {
    breakfast: {
      type: [Food.schema],
      required: false
    },
    lunch: {
      type: [Food.schema],
      required: false
    },
    dinner : {
      type: [Food.schema],
      required: false
    },
    snacks: {
      type: [Food.schema],
      required: false
    }
  },
  {
    timestamps: true
  }
})
```

Fig. 13: Meal Model.

Il seguente modello rappresenta i pasti consumati da un utente all'interno della nostra applicazione, ed è denominato Meal.

I dati principali inclusi in questo modello sono array di prodotti alimentari per ciascun pasto della giornata:

- Breakfast;
- Snacks;
- Lunch;
- Dinner.

API

Vengono di seguito descritte le API utilizzate per lo scambio di informazioni tra applicazione frontend, database di cache Redis e database MongoDB.

API modello User

GET getUser

- **Descrizione:** Questa API viene utilizzata per prendere tutte le varie informazioni dell'utente attualmente in sessione nel sito. Il dato se presente viene preso direttamente dal nostro database di cache Redis con key `user_{clerkUserId}`, altrimenti se non presente viene preso dal database mongoDB, e successivamente salvato in cache;
- **tipologia:** Si tratta di una richiesta di tipo **GET**;
- **parametri di Ingresso:** Viene fornito in input, direttamente nell'url della richiesta, l'userId di clerk univoco identificativo dell'utente;
- **esempio url richiesta:**
`https://fitcal-api.kevinazemi.com/api/users/clerk/{clerkUserId}`;
- **output:** In output invece, viene fornito lo stato della risposta (es. 200), i vari dati in formato JSON, oppure l'eventuale errore.

PATCH updateUserNeeds

- **Descrizione:** Questa API viene utilizzata per aggiornare il fabbisogno calorico e di macronutrienti dell'utente. Il dato viene aggiornato anche nel database di cache Redis;
- **tipologia:** Si tratta di una richiesta di tipo **PATCH**;
- **parametri di Ingresso:** Viene fornito in input, direttamente nell'url della richiesta, l'userId di clerk univoco identificativo dell'utente;
Come secondo input invece, vengono forniti nel body in formato JSON i nuovi dati per l'utente:

```
{
  "needs": {
    "calories": 1678,
    "carbohydrates": 189,
    "proteins": 126,
    "fats": 47
  }
}
```

Fig. 14: Needs parametro d'ingresso.

- **esempio url richiesta:**
`https://fitcal-api.kevinazemi.com/api/users/clerk/{clerkUserId}`;

- **output:** In output invece, viene fornito lo stato della risposta (es. 200), i vari dati aggiornati dell'utente in formato JSON, oppure l'eventuale errore.

DELETE deleteUser

- **Descrizione:** Questa API viene utilizzata per eliminare un utente dal database. L'utente viene eliminato anche dal database di cache Redis;
- **tipologia:** Si tratta di una richiesta di tipo **DELETE**;
- **parametri di Ingresso:** Viene fornito in input, direttamente nell'url della richiesta, l'userId di clerk univoco identificativo dell'utente;
- **esempio url richiesta:**
`https://fitcal-api.kevinazemi.com/api/users/clerk/{{clerkUserId}};`
- **output:** In output invece, viene fornito lo stato della risposta (es. 200), i vari dati dell'utente appena eliminato in formato JSON, oppure l'eventuale errore.

API modello Meals

GET getMeals

- **Descrizione:** Questa API viene utilizzata per prendere tutti i pasti di una determinata giornata dell'utente;
- **tipologia:** Si tratta di una richiesta di tipo **GET**;
- **parametri di Ingresso:** Viene fornito in input, direttamente nell'url della richiesta, l'userId di clerk univoco identificativo dell'utente. Poi viene fornito sempre nell'url della richiesta anche la data nel formato YYYY-MM-DD;
- **esempio url richiesta:**
`https://fitcal-api.kevinazemi.com/api/meals/clerk/{{clerkUserId}}/date/{{YYYY-MM-DD}};`
- **output:** In output invece, viene fornito lo stato della risposta (es. 200), i vari dati in formato JSON, oppure l'eventuale errore.

POST insertFoodInMeal

- **Descrizione:** Questa API viene utilizzata per aggiungere un cibo all'interno di un pasto. Il dato viene aggiornato anche nel database di cache Redis;
- **tipologia:** Si tratta di una richiesta di tipo **POST**;
- **parametri di Ingresso:** Viene fornito in input, direttamente nell'url della richiesta, l'userId di clerk univoco identificativo dell'utente. Poi viene fornito sempre nell'url della richiesta anche la data nel formato YYYY-MM-DD;
Nel body della richiesta, vengono forniti in formato JSON i dati del cibo da aggiungere al pasto:

```
{
  "mealType": "breakfast",
  "food": {
    "grams": 121,
    "barcode": "80135876"
  }
}
```

Fig. 15: Meal parametro d'ingresso.

- **esempio url richiesta:**
<https://fitcal-api.kevinazemi.com/api/meals/clerk/{{clerkUserId}}/foods/date/{{YYYYY-MM-DD}}>;
- **output:** In output invece, viene fornito lo stato della risposta (es. 200), i vari pasti dell'utente in formato JSON, oppure l'eventuale errore.

DELETE deleteFoodInMeal

- **Descrizione:** Questa API viene utilizzata per eliminare un cibo all'interno di un determinato pasto. Il dato viene eliminato anche dal database di cache Redis.
- **tipologia:** Si tratta di una richiesta di tipo **DELETE**;
- **parametri di Ingresso:** Viene fornito in input, direttamente nell'url della richiesta, l'userId di clerk univoco identificativo dell'utente, la data in formato YYYY-MM-DD, l'id identificativo del pasto, il tipo di pasto (es. breakfast) e l'id del cibo;
- **esempio url richiesta:**
<https://fitcal-api.kevinazemi.com/api/meals/clerk/{{clerkUserId}}/foods/date/{{YYYYY-MM-DD}}/{{mealId}}/{{mealType}}/{{foodIndex}}>;
- **output:** In output invece, viene fornito lo stato della risposta (es. 200), i vari pasti dell'utente in formato JSON, oppure l'eventuale errore.

API modello Food

GET getMeals

- **Descrizione:** Questa API viene utilizzata per prendere le informazioni di un cibo. Era possibile non implementare questo endpoint, ma e' stato volutamente implementato per poter salvare e prendere questi dati anche direttamente dal nostro sistema di cache Redis, per velocizzarne la risposta e per non effettuare troppe chiamate alle api pubbliche di Open Food Data. Se il barcode con key `food_{barcode}` non viene trovato nel database di cache Redis, allora la richiesta viene inoltrata alle api pubbliche di Open Food, e successivamente se la richiesta va a buon fine, i dati ottenuti vengono salvati nella nostra cache, altrimenti se la key viene trovata nella cache, il risultato della risposta viene fornito direttamente da lì;
- **tipologia:** Si tratta di una richiesta di tipo **GET**;
- **parametri di Ingresso:** Viene fornito in input, direttamente nell'url della richiesta, il barcode del cibo;
- **esempio url richiesta:**
`https://fitcal-api.kevinazemi.com/api/openfood/{{barcode}};`
- **output:** In output invece, viene fornito lo stato della risposta (es. 200), i vari dati in formato JSON, oppure l'eventuale errore.

API Documentation - Postman

Le API sviluppate per l'applicazione e descritte nel dettaglio precedentemente sono state documentate utilizzando Postman. Postman Documentation permette di avere una panoramica di tutte le API e le loro informazioni. Il link per poter visitare la documentazione delle API è il seguente:

<https://fitcal-docs.kevinazemi.com/>

API Utente:

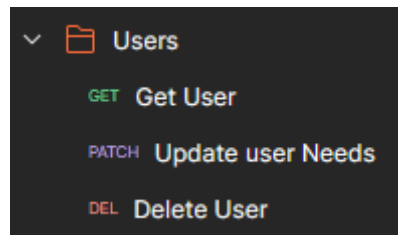


Fig. 16: Schema generale User API.

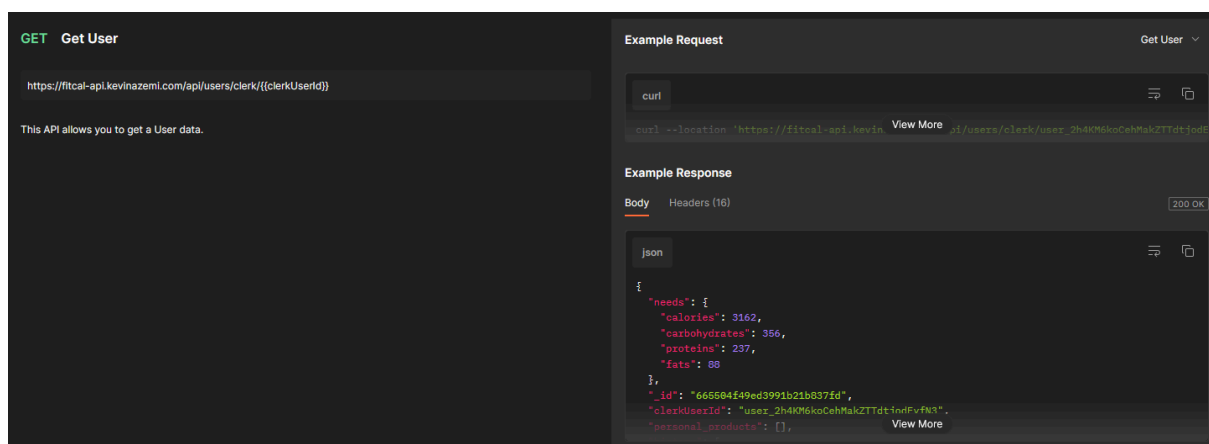


Fig. 17: GET Get User API.

PATCH

Update user Needs

<https://fitcal-api.kevinazemi.com/api/users/clerk/{{clerkUserId}}>

This API allows you to update user Needs.

For example, in our frontend, this api is called when the user saves the new TDEE data he calculated.

Body

raw (json)

json

```
{
  "needs": {
    "calories": 1678,
    "carbohydrates": 189,
    "proteins": 126,
    "fats": 47
  }
}
```

Example Request

Update User Needs

curl

```
curl --location --request PATCH 'https://fitcal-api.kevinazemi.com/api/users/clerk/user_2hZGMW' --data '{
  "needs": {
    "calories": 1678,
    "carbohydrates": 189,
    "proteins": 126,
    "fats": 47
  }
}'
```

View More

Example Response

Body

Headers (16)

200 OK

json

```
{
  "needs": {
    "calories": 1678,
    "carbohydrates": 189,
    "proteins": 126,
    "fats": 47
  },
  "_id": "666429765c0988bffb1b41ee",
  "clerkUserId": "user_2hZGMWp7w0N4ysHFAVne411u0z",
  "personal_products": []
}
```

View More

Fig. 18: PATCH Update User Needs.

DELETE

Delete User

<https://fitcal-api.kevinazemi.com/api/users/clerk/{{clerkUserId}}>

This API allows you to delete a User by providing his clerk user id.

Note that in our frontend, this API won't be ever used since the user deletion is invoked by the clerk user.deleted webhook event.

Example Request

Delete User

curl

```
curl --location --request DELETE 'https://fitcal-api.kevinazemi.com/api/users/clerk/user_2hZGMW'
```

View More

Example Response

Body

Headers (16)

200 OK

json

```
{
  "needs": {
    "calories": 1678,
    "carbohydrates": 189,
    "proteins": 126,
    "fats": 47
  },
  "_id": "666429765c0988bffb1b41ee",
  "clerkUserId": "user_2hZGMWp7w0N4ysHFAVne411u0z",
  "personal_products": []
}
```

View More

Fig. 19: DELETE Delete User.

API Meal:

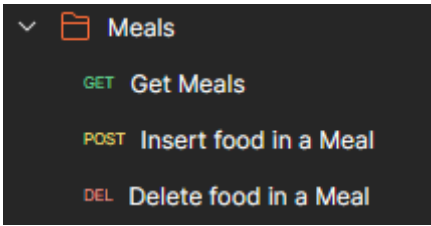


Fig. 20: Schema generale Meals API.

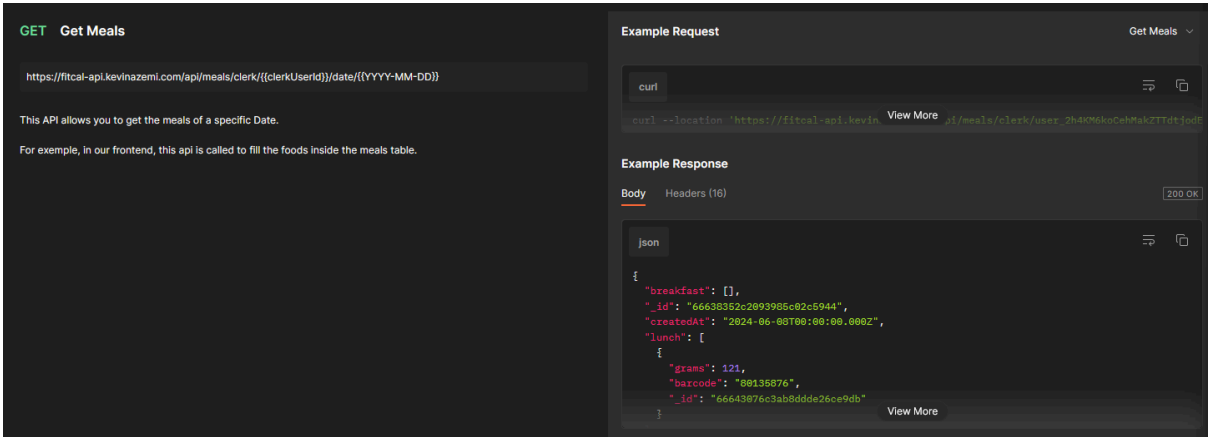


Fig. 21: GET Get Meals.

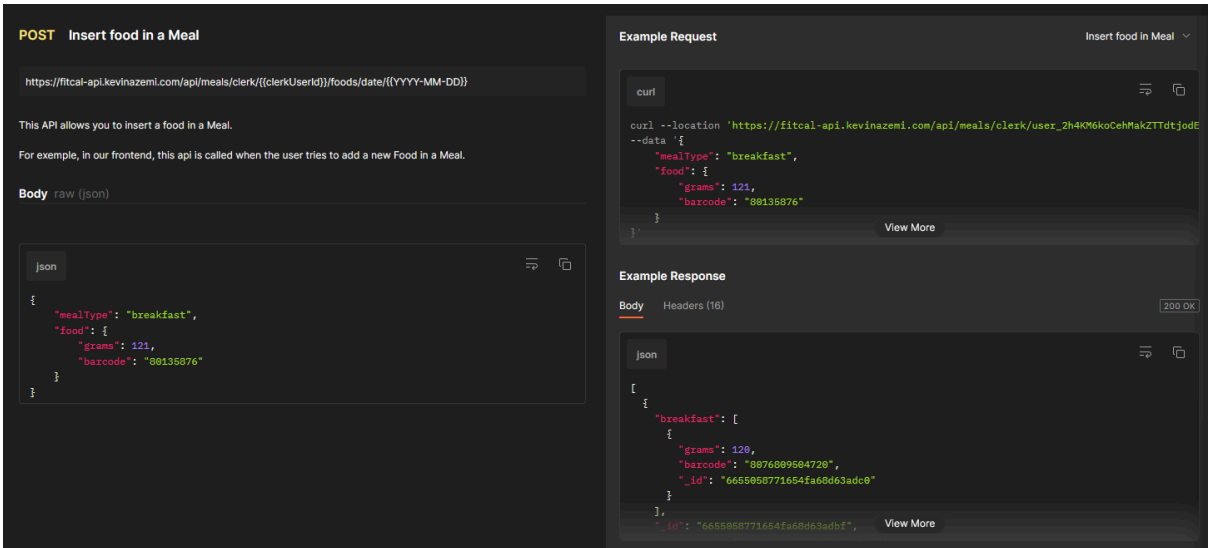


Fig. 22: POST Insert food in Meal.

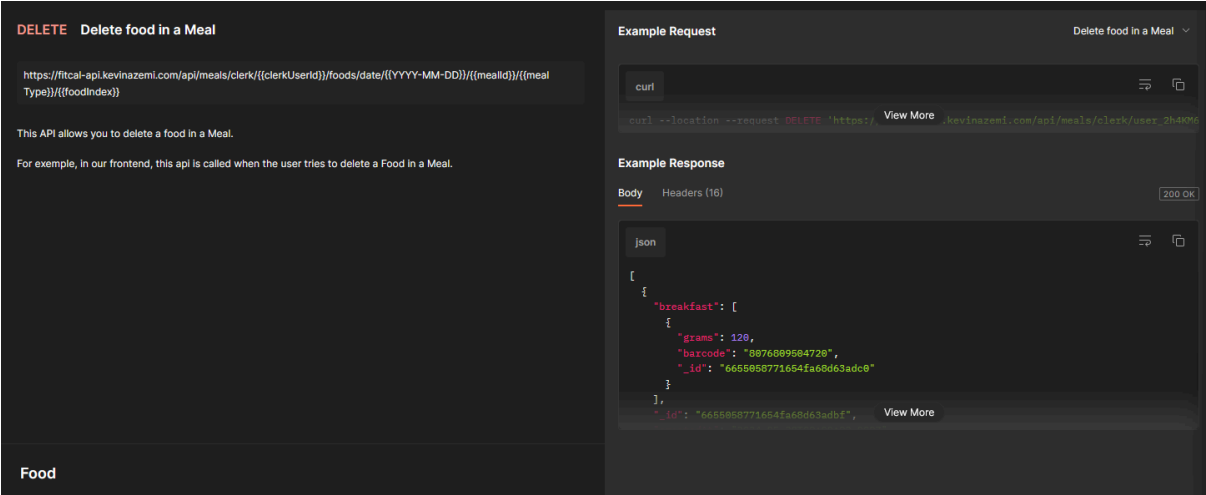


Fig. 23: DELETE Delete food in a Meal.

API Food:

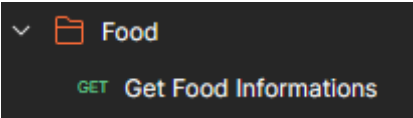


Fig. 24: Schema generale Food API.

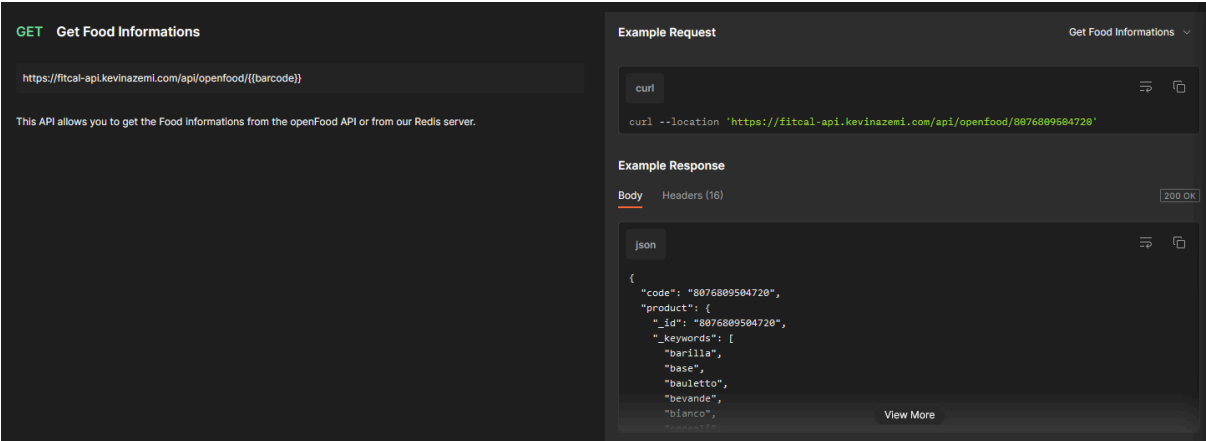


Fig. 25: GET Get Food Informations.

Frontend

In questo capitolo andremo a descrivere il front-end dell'applicazione FitCal; verra' utilizzato un linguaggio naturale, insieme a foto del sito web. L'obiettivo e' mostrare l'implementazione dei concetti e dei requisiti funzionali, discussi nei precedenti documenti.

1. Home page

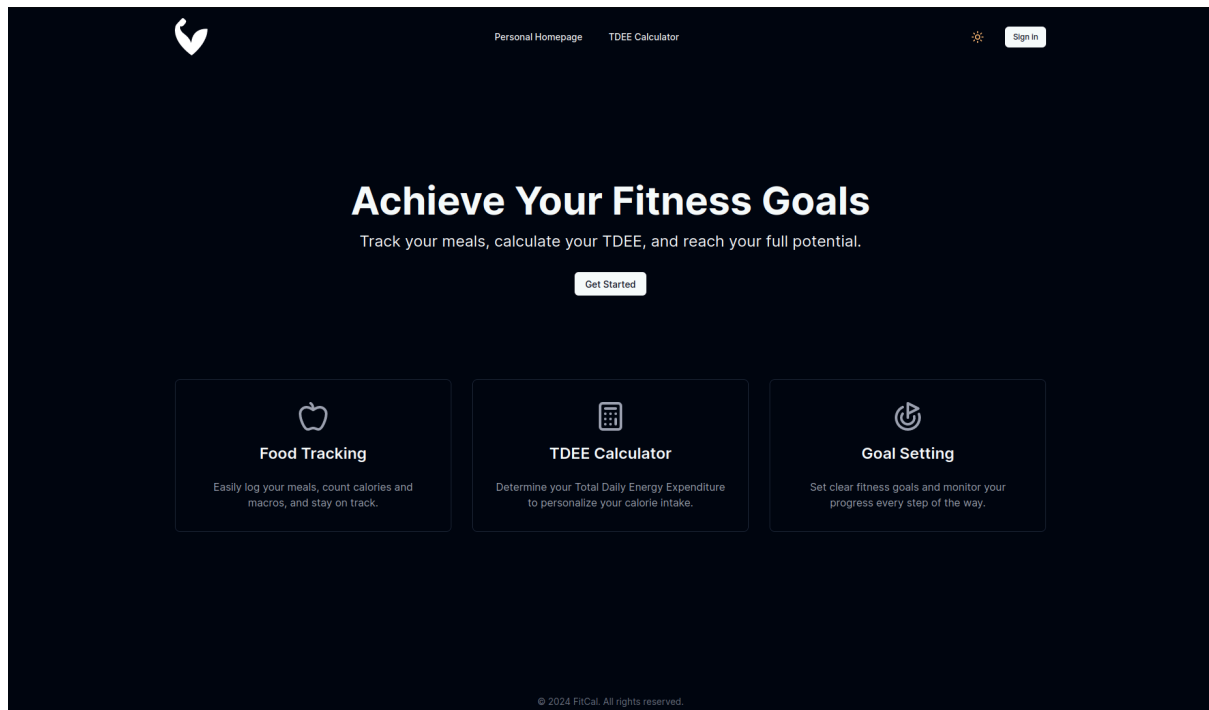


Fig. 26: GET Get Food Informations.

La Homepage rappresenta il punto di partenza del sito web. Essa mostra i principali obiettivi dell'applicazione, tramite uno stile sobrio e pulito, ricorrente in ogni pagina. L'utente che si trova sulla home page, come mostrato nel D2 tramite l'analisi del contesto dell'applicazione, può essere di due tipi:

- non autenticato;
- autenticato.

Entrambi rappresentano degli attori esterni, tuttavia distinguiamo le due situazioni:

- per quanto riguarda il primo, le icone centrali in alto, "Personal Homepage" e "TDEE Calculator", sono entrambe disattivate. Cliccando su "Get Started" egli avrà accesso alla pagina per il login, ed eventualmente a quella per la registrazione. Allo stesso modo potrà autenticarsi cliccando l'icona in alto a sinistra "Sign in";
- per l'utente autenticato, invece, ogni icona è attiva, egli può quindi anche andare sulla "Personal Homepage" o sulla pagina "TDEE Calculator".

2. Possibilita' di cambiare tema

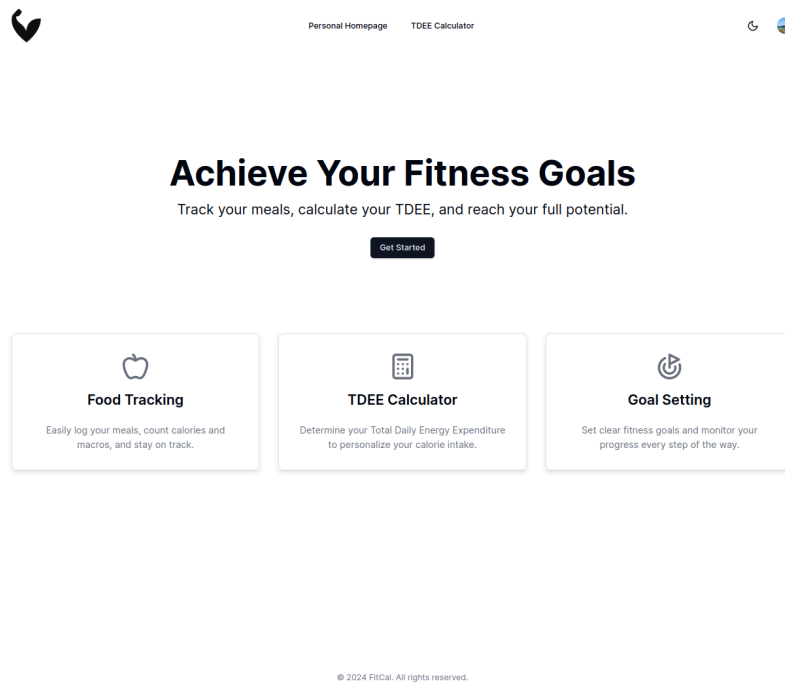


Fig. 27: GET Get Food Informations.

Tramite l'icona che reca il simbolo, del sole se il tema attuale e' lo scuro, della luna altrimenti, l'utente, che sia autenticato o meno, potra' cambiare il tema dell'applicazione web: scegliendo tra:

- Light;
- Dark;
- System.

Nel corso della descrizione del frontend faremo uso di pagine impostate con il tema Dark, tuttavia è bene tenere presente che il tema puo' essere cambiato da ogni pagina dell'applicazione web, come sara' verificabile tramite il link in fondo al documento.

3. Pagina autenticazione

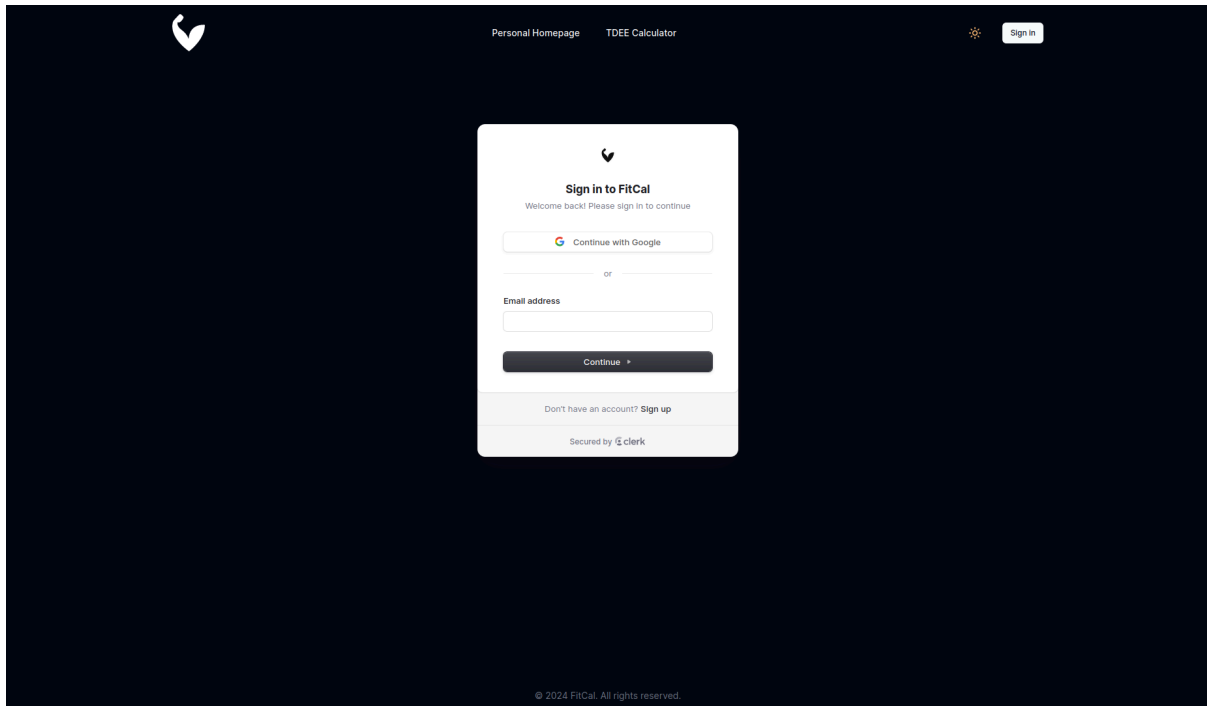


Fig. 28: GET Get Food Informations.

La pagina in questione permette all'utente anonimo di effettuare il login al sito. Vengono mostrate le due modalita' disponibili:

- login tramite credenziali FitCal, implementata tramite il sistema esterno FitCal Secure Mail Service;
- login con credenziali Google, implementata tramite il sistema esterno Google OAuth.

In entrambi i casi l'utente, dopo aver inserito l'email, fornisce la relativa password: se le credenziali sono corrette, e l'utente era gia' registrato, il sito mostra la Personal Homepage, altrimenti invia una notifica di errore.

Infine, il pannello di registrazione mostra, in basso, l'icona "Sign up", che l'utente dovra' cliccare se, appunto, non dispone ancora di un account.

4. Pagina registrazione

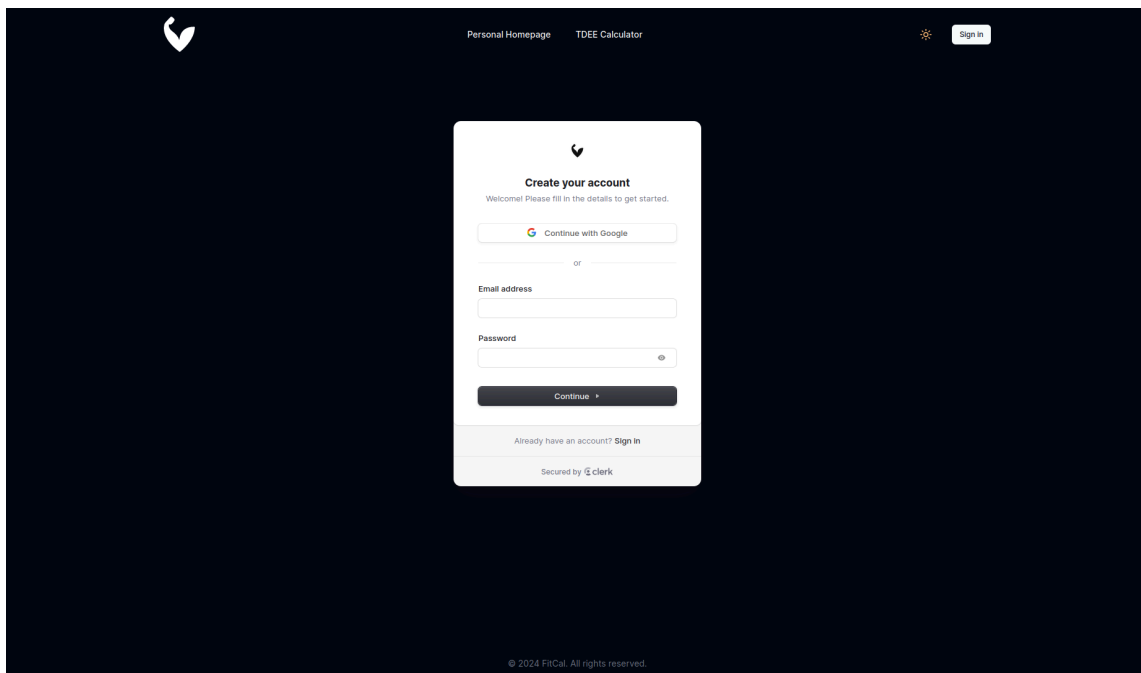


Fig. 29: GET Get Food Informations.

Questa pagina permette all'utente che non dispone ancora di un account di registrarsi. Anche qui, come nel caso del login, vengono fornite due opzioni:

- registrazione tramite credenziali FitCal, modalita' implementata dal sistema esterno FitCal Secure Mail Service;
- registrazione tramite credenziali Google, implementata questa tramite il sistema Google OAuth.

Scegliendo la registrazione con Google OAuth, l'utente dovra' fornire delle credenziali di un account Google esistente, per poi concludere il processo come in ogni registrazione Google.

Il sistema di posta di FitCal procede in modo simile, inviando anche un codice di sicurezza alla mail fornita dall'utente, che dovra' essere inserito per portare a termine la verifica.

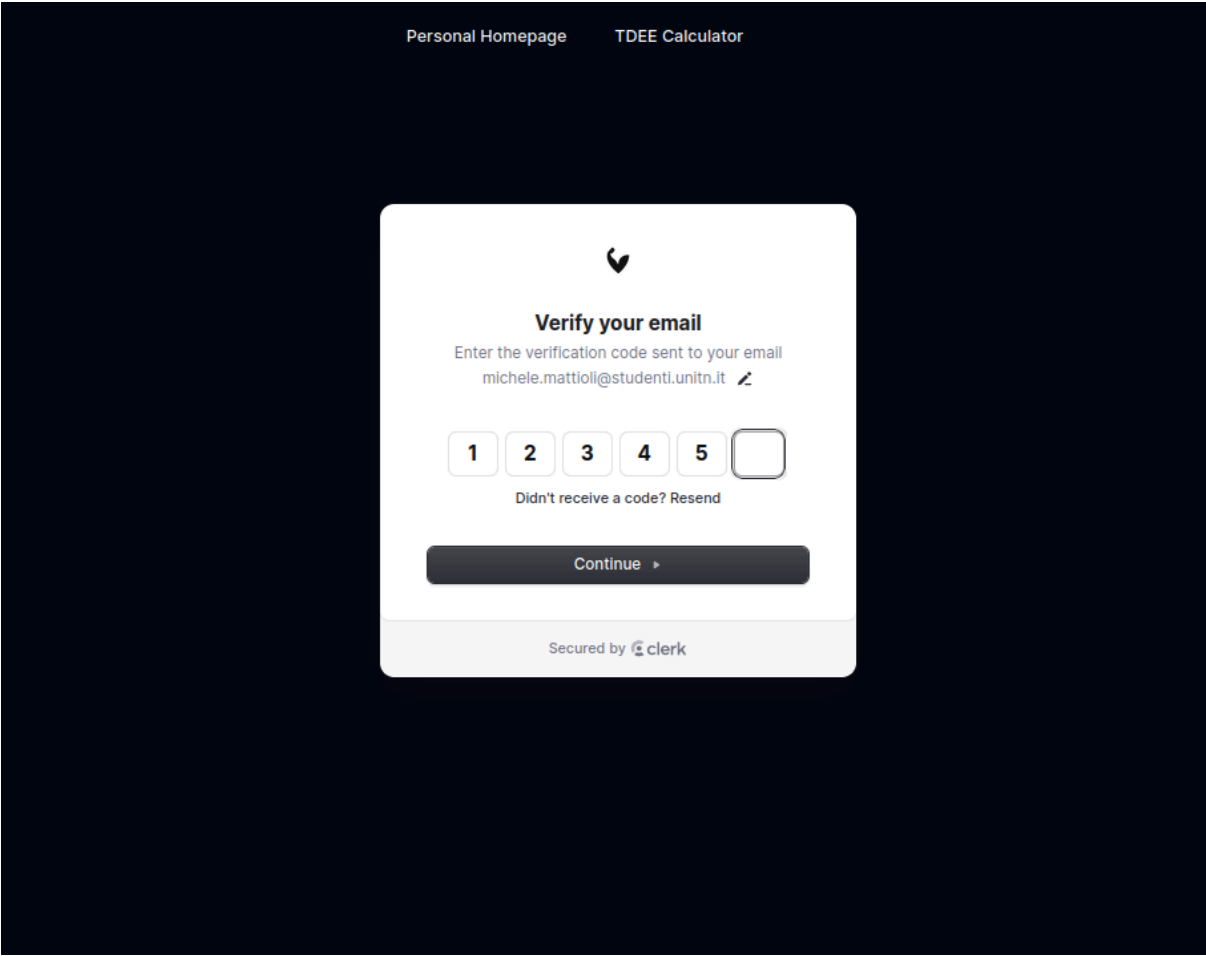


Fig. 30: GET Get Food Informations.

5. Personal Home page

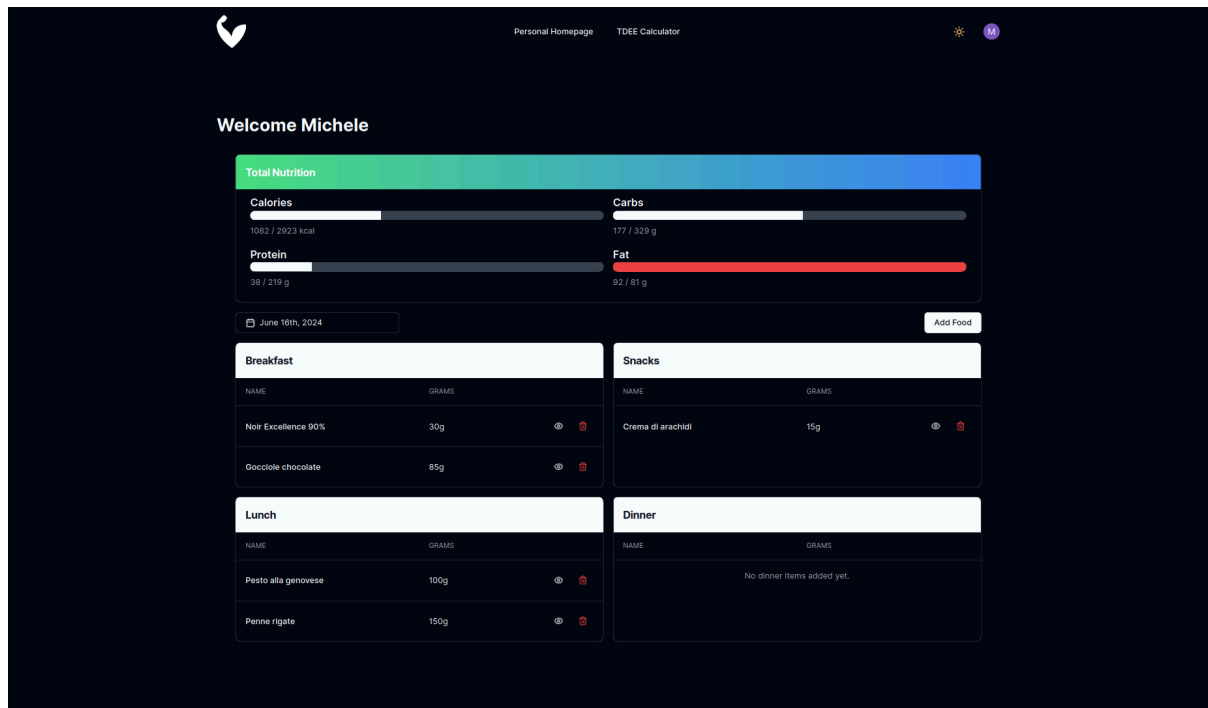


Fig. 31: GET Get Food Informations.

Questa pagina costituisce il fulcro dell'applicazione FitCal, in quanto la maggior parte dei requisiti funzionali vengono svolti da qua. All'utente, a questo punto non più anonimo, vengono mostrate due sezioni principali:

- Total Nutrition, che contiene l'avanzamento dell'utente rispetto al TDEE (Total Daily Energy Expenditure) giornaliero impostato;
- sezione pasti, che contiene i prodotti inseriti dall'utente, suddivisi in base al pasto scelto.

Nel dettaglio, il pannello Total Nutrition, situato sotto al nome utente, in alto a sinistra, mostra le calorie ed i macronutrienti (Proteine, Carboidrati, Grassi) assunti fino ad adesso. Dalla foto si evince anche che, quando l'utente supera uno dei quattro parametri, la barra del progresso diventa rossa.

In basso abbiamo invece la sezione che permette all'utente di visualizzare come ha organizzato i propri pasti: ogni riga reca il nome del prodotto, la quantità, in grammi, consumata, e due icone, per effettuare delle operazioni su di esso.

Tra le due sezioni principali, sono presenti due icone:

- data attuale, a sinistra: indica lo storico prodotti;
- "Add Food", a destra: indica l'opzione di inserimento prodotto.

Selezionando la prima, l'utente ha accesso allo storico prodotti: scegliendo una determinata data egli potrà vedere i pasti consumati ed il progresso rispetto al TDEE per tale giorno. Se inoltre è una data futura, l'utente ha la possibilità di inserire pure prodotti, organizzando

quindi i pasti della giornata che verra'. Cio', ovviamente, non e' possibile se la data selezionata e' antecedente alla data attuale.

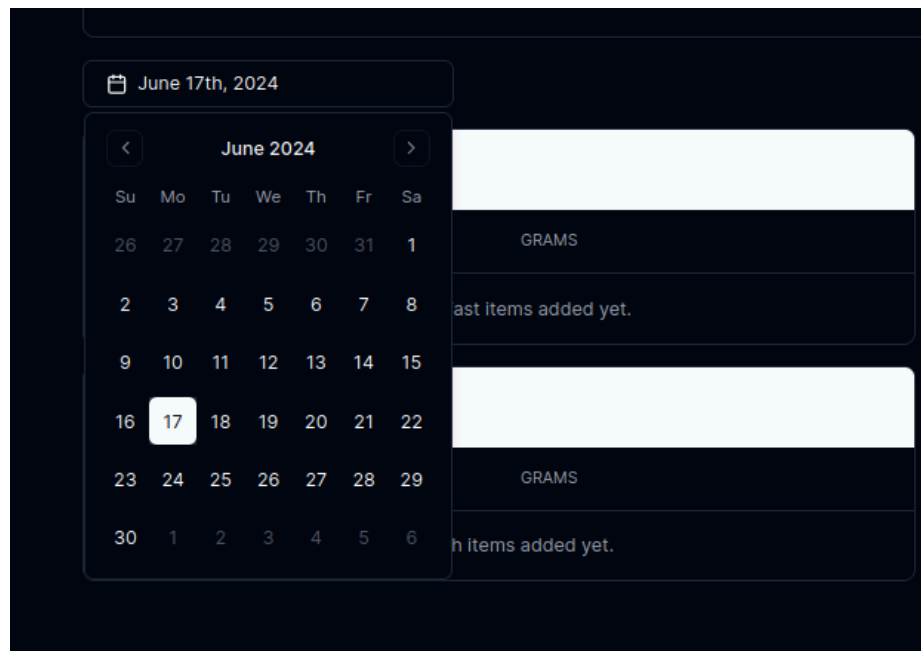


Fig. 32: GET Get Food Informations.

La seconda, quella con scritto “Add Food”, permette di inserire un prodotto in uno dei pasti disponibili. Cliccando, infatti, tale icona, si apre un pannello con tre campi da riempire, in quest’ordine:

- Meal Type: da scegliere tra Breakfast, Snacks, Lunch, Dinner;
- Grams: quantita’ di grammi che si intende consumare;
- Barcode: codice identificativo del prodotto.

Cliccando, poi, sull’icona “Add Item” l’applicazione web aggiunge il prodotto al pasto; se invece si intende annullare l’operazione, ciò e’ possibile cliccando “Cancel”. Nel primo caso, se il barcode e’ corretto e l’operazione va a buon fine, comparirà una notifica sul corretto esito dell’operazione in basso a destra.

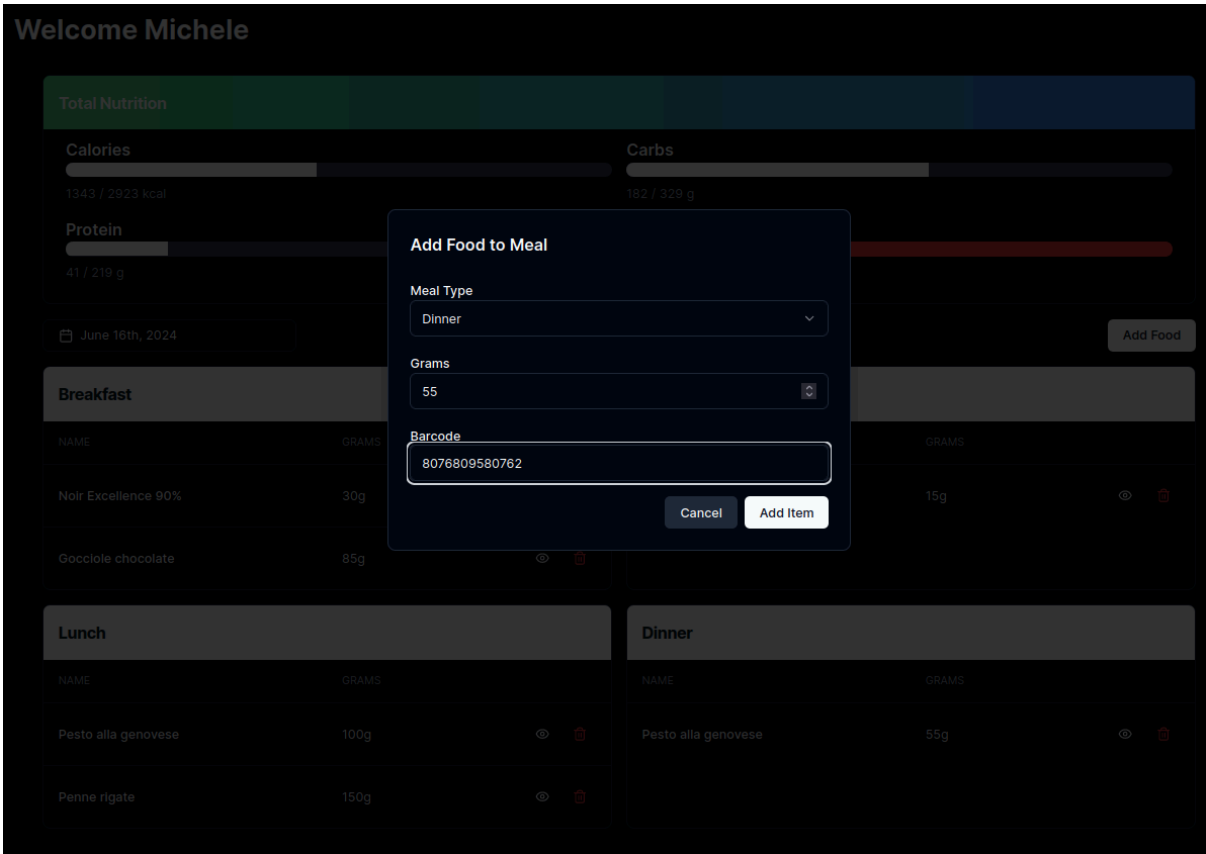


Fig. 33: GET Get Food Informations.

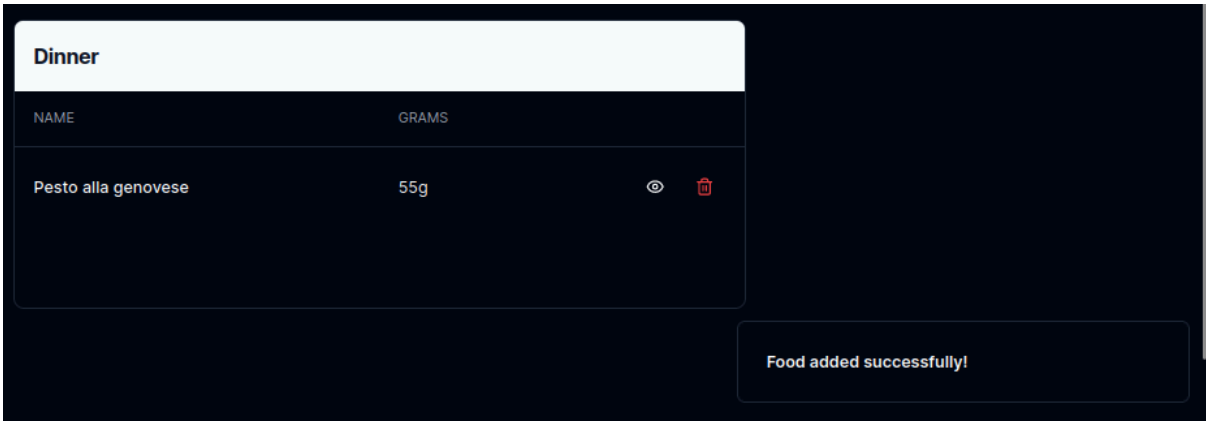


Fig. 34: GET Get Food Informations.

Sempre dalla Personal Homepage l'utente potra' visualizzare le informazioni su un determinato prodotto, semplicemente selezionando l'icona che reca il simbolo dell'occhio accanto al prodotto in questione. Comparira' un pannello che mostra la foto del cibo scelto a sinistra, e le seguenti informazioni a destra:

- nome;
- barcode;
- calorie e macro per la quantita' inserita dall'utente;

- calorie e macro per 100g.

E' possibile uscire da tale schermata cliccando "Close" in basso a destra.

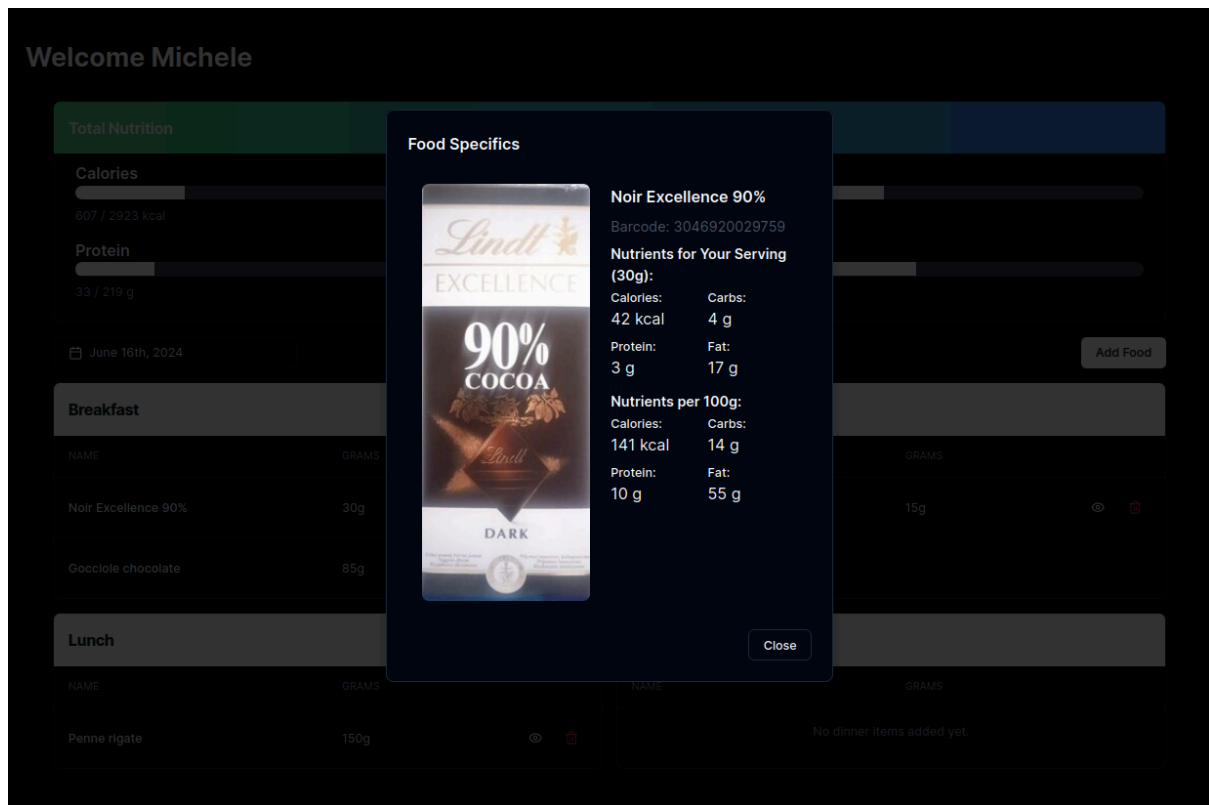


Fig. 35: GET Get Food Informations.

Infine, dalla Personal Homepage l'utente puo' eliminare un determinato prodotto, con conseguente aggiornamento della sezione del progresso giornaliero. Cio' e' possibile selezionando l'icona che reca il simbolo di un cestino rosso, a destra del nome del prodotto in questione. Subito dopo la rimozione, una notifica di corretta esecuzione comparirà in basso, sulla destra.

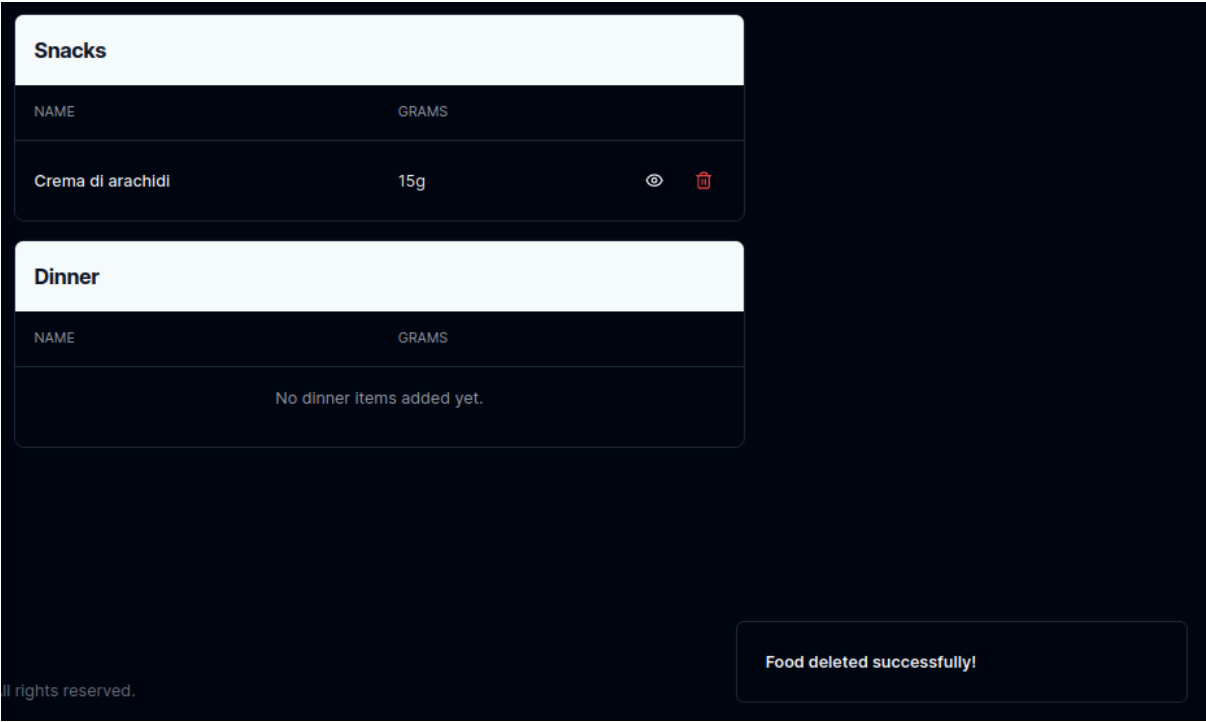


Fig. 36: GET Get Food Informations.

6. Pagina TDEE Calculator

Fig. 37: GET Get Food Informations.

Tramite questa pagina l'utente può inserire i dati per ricavare il proprio TDEE. Essa è composta da due pannelli:

- sulla destra abbiamo la sezione per specificare i parametri, anche tramite freccette. I parametri in questione sono:
 1. Weight (Kg);
 2. Height (cm);
 3. Age;
 4. Gender;
 5. Activity Level.
- cliccando su "Check preview" l'applicazione FitCal mostra sulla sinistra il TDEE, calcolato, comprendente del numero di calorie e della quantità di macronutrienti da assumere giornalmente dall'utente.

7. Pagina Gestione Account: Profile

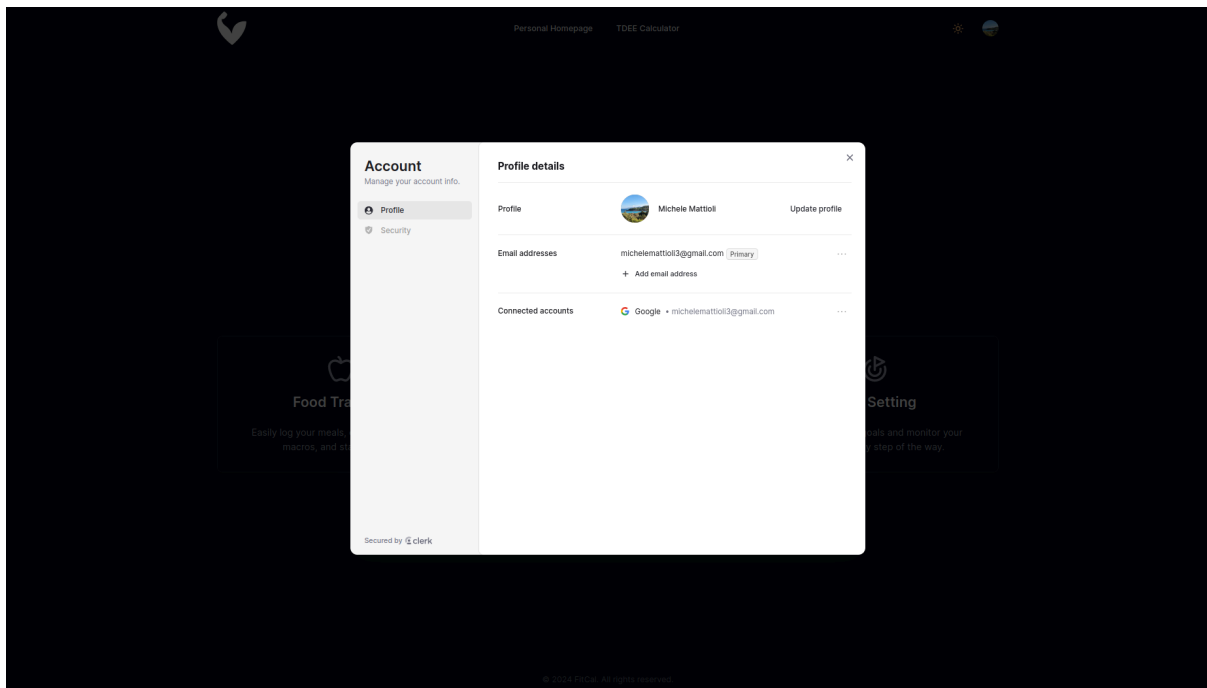


Fig. 38: GET Get Food Informations.

Questa pagina permette all'utente di visualizzare i dati del suo profilo, che comprendono:

- nome utente;
- foto profilo;
- email registrate;
- account collegati.

E' possibile, inoltre, modificare la foto profilo, cliccando su "Update profile" e poi su "Upload", a patto che tale foto non superi la dimensione di 10 MB. Per quanto riguarda le email registrate e gli account collegati, anch'essi sono modificabili:

- e' possibile sia rimuovere che aggiungere un indirizzo email; quest'ultima operazione richiederà la verifica di un codice a sei cifre, che verrà inviato all'indirizzo mail fornito.
- e' possibile rimuovere un account connesso.

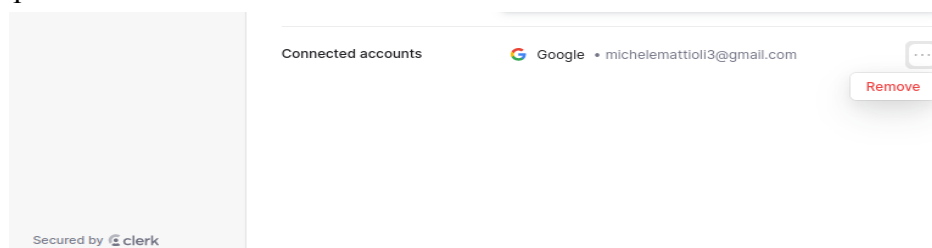


Fig. 39: GET Get Food Informations.

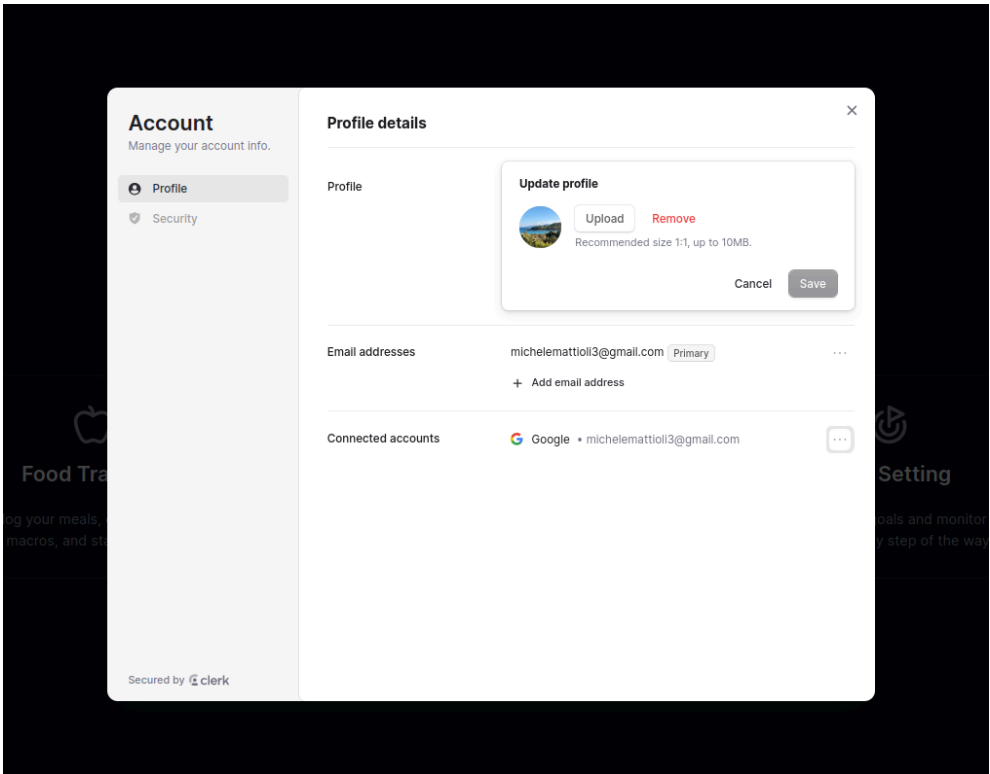


Fig. 40: GET Get Food Informations.

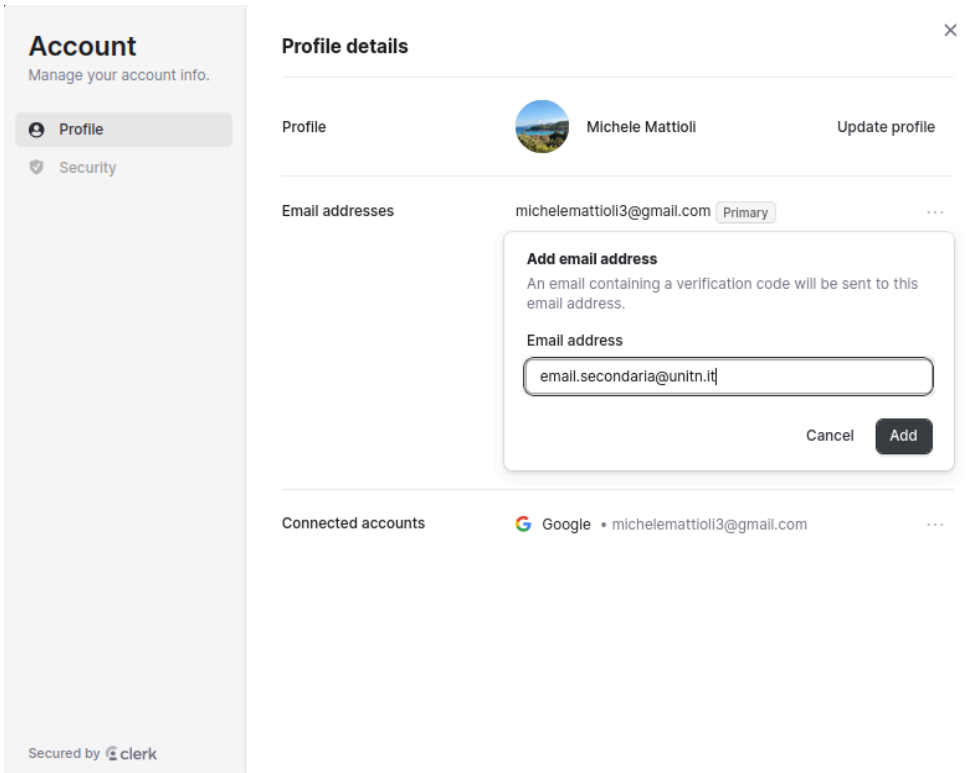


Fig. 41: GET Get Food Informations.

8. Pagina Gestione Account: Security

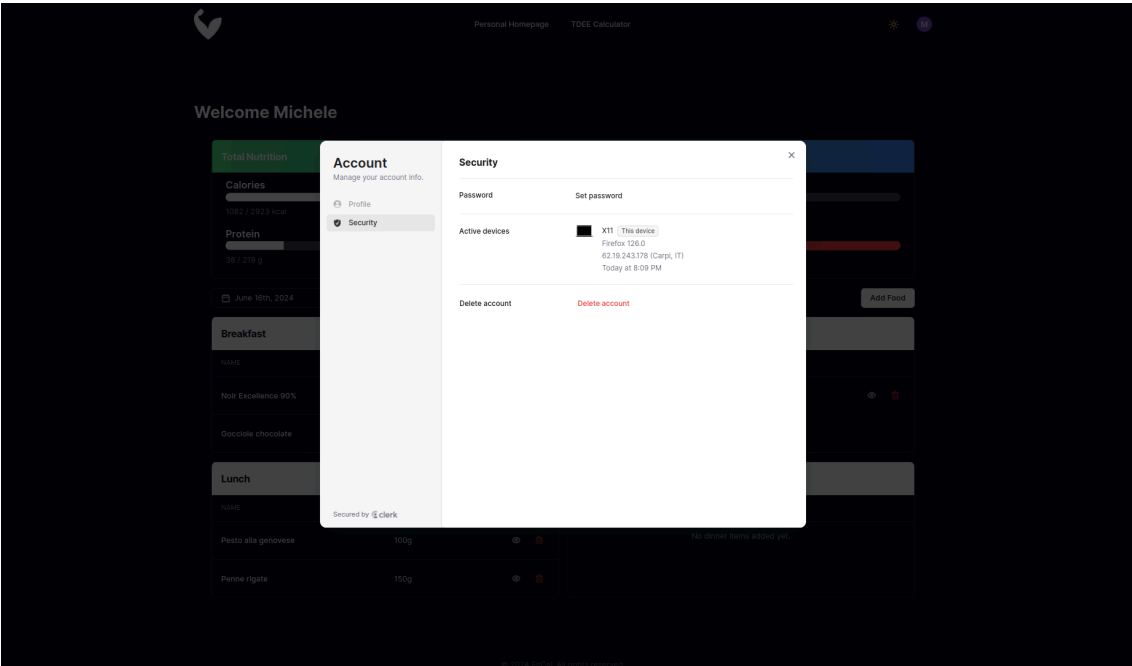


Fig. 42: GET Get Food Informations.

Questa pagina permette all'utente di effettuare operazioni sulle proprie credenziali di accesso, oltre che visualizzare i dispositivi connessi. Partendo dalla password, e' possibile inserirne una nuova, cliccando su "Set password" e inserendo per due volte quella nuova.

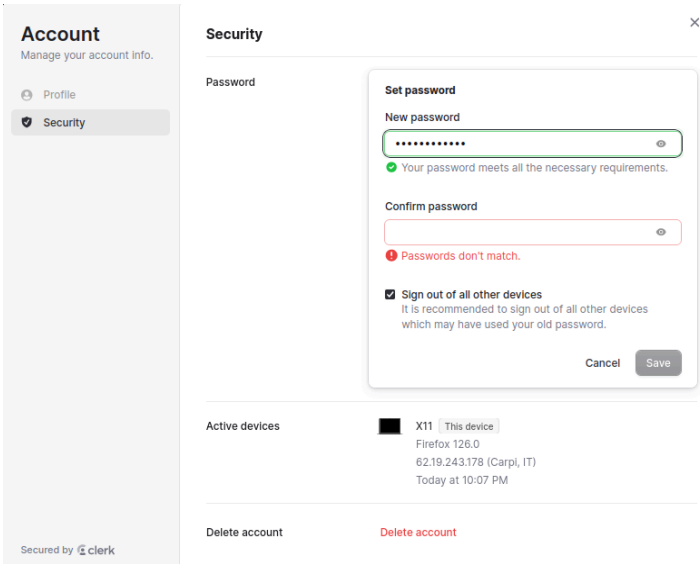


Fig. 43: GET Get Food Informations.

Per quanto riguarda la rimozione dell'account, tale funzionalita' prevede, dopo aver selezionato "Delete account", di digitare la frase "Delete account", come conferma

dell'operazione. In caso i caratteri inseriti siano corretti, il sito web procede con l'eliminazione dell'account, reindirizzando l'utente alla Home Page, da cui potrà nuovamente effettuare la registrazione a FitCal. Al contrario, in caso di errore il sistema mostrerà una notifica sull'esito non corretto dell'operazione.

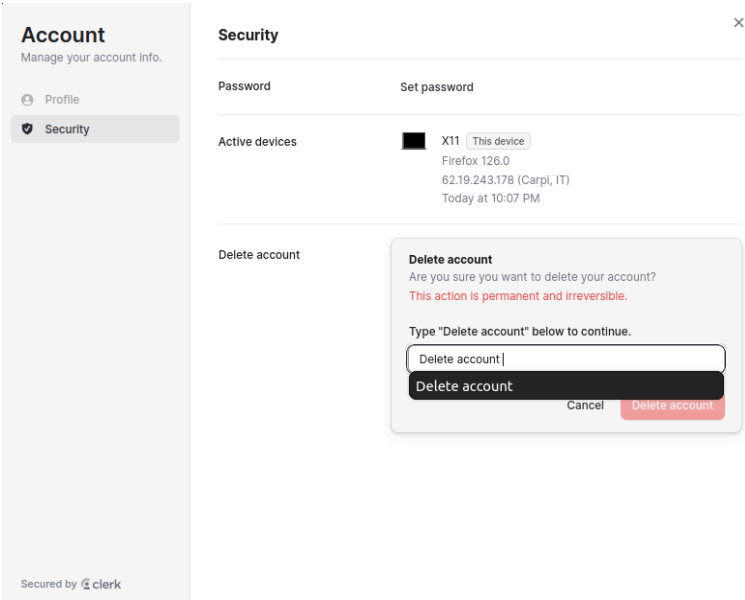


Fig. 44: GET Get Food Informations.

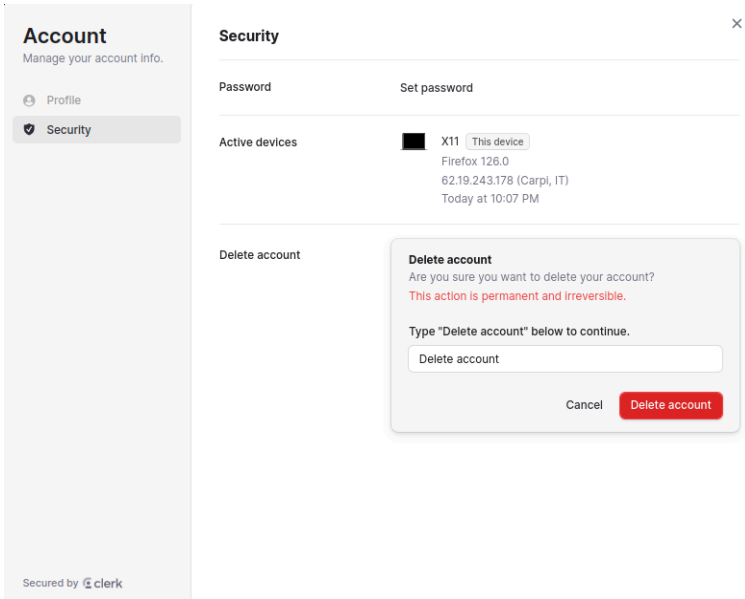


Fig. 45: GET Get Food Informations.

9. Gestione account: Sign out

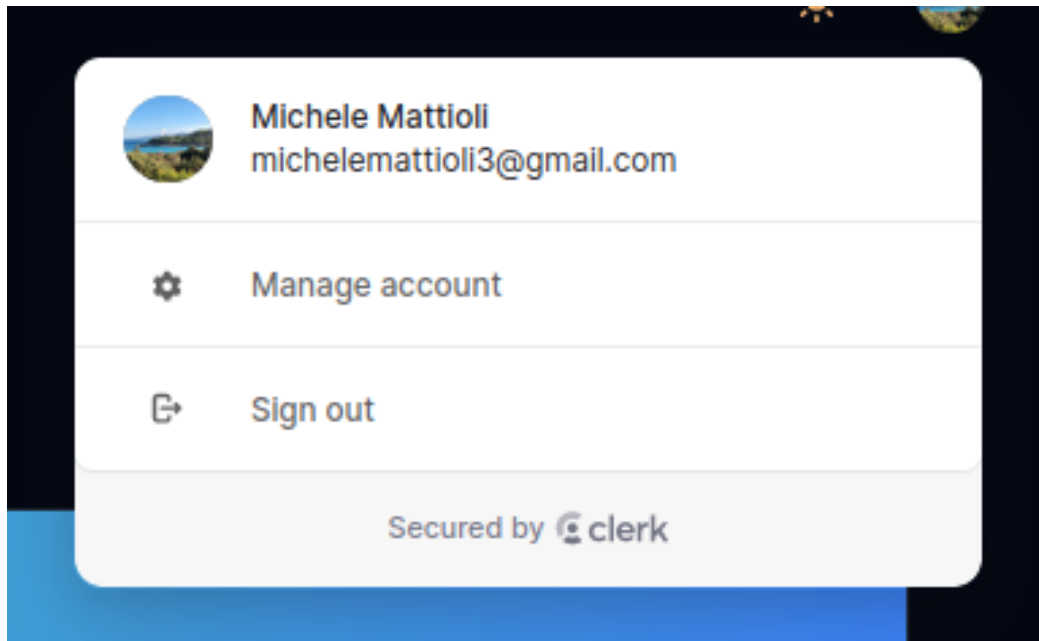


Fig. 46: GET Get Food Informations.

Questa pagina permette di scollegare il proprio account dal sito web. Cliccando l'icona "Sign out" l'utente sarà indirizzato alla Home Page, dalla quale dovrà rifare l'autenticazione.

Testing

Per il testing, abbiamo sfruttato la libreria Jest e il metodo fetch per interagire con le API. I test sono organizzati nella directory 'test/[nomeModello]'. In questa directory, undici file ".test.js" coprono i casi di test per ogni modello con API definite.



Fig. 47: Struttura cartella test.

Prendiamo come esempio `user.test.js`, la cui struttura è rappresentativa di tutti i nostri file di `test.z`

```

1  const supertest = require('supertest');
2  const { app, startServer, closeServer } = require('../server'); // Import the app object
3  const nock = require('nock');
4
5
6  const User = require('../models/userModel');
7
8  const apiUrl = 'https://fitcal-api.kevinazemi.com';
9
10 describe('API Integration Tests', () => {
11   beforeAll(async () => {
12     await startServer();
13   });
14
15   afterAll(async () => {
16     await closeServer();
17   });
18
19   describe('getUser (External API Integration)', () => {
20
21     it('should fetch user data from the external API for a valid clerkUserId', async () => {
22       const clerkUserId = 'user_2h4MM1Eqxwsur9rGAN4lTN1dJ0l';
23
24       // Mock the external API response
25       nock(apiUrl)
26         .get(`/api/users/clerk/${clerkUserId}`)
27         .reply(200);
28
29       const res = await supertest(app).get(`/api/users/clerk/${clerkUserId}`);
30
31       expect(res.status).toBe(200);
32     });
33
34     it('should return 404 for a non-existent clerkUserId', async () => {
35       const clerkUserId = 'non-existent-user';
36
37       // Mock the external API to return a 404
38       nock(apiUrl)
39         .get(`/api/users/clerk/${clerkUserId}`)
40         .reply(404);
41
42       const res = await supertest(app).get(`/api/users/clerk/${clerkUserId}`);
43
44       expect(res.status).toBe(404);
45     });
46   });
47
48   describe('updateUserNeeds (External API Integration)', () => {
49     it('should successfully update user needs for a valid clerkUserId', async () => {
50       const clerkUserId = 'user_2h4MM1Eqxwsur9rGAN4lTN1dJ0l';
51       const updatedNeedsPayload = { // The payload sent in the request
52         needs: {
53           calories: 1750,
54           carbohydrates: 200,
55           proteins: 130,

```

Fig. 48: Esempio file test `user.test.js`.

Questo script di test verifica l'integrazione tra l'applicazione backend locale e le nostre API esterna pubbliche (situata all'indirizzo <https://fitcal-api.kevinazemi.com/api>). I test coprono tre endpoint chiave:

1. *GET /api/users/clerk/:clerkUserId*: Recupera i dati dell'utente dall'API esterna;
2. *PATCH /api/users/clerk/:clerkUserId*: Aggiorna le esigenze nutrizionali dell'utente nell'API esterna;
3. *DELETE /api/users/clerk/:clerkUserId*: Elimina l'utente sia dal database locale che dall'API esterna.

Dipendenze:

- **supertest**: Libreria per eseguire chiamate HTTP all'API in fase di test.
- **nock**: Libreria per simulare le risposte dell'API esterna, isolando i test dal comportamento reale dell'API;
- **jest**: Framework di testing di JavaScript, utilizzato per organizzare e gestire i test, e per definire assertion (verifiche);
- **app, startServer, closeServer**: Moduli importati dall'applicazione backend per avviare e chiudere il server durante i test;
- **User**: Modello Mongoose per interagire con la collezione degli utenti nel database locale.

Struttura del Test:

1. **beforeAll**: Esegue *startServer* per preparare l'ambiente di test prima di tutti i casi di test.
2. **afterAll**: Esegue *closeServer* per terminare il server dopo tutti i test.
3. **describe (API Integration Tests)**: Raggruppa tutti i casi di test relativi all'integrazione API.
 - **describe (getUser)**: Raggruppa i casi di test per l'endpoint *GET /api/users/clerk/:clerkUserId*.
 - **it ("should fetch...")**: Verifica il recupero corretto dei dati per un *clerkUserId* valido.
 - **it ("should return 404...")**: Verifica la risposta 404 per un *clerkUserId* inesistente.
 - **describe (updateUserNeeds)**: Raggruppa i test per *PATCH /api/users/clerk/:clerkUserId*.
 - **it ("should successfully update...")**: Verifica l'aggiornamento corretto delle esigenze.
 - **it ("should return 404...")**: Verifica la risposta 404 per un *clerkUserId* inesistente.
 - **describe (deleteUser)**: Raggruppa i test per *DELETE /api/users/clerk/:clerkUserId*.
 - **it ("should successfully delete...")**: Verifica l'eliminazione corretta dell'utente.
 - **it ("should return 404...")**: Verifica la risposta 404 per un *clerkUserId* inesistente.

Utilizzo di Nock:

In ogni caso di test, Nock viene utilizzato per intercettare le richieste verso l'API esterna e fornire risposte simulate. Questo permette di testare il comportamento dell'applicazione backend indipendentemente dallo stato reale dell'API esterna.

Utilizzo di Jest:

Jest fornisce le funzioni *describe*, *it*, *beforeAll*, *afterAll*, *expect* per strutturare i test e definire le assertion. *jest.spyOn* viene utilizzato per monitorare le chiamate al metodo *findOneAndDelete* del modello *User*.

Risultati del testing

Per poter eseguire i test, nel file `package.json` abbiamo aggiunto una riga per poter invocare `jest`.

```
"scripts": {  
  "test": "jest",  
  "serve": "node server.js",  
  "dev": "nodemon server.js"  
},
```

Fig. 49: File `package.json` con test.

Eseguendo il comando “`npm test`” nella directory principale del progetto, vengono eseguiti tutte le suites di test presenti nella cartella `test`.

Esempio di test:

```
Tests:      21 passed, 21 total  
Snapshots:  0 total  
Time:       2.034 s, estimated 3 s  
Ran all test suites.
```

Fig. 50: Esempio risultato test.

In questo caso, vediamo che vengono eseguiti e superati 21 test con successo.

GitHub repository e Deployment

Al seguente URL e' possibile scaricare e vedere tutte le repository del progetto:

<https://github.com/orgs/FitCal-app/repositories>

Il sito completo e funzionante è visionabile al seguente link:

<https://fitcal.kevinazemi.com/>

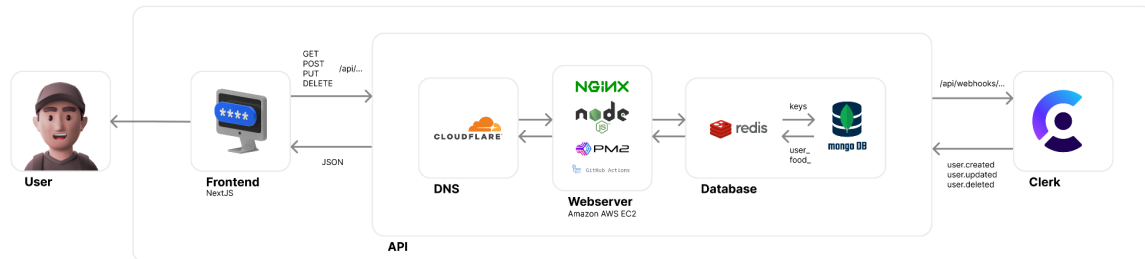


Fig. 51: Schema infrastruttura.

Come da schema, per poter effettuare il deploy in produzione del nostro progetto abbiamo dovuto utilizzare alcuni servizi.

Front-end:

Il front-end e' hostato su [vercel](https://vercel.com), e per poterne fare il deploy sono necessarie le seguenti variabili d'ambiente:

```
NEXT_PUBLIC_CLERK_PUBLISHABLE_KEY=' '
CLERK_SECRET_KEY=' '

NEXT_PUBLIC_API_URL=' '
```

Fig. 52: Variabili Frontend.

NEXT_PUBLIC_CLERK_PUBLISHABLE_KEY e CLERK_SECRET_KEY vengono popolate grazie a dei dati presenti sul proprio account di [clerk](https://clerk.com), il nostro sistema di autenticazione.

Invece, in NEXT_PUBLIC_API_URL va inserito l'url delle API, da cui il frontend andrà a fare le richieste per prendere le varie informazioni dal database.

Per poter utilizzare il dominio fitcal.kevinazemi.com, abbiamo creato un record CNAME su [cloudflare](https://cloudflare.com), che punta ai server di vercel.

Back-end:

Le nostre API sono pubbliche, ed e' possibile accedervi al seguente indirizzo:

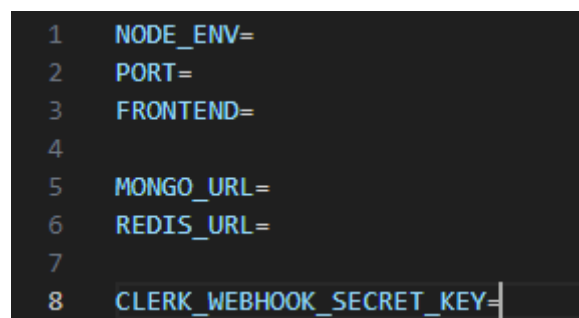
<https://fitcal-api.kevinazemi.com/api>

Come web server, utilizziamo [Nginx](#) hostato in una macchina EC2 di [Amazon AWS](#).

Per poter effettuare il deploy automatico da github, abbiamo implementato una Github Action. È possibile visualizzare i workflow direttamente dalla [repository delle api](#) presenti nel nostro github. Il server node viene avviato e tenuto online da [pm2](#), un process manager per node.js.

Riguardo il database, per il nostro sistema di caching utilizziamo un istanza gratuita offerta da [redis lab](#). Invece riguardo al database persistente, utilizziamo anche qui un istanza gratuita gentilmente offerta dai cluster cloud di [MongoDB](#).

Per poter eseguire le nostre API, sono necessarie alcune variabili d'ambiente:



```
1  NODE_ENV=  
2  PORT=  
3  FRONTEND=  
4  
5  MONGO_URL=  
6  REDIS_URL=  
7  
8  CLERK_WEBHOOK_SECRET_KEY=
```

Fig. 53: Variabili backend API.

NODE_ENV identifica il come vogliamo avviare le nostre API, puo' assumere i seguenti valori:

- production
- development
- test

PORT indica la porta in cui vogliamo che venga eseguito il processo del server.

FRONTEND indica l'url del nostro frontend, questo ci e' utile per andare a definire l'origine dei cors.

MONGO_URL e' l'url di connessione al database MongoDB.

REDIS_URL e' l'url di connessione al database Redis.

CLERK_WEBHOOK_SECRET_KEY e' la key per poter utilizzare i webhook di clerk.

Tutte le nostre API sono state documentate utilizzando [Postman API Documentation](#).

E' possibile accedervi al seguente link:

<https://fitcal-docs.kevinazemi.com>