



---

## Ingegneria del Software

Dipartimento di Ingegneria e Scienza dell'Informazione  
Anno accademico 2022 - 2023  
Bevilacqua Lorenzo - Sartore Alessandro - Tecchio Luca

Hungry

Everywhere



**Titolo del documento:**  
**Sviluppo applicazione**

# INDICE

<b>1. SCOPO DEL DOCUMENTO</b>	<b>6</b>
<b>2. USER FLOWS</b>	<b>7</b>
Cliente	7
GestoreRistorante	8
<b>Figura 2. User Flow GestoreRistorante</b>	<b>9</b>
<b>3. APPLICATION IMPLEMENTATION AND DOCUMENTATION</b>	<b>10</b>
3.1 PROJECT STRUCTURE	10
3.2 PROJECT DEPENDENCIES	11
3.3 PROJECT DATA OR DB	12
Figura 4. Collezione Dati usati nell'applicazione	12
Figura 5. Tipo di dato “menu”	13
Figura 6. Tipo di dato “pietanza”	13
Figura 8. Tipo di dato “utente”	14
3.4 PROJECT APIs	15
3.4.1 RESOURCES EXTRACTION FROM THE CLASS DIAGRAM	15
SistemaPagamento	15
Cliente	15
GestoreRistorante	16
Ordine	17
Ristorante	17
Carrello	18
Posizione	18

---

3.4.2 Resources Model	20
Procedure di autenticazione (DA RIVEDERE)	20
GestoreRistorante	21
Ristorante	24
Carrello	27
Posizione	28
3.5 Sviluppo API	30
3.5.1 Login	30
3.5.2 Register	31
3.5.3 Thumbnails	32
3.5.4 My Restaurants	33
3.5.5 Get Menu	34
3.5.6 Add Menu	35
3.5.7 Edit Menu	36
3.5.8 Delete Menu	37
3.5.9 Get Restaurant Info	38
3.5.10 Add Restaurant	39
3.5.11 Edit Restaurant	40
3.5.12 Delete Restaurant	41
<b>4. API documentation</b>	<b>42</b>
<b>5. FrontEnd Implementation</b>	<b>43</b>
5.1 - HomePage	43
5.2 - Pagina di Login	44
5.3 - Recupero password	45

---

5.4 - Registrati	46
5.5 - Il mio profilo	47
5.6 - Modifica email	48
5.7 - Modifica password	49
5.8 - Ristoranti vicini (cliente)	50
5.9 - Dettagli ristorante (cliente)	51
5.10 - Focus pietanza	52
5.11 - Carrello	53
5.12 - I miei preferiti	54
5.13 - Ristoranti vicini	55
5.14 - Dettagli ristorante (gestore)	56
5.15 - Schermata dei ristoranti del gestore	57
5.16 - Elimina ristorante	58
5.17 - Schermata di aggiunta/modifica ristorante	59
5.18 - Riepilogo ordini del gestore	60
5.19 - Focus sull'ordine	61
5.20 - Pagine accessibili dal cliente	62
5.21 - Pagine accessibili dal gestore	63
<b>6. GitHub Repository and Deployment Info</b>	<b>64</b>
6.1 Struttura GitHub Organization	64
6.2 Struttura Codebase	64
6.3 Deployment locale	65
6.4 Deployment su Vercel	65
<b>7. Testing</b>	<b>66</b>

7.1 API testate con Jest 66

7.2 API testate con Postman 68

## 1. SCOPO DEL DOCUMENTO

Il presente documento riporta tutte le informazioni necessarie per lo sviluppo di una parte dell'applicazione Hungry Everywhere. In particolare, presenta tutti gli artefatti necessari per realizzare i servizi di gestione dei clienti e dei gestori dei ristoranti dell'applicazione Hungry Everywhere. Partendo dalla descrizione degli user flow legati al cliente e al gestore, il documento prosegue con la presentazione delle API necessarie (tramite l'API Model e il Modello delle risorse) per poter aggiungere prodotti al carrello del cliente, aggiungere, modificare e rimuovere ristoranti, gestire gli ordini e recuperare le informazioni riguardanti la posizione, il tutto all'interno di Hungry Everywhere. Per ogni API realizzata, oltre ad una descrizione delle funzionalità fornite, il documento presenta la sua documentazione e i test effettuati. Infine una sezione è dedicata alle informazioni del Git Repository e il deployment dell'applicazione stessa.

## 2. USER FLOWS

### Cliente

In questa sezione del documento di sviluppo riportiamo gli “user flows” per il ruolo del Cliente nella nostra applicazione. La figura 1 descrive il flow di operazioni che un utente può fare. Dato che l’utente ha la possibilità di muoversi liberamente tra le pagine, il seguente user flow è stato realizzato in modo che l’utente faccia le operazioni nel modo più naturale possibile. Una volta registrato e fatto il login viene considerato come Cliente. Dalla pagina principale il Cliente inserisce il proprio indirizzo (o utilizza le API di Google) e visualizza i ristoranti vicini a sé. Da qui in poi l’utente, se ha effettuato l’accesso come Cliente, può terminare la sessione effettuando il logout. Dalla visualizzazione dei ristoranti è possibile selezionare il singolo ristorante da cui sarà possibile visualizzare il menù, modificare e aggiungere pietanze al carrello. Per questo passaggio viene eseguito un controllo sulla sessione per verificare se l’utente è autenticato o meno, nel caso non lo sia viene riportato nella pagina di login. Una volta finito l’ordine, è possibile visionare il carrello per rimuovere o modificare le pietanze inserite. Il Cliente può tornare alla pagina dei ristoranti senza pagare l’ordine, ma ciò cancella il carrello corrente. Infine il Cliente può passare alla consegna e al pagamento dell’ordine. Il Cliente può anche accedere alla pagina “Il mio profilo” da cui potrà apportare modifiche ai propri dati. Ad ogni modo nella figura 1 sono illustrate tutte le varie interazioni tra il Cliente e l’applicazione, ma anche le varie relazioni tra le azioni. Nella figura 1 è anche presente una legenda che descrive i simboli usati nello user flow.

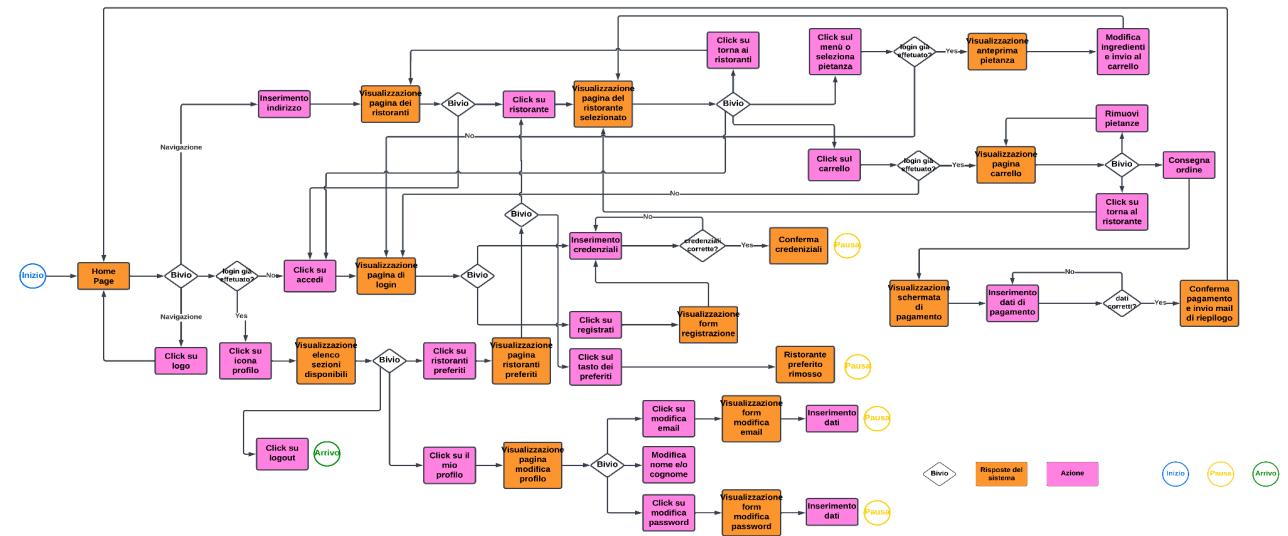
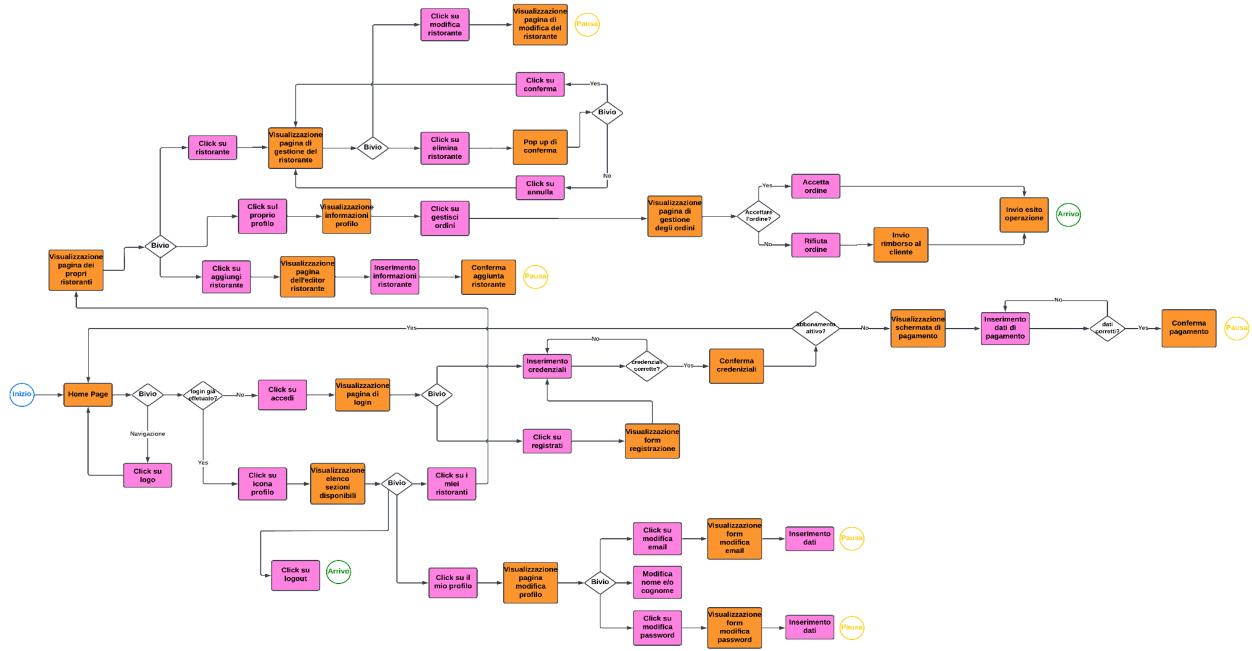


Figura 1. User Flow Cliente

## GestoreRistorante

In questa sezione del documento di sviluppo riportiamo gli “user flows” per il ruolo del GestoreRistorante nella nostra applicazione. La figura 2 descrive lo user flow relativo alle operazioni che un gestore può svolgere all’interno dell’applicazione Hungry Everywhere. Dato che il gestore ha la possibilità di muoversi liberamente tra le pagine, il seguente user flow è stato realizzato in modo che il gestore faccia le operazioni nel modo più naturale possibile. Il gestore, per essere considerato tale e distinguersi dal cliente, deve necessariamente effettuare il login come ristoratore nella home page del sito. Nel caso possieda già le credenziali le dovrà inserire negli appositi campi, altrimenti dovrà effettuare una registrazione mediante un form. Da qui in poi il gestore può terminare la sessione effettuando il logout. Per poter gestire i propri ristoranti, il gestore deve sottoscrivere un abbonamento che gli permette di aggiungere, modificare tramite apposito form o eliminare i propri ristoranti. Per ogni ordine il gestore potrà scegliere se accettarlo o rifiutarlo, in tal caso però invierà anche il rimborso, ed invierà una mail per notificare al cliente l’esito dell’operazione. Il Gestore può anche accedere alla pagina “Il mio profilo” da cui potrà apportare modifiche ai propri dati. Nella figura 2 sono illustrate le varie interazioni tra il gestore e l’applicazione, ma

anche le varie relazioni tra le azioni. Nella figura 1 è anche presente una legenda che descrive i simboli usati nello user flow.



*Figura 2. User Flow GestoreRistorante*

## 3. APPLICATION IMPLEMENTATION AND DOCUMENTATION

Nelle sezioni precedenti abbiamo identificato le varie features che devono essere implementate per la nostra applicazione con un'idea di come il nostro cliente o il nostro gestore potranno utilizzarle nel loro flusso applicativo. L'applicazione è stata sviluppata principalmente utilizzando JavaScript e TypeScript, con l'utilizzo di NextJs e React come framework e Chakra UI per il FrontEnd. Per la gestione dei dati abbiamo utilizzato MongoDB.

### 3.1 PROJECT STRUCTURE

```

components
├── authentication.tsx
├── chakraNextLink.tsx
├── dish.tsx
├── footer.tsx
├── layout.tsx
├── navbar.tsx
└── restaurant.tsx
└── utils.tsx
lib
├── models
│   ├── dbConnect.ts
│   ├── emotion-cache.ts
│   ├── typings.ts
│   └── yupSchemas.ts
pages
└── api
    ├── auth
    │   ├── menu.ts
    │   ├── myrestaurants.ts
    │   ├── restaurant.ts
    │   └── thumbnails.ts
    ├── auth
    ├── ristorante
    ├── utente
    └── ...
    └── _app.tsx
    └── _document.tsx
    └── api-doc.jsx
    └── carrello.jsx
    └── index.tsx
    └── ristoranti.tsx
public
└── theme
    ├── components
    │   ├── index.ts
    │   └── styles.ts
    └── .eslintrc.json
.git
└── .gitignore
next.config.js
└── package-lock.json
└── package.json
└── README.md
└── tsconfig.json

```

La struttura del progetto è presentata nella figura qui a fianco ed è composta dalle seguenti cartelle principali:

- **components**, in cui sono presenti delle componenti React utilizzate in piu' pagine;
- **lib**, in cui sono presenti funzioni piu' generiche utilizzate sia in back-end che in front-end e i modelli per il database nella cartella **models**;
- **pages**, che rappresenta la struttura effettiva del sito web, al cui interno si trova la cartella **api** che contiene tutte le API implementate;
- **public**, in cui sono presenti risorse statiche a cui hanno bisogno di accedere le varie pagine, tra cui favicon.ico e altre immagini;
- **theme**, che contiene le impostazioni di tema dei componenti di Chakra UI.

*Figura 3. Struttura Codice Sorgente*

Sono presenti altri file utili per il corretto funzionamento e sviluppo dell'applicazione quali:

- **.eslintrc.json**, che è la configurazione del linter per far sì che il codice sia il più possibile coerente;
- **.gitignore**, per far sì che non vengano caricati file importanti sulla repository su GitHub;
- **package.json**, in cui è presente la lista di librerie necessarie per il progetto
- altri file di configurazione di nextjs e typescript.

## 3.2 PROJECT DEPENDENCIES

I seguenti moduli Node sono stati utilizzati e aggiunti al file package.Json:

- next
- react
- chakra-ui
- bcrypt
- formik
- mongoose
- next-auth
- swr
- yup, yup-password

### 3.3 PROJECT DATA OR DB

Per la gestione dei dati utili all'applicazione abbiamo definito quattro principali strutture dati come illustrato in Figura 4. Una collezione per i "menu", una per le "pietanze", una per i "ristoranti" e una per gli "utenti".

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
menus	13	1.29MB	101.87KB	2.09MB	4	144KB	36KB
pietanzas	0	0B	0B	4KB	2	12KB	6KB
ristorantes	8	33.88KB	4.24KB	100KB	2	72KB	36KB
users	11	2.49KB	232B	36KB	2	72KB	36KB

Figura 4. Collezione Dati usati nell'applicazione

---

Per rappresentare le collezioni sopra citate abbiamo definito i seguenti tipi di dati (vedi figure 5, 6, 7 e 8).

```
_id: ObjectId('63af5b0e7fdf18073af45a39')
id: "20e0ffac-7e4a-41be-87c3-6d7181e7356b"
nome: "primi"
ristoranteId: "ce4b9005-bbcf-4451-a435-46d2cfa9307b"
tipologia: "primi"
▼ piatti: Array
  > 0: Object
  > 1: Object
  > 2: Object
__v: 0
```

*Figura 5. Tipo di dato “menu”*

```
id: "93294838-f251-436d-9793-c2a541c33c98"
nome: "Scialatelli al ragù"
prezzo: 8
immagine: "/9j/4AAQSkZJRgABAQAAAQABAAAD/2wCEAAoHCBYWFRgWFRUYGBgZGBoYGBwaGBwcGBkZGB..."
▼ allergeni: Array
  0: "glutine"
  1: "latte e derivati"
calorie: 900
▼ ingredienti: Array
  0: "pasta"
  1: "ragù di manzo"
  2: "grana"
  3: "pepe"
isDisponibile: true
_id: ObjectId('63af5b0e7fdf18073af45a3b')
```

*Figura 6. Tipo di dato “pietanza”*

```
_id: ObjectId('63ae120fed103721ee69e5d4')
id: "ce4b9005-bbcf-4451-a435-46d2cfa9307b"
nome: "Bella Italia"
immagine: "iVBORw0KGgoAAAANSUhEUgAAAGQAAABkCAIAAAD/gAIDAAAAGXRFWHRTb2Z0d2FyZQBBZG..."
indirizzo: "Via Gorizia 5, Trento, 38123"
orariApertura: "11:00 - 14:00
                18:00 - 23:00"
telefono: "+39 3465219086"
sito: "A"
email: "Bella.Italia@restaurants.com"
descrizione: "A"
tipologia: "ristorante"
gestoreId: "75478205-da4b-41d7-a3f1-115cf97786d8"
valutazione: 3
menuIds: Array
  0: "20e0ffac-7e4a-41be-87c3-6d7181e7356b"
  1: "c29057e2-93db-4a69-9df0-00ab10d450b0"
  2: "8b880083-a500-4b46-8cf1-aab7974afc4b"
  3: "1761a406-8ccf-459a-96e6-3acebe3df1fd"
  4: "7e483d67-1183-4295-a126-5eac7d3a394b"
__v: 0
```

*Figura 7. Tipo di dato “ristorante”*

```
_id: ObjectId('63af769a3140b62336edf5bc')
id: "9358344a-2858-4bea-a2dd-2f5151896c82"
name: "Mario"
surname: "Rossi"
isUnitn: false
email: "mariorossi@gmail.com"
password: "$2b$12$Y/oFUAsxH2r2/DYvWw2i/u0MqLNx0GE/nZTw/YE711udcK0mbC23a"
isGestore: true
__v: 0
```

*Figura 8. Tipo di dato “utente”*

## 3.4 PROJECT APIs

### 3.4.1 RESOURCES EXTRACTION FROM THE CLASS DIAGRAM

Abbiamo individuato le seguenti risorse: *SistemaPagamento*, *Cliente*, *GestoreRistorante*, *Ordine*, *Ristorante*, *Carrello*. Per ogni risorsa sono state individuate delle funzionalità.

#### **SistemaPagamento**

Questa risorsa contiene i seguenti attributi: idMittente e idDestinatario. La risorsa implementa la seguente API:

- **validaPagamento:** Utilizza il metodo POST passando come parametri idMittente, idDestinatario e importo. La funzione è quella di validare un pagamento.

#### **Cliente**

Questa risorsa contiene i seguenti attributi: password, email, nome e cognome. La risorsa implementa le seguenti API:

- **login:** Utilizza il metodo POST passando come parametri email e password. La funzione è quella di effettuare il login dell'utente.
- **signout:** Utilizza il metodo POST per terminare la sessione dell'utente e fargli eseguire il logout.
- **signup:** Utilizza il metodo POST passando come parametri email, password, nome, cognome e isGestore. La funzione è quella di registrare l'utente nel database.
- **session:** Utilizza il metodo GET per recuperare la sessione.
- **getRestaurant:** Utilizza il metodo GET con parametro restaurantId e ritorna il ristorante con quell'id.

## GestoreRistorante

Questa risorsa contiene i seguenti attributi: password, email, nome, cognome e ristoranti. La risorsa implementa le seguenti API:

- **login:** Utilizza il metodo POST passando come parametri email e password. La funzione è quella di effettuare il login dell'utente.
- **signout:** Utilizza il metodo POST per terminare la sessione dell'utente e fargli eseguire il logout.
- **signup:** Utilizza il metodo POST passando come parametri email, password, nome, cognome e isGestore. La funzione è quella di registrare il gestore nel database.
- **session:** Utilizza il metodo GET per recuperare la sessione
- **gestioneOrdine:** Utilizza il metodo POST passando come parametri ordine e accettazione. La funzione è quella di confermare o rifiutare un ordine.
- **getOrdine:** Utilizza il metodo GET per recuperare un'ordinazione specifica.
- **getRestaurants:** Utilizza il metodo GET e ritorna tutti i ristoranti del gestore.
- **getRestaurant:** Utilizza il metodo GET con parametro restaurantId e ritorna il ristorante con quell'id.
- **addRistorante:** Utilizza il metodo POST passando come parametri le informazioni del ristorante da aggiungere.
- **updateRistorante:** Utilizza il metodo PUT passando come parametri le informazioni del ristorante che devono venir modificate.
- **deleteRistorante:** Utilizza il metodo DELETE passando come parametri il restaurantId del ristorante che si vuole rimuovere.

## Ordine

Questa risorsa contiene i seguenti attributi: ordine e isValid. La risorsa implementa la seguente API:

- **inviaOrdine:** Utilizza il metodo POST passando come parametro utente. La funzione è quella di inviare l'ordinazione di quell'utente passato come parametro.

## Ristorante

Questa risorsa contiene i seguenti attributi: nome, indirizzo e menù. La risorsa implementa le seguenti API:

- **getInfo:** Utilizza il metodo GET passando come parametro ristorantId. La funzione è quella di ritornare tutte le informazioni del singolo ristorante passato come parametro.
- **getMenu:** Utilizza il metodo GET passando come parametro ristorantId. La funzione è quella di ritornare il menu del singolo ristorante passato come parametro.
- **getThumbnails:** Utilizza il metodo GET passando come parametri il numero di ristoranti da visualizzare e l'indice del ristorante da cui iniziare la visualizzazione. La funzione è quella di ritornare le copertine dei numeri di ristoranti passati come parametro.
- **addMenu:** Utilizza il metodo POST passando come parametri il nome del menù, i piatti e l'id del ristorante a cui inserire il menù.
- **deleteMenu:** Utilizza il metodo DELETE passando come parametro il menuId che corrisponde al menù che si intende eliminare.
- **updateMenu:** Utilizza il metodo PUT passando come parametri il nuovo menù e il menuId che corrisponde al menù che viene modificato.

## Carrello

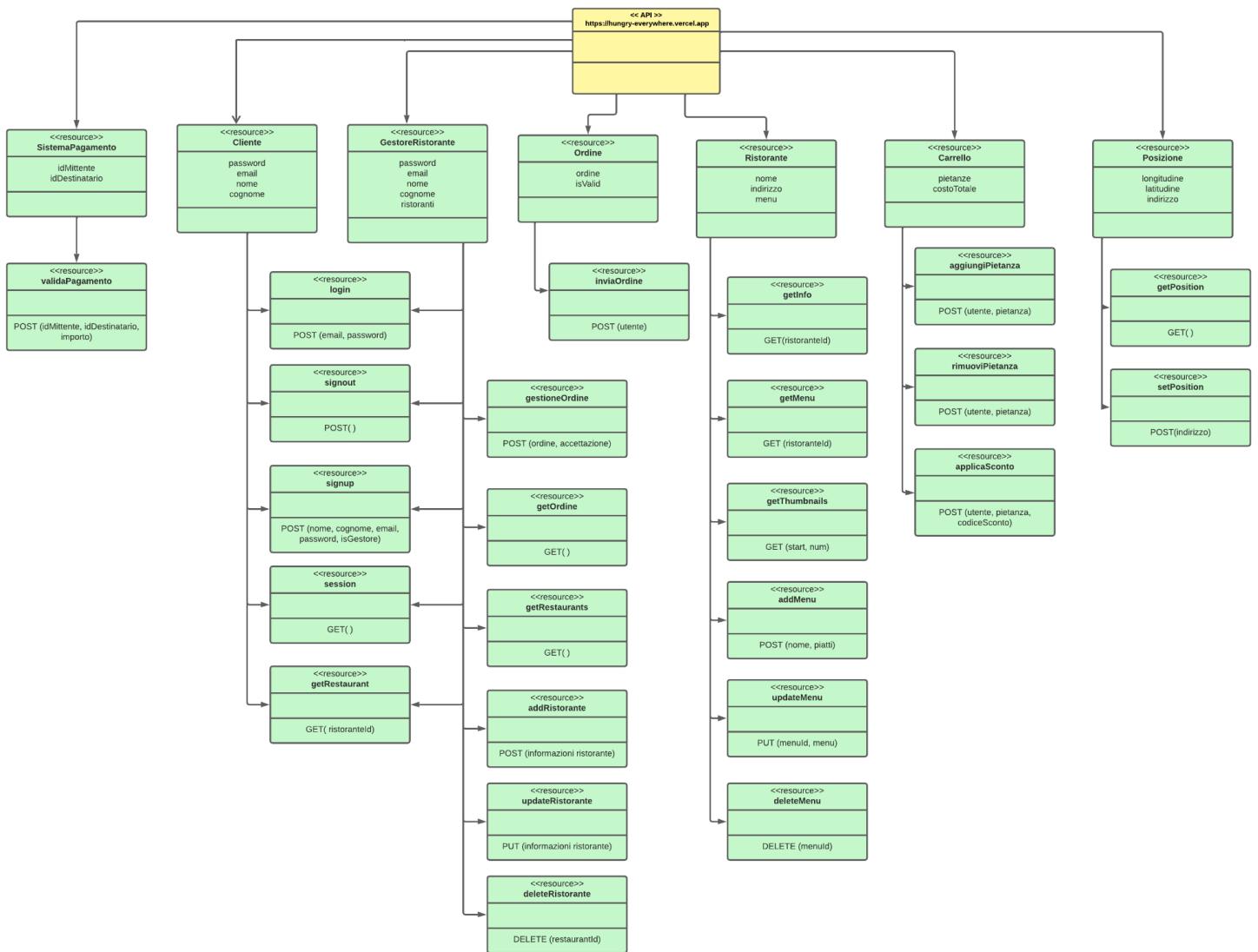
Questa risorsa contiene i seguenti attributi: pietanza e costoTotale. La risorsa implementa le seguenti API:

- aggiungiPietanza: Utilizza il metodo POST passando come parametri utente e pietanza. La funzione è quella di aggiungere la pietanza passata come parametro al carrello dell'utente passato come parametro.
- rimuoviPietanza: Utilizza il metodo POST passando come parametri utente e pietanza. La funzione è quella di rimuovere la pietanza passata come parametro al carrello dell'utente passato come parametro.
- applicaSconto: Utilizza il metodo POST passando come parametri utente, pietanza e codiceSconto. La funzione è quella di controllare come l'utente è autenticato e controllare che per quella pietanza ci sia uno sconto applicabile. Il codiceSconto può essere utilizzato come alternativa per applicare lo sconto.

## Posizione

Questa risorsa contiene i seguenti attributi: longitudine, latitudine e indirizzo. La risorsa implementa le seguenti API:

- getPosition: Utilizza il metodo GET per ritornare la posizione dell'utente nel caso questi abbia permesso l'accesso alla propria posizione.
- setPosition: Utilizza il metodo POST passando come parametro indirizzo. La funzione è quella di impostare l'indirizzo come posizione dell'utente.



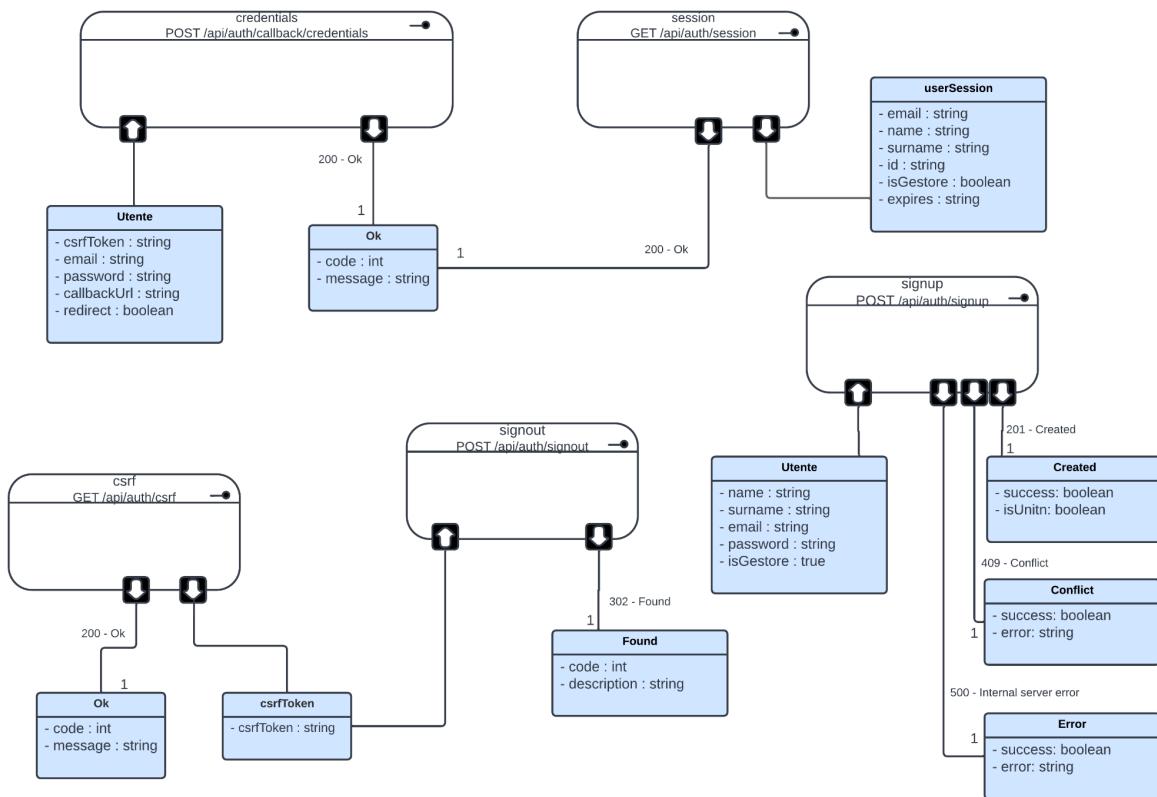
### 3.4.2 Resources Model

Il Resources Model che viene presentato è una specifica del Resources Extraction. Sono presenti solo le API più importanti per il progetto, insieme ad alcune API aggiuntive che sono necessarie per il funzionamento di quelle principali.

#### Procedure di autenticazione

Le procedure di autenticazione non identificano una risorsa ma sono i processi di login e logout che vengono utilizzati sia dal Cliente che dal GestoreRistorante per accedere e registrarsi al sistema. Tutto ciò è descritto dal Resource Model sotto indicato.

- **signup:** Utilizza il metodo POST prendendo in input un utente. Restituisce due boolean tramite il codice di risposta 201 - Created. Un boolean indica se l'utente è stato creato correttamente, mentre l'altro specifica se è un utente UniTN. Se esiste già un utente con le stesse credenziali, viene restituito il codice di risposta 409 - Conflict. In caso di errore del server durante la creazione dell'utente oppure parametri invalidi, viene restituito il codice di risposta 500 - Internal server error.
- **signout:** Utilizza il metodo POST prendendo in input un csrfToken. Restituisce tramite codice di risposta 302 - Found che l'utente ha effettuato il logout correttamente.
- **csrf:** Utilizza il metodo GET e restituisce un csrfToken. Viene anche restituito un messaggio di OK attraverso il codice 200 nel caso tutto sia andato a buon fine.
- **credential:** Utilizza il metodo POST prendendo in input un utente. Restituisce un messaggio di OK attraverso il codice 200 nel caso tutto sia andato a buon fine.
- **session:** Utilizza il metodo GET per recuperare la sessione dell'utente. Restituisce le informazioni essenziali della sessione dell'utente in un oggetto userSession e il codice di risposta 200 - OK per indicare che la richiesta è stata eseguita con successo.



## GestoreRistorante

La risorsa GestoreRistorante implementa le seguenti API, descritte attraverso il Resource Model sotto indicato:

- **updateRestaurant:** Utilizza il metodo PUT con input il Ristorante da modificare. Se tutto va a buon fine viene restituito un messaggio di avvenuta modifica attraverso il codice 200 - OK. Può succedere che ci siano degli errori:
  - 400 - Bad Request, nel caso in cui il gestore non abbia inserito l'id del ristorante;
  - 401 - Unauthorized, nel caso in cui il gestore non possa modificare quel ristorante;
  - 404 - NotFound, nel caso in cui l'id del ristorante inserito dal gestore non esiste.

Tutti gli errori vengono notificati all'utente attraverso opportuni messaggi di errore.

- **addRestaurant:** Utilizza il metodo POST con input il ristorante da aggiungere. Se tutto va a buon fine viene restituito un messaggio “aggiunto correttamente” attraverso il codice 200 - OK. Può succedere che ci siano degli errori:

- 400 - Bad Request, nel caso in cui il gestore non abbia inserito l'id del ristorante;
- 401 - Unauthorized, nel caso in cui un utente che non è un gestore prova a caricare un ristorante.

Tutti gli errori vengono notificati all'utente attraverso opportuni messaggi di errore.

- **getRestaurant:** Utilizza il metodo GET con parametro restaurantId e restituisce il ristorante con quell'id. Se tutto va a buon fine (codice 200 - OK) viene visualizzato il ristorante scelto. Può succedere che ci siano degli errori:

- 400 - Bad Request, nel caso in cui non si sia inserito l'id del ristorante;
- 404 - Not Found, nel caso in cui non esista un ristorante con quell'id.

Tutti gli errori vengono notificati all'utente attraverso opportuni messaggi di errore.

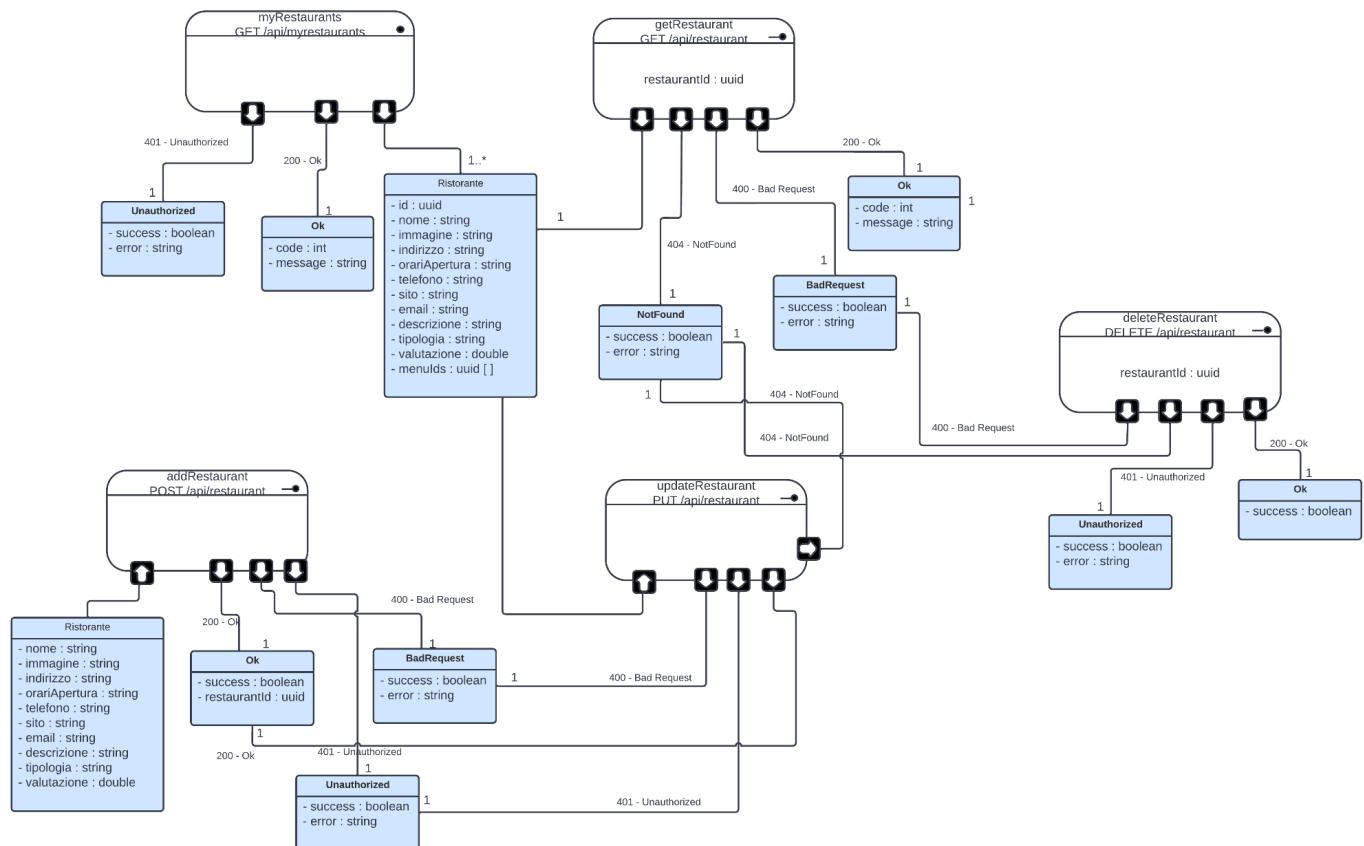
- **deleteRestaurant:** Utilizza il metodo DELETE con parametro ristorantId. Se tutto va a buon fine viene restituito un messaggio di avvenuta rimozione attraverso il codice 200 - OK. Può succedere che ci siano degli errori:

- 400 - Bad Request, nel caso in cui non si sia inserito l'id del ristorante;
- 401 - Unauthorized, nel caso in cui non possa rimuovere quel ristorante;

- 404 - Not Found, nel caso in cui l'id del ristorante inserito non esista.

Tutti gli errori vengono notificati all'utente attraverso opportuni messaggi di errore.

- **myRestaurants**: utilizza il metodo GET. Se tutto va a buon fine (codice 200 - OK) viene restituita la lista di tutti i ristoranti. Viene restituito, con codice 401 - Unauthorized, un errore quando un utente che non è autenticato come gestore richiede i propri ristoranti.



## Ristorante

La risorsa Ristorante implementa le seguenti API, descritte attraverso il Resource Model sotto indicato:

- **getMenu:** Utilizza il metodo GET con parametro menuId e ritorna il menù corrispondente se tutto va a buon fine (codice 200 - OK). Può succedere che ci siano degli errori:

- 400 - Bad Request, nel caso in cui non si sia inserito l'id del menù;
- 404 - Not Found, nel caso in cui non venga trovato quel menù.

Tutti gli errori vengono notificati all'utente attraverso opportuni messaggi di errore.

- **getThumbnails:** Utilizza il metodo GET con parametri "start" e "num", ritorna "num" ristoranti partendo da "start". Ogni ristorante contiene solo alcune informazioni che vengono visualizzate nella pagina di tutti i ristoranti ([5.13](#))

- **addMenu:** Utilizza il metodo POST con input il menù da aggiungere. Se tutto va a buon fine viene restituito un messaggio "aggiunto correttamente" attraverso il codice 200 - OK. Può succedere che ci siano degli errori:

- 400 - Bad Request, nel caso in cui non ci siano tutti i parametri;
- 401 - Unauthorized, nel caso in cui l'utente non sia registrato come gestore o non abbia il ristorante a cui appartiene quel menù.

Tutti gli errori vengono notificati all'utente attraverso opportuni messaggi di errore.

- **updateMenu:** Utilizza il metodo PUT con input il menù da modificare. Se tutto va a buon fine viene restituito un

messaggio di avvenuta modifica attraverso il codice 200 - OK.  
Può succedere che ci siano degli errori:

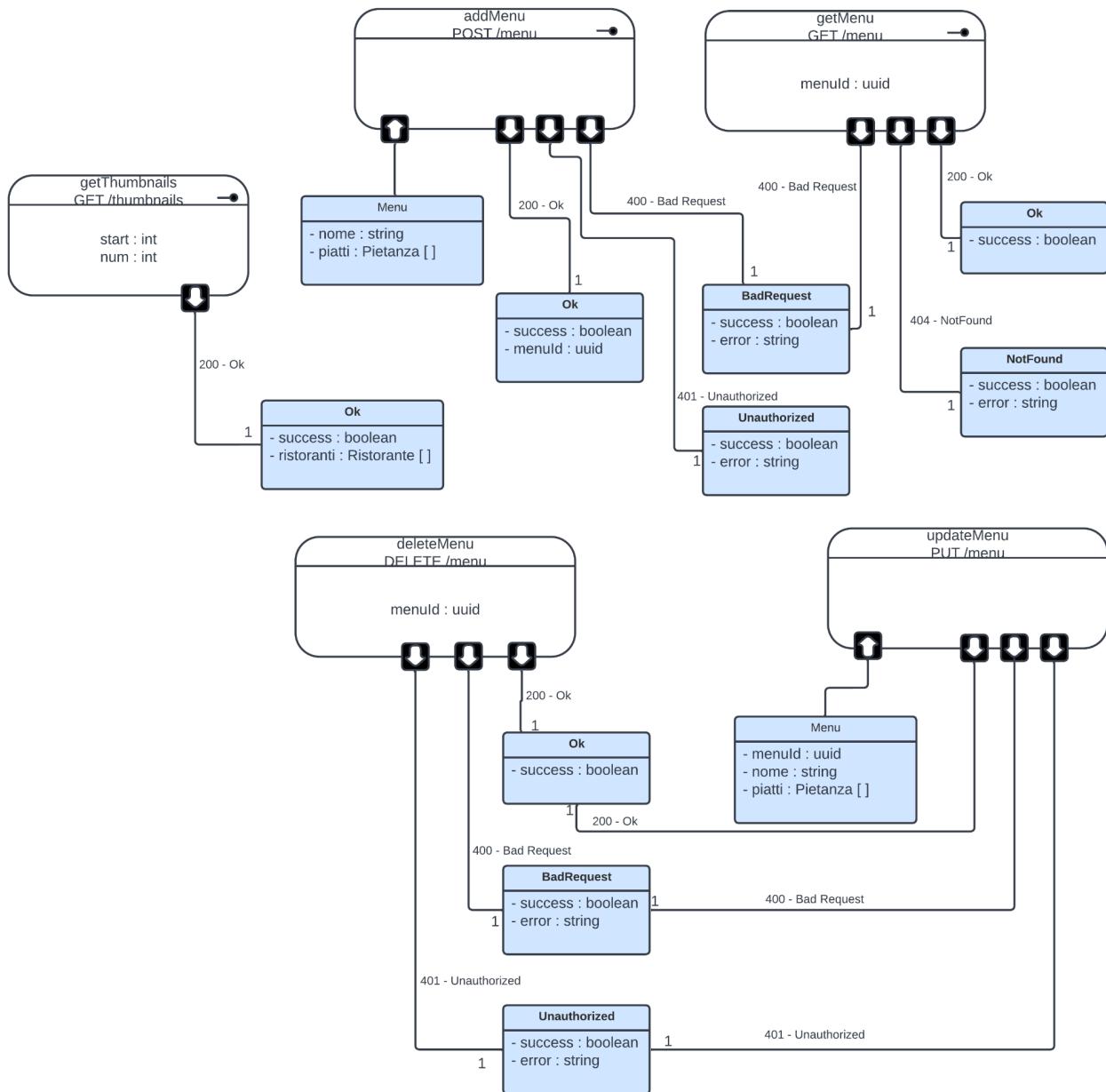
- 400 - Bad Request, nel caso in cui non vengano inseriti tutti i parametri;
- 401 - Unauthorized, nel caso in cui l'utente non sia registrato come gestore o non abbia il ristorante a cui appartiene quel menù.

Tutti gli errori vengono notificati all'utente attraverso opportuni messaggi di errore.

- **deleteMenu:** Utilizza il metodo DELETE con parametro menuld. Se tutto va a buon fine viene restituito un messaggio di avvenuta rimozione attraverso il codice 200 - OK. Può succedere che ci siano degli errori:

- 400 - Bad Request, nel caso in cui non vengano inseriti tutti i parametri;
- 401 - Unauthorized, nel caso in cui l'utente non sia registrato come gestore o non abbia il ristorante a cui appartiene quel menù.

Tutti gli errori vengono notificati all'utente attraverso opportuni messaggi di errore.



## Carrello

La risorsa Carrello implementa le seguenti API, descritte attraverso il Resource Model sotto indicato:

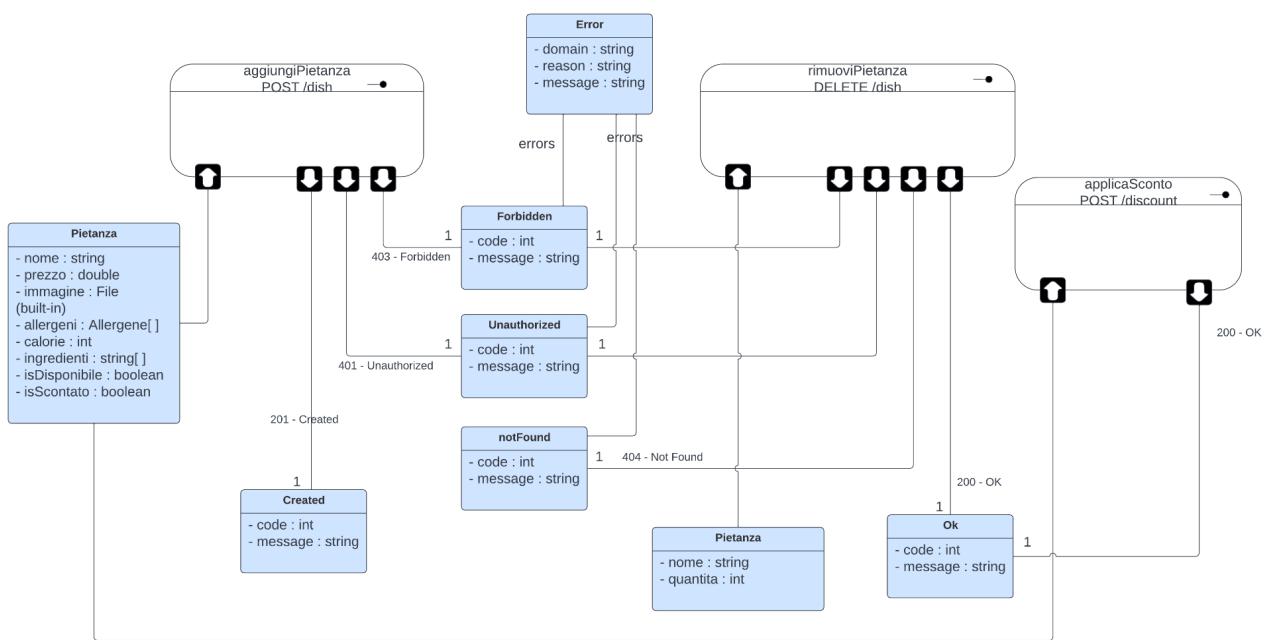
- **aggiungiPietanza:** Utilizza il metodo POST con input la pietanza da aggiungere. Se tutto va a buon fine viene aggiunta la pietanza al carrello e viene notificato attraverso il codice 201 - Created con relativo messaggio. Può succedere che ci siano degli errori:
  - 401 - Unauthorized, nel caso in cui l'utente non abbia effettuato l'accesso e provi ad aggiungere la pietanza;
  - 403 - Forbidden, nel caso in cui all'utente non sia permesso di aggiungere la pietanza;

Tutti gli errori vengono notificati all'utente attraverso opportuni messaggi di errore.

- **rimuoviPietanza:** Utilizza il metodo DELETE con input la pietanza da rimuovere e la quantità. Se tutto va a buon fine viene notificato all'utente la corretta rimozione della pietanza attraverso il codice 200 - OK. Può succedere che ci siano degli errori:
  - 401 - Unauthorized, nel caso in cui l'utente non possa rimuovere quella pietanza;
  - 403 - Forbidden, nel caso in cui all'utente non sia permessa la rimozione della pietanza;
  - 404 - Not Found, nel caso in cui si provi a rimuovere una pietanza inesistente.

Tutti gli errori vengono notificati all'utente attraverso opportuni messaggi di errore.

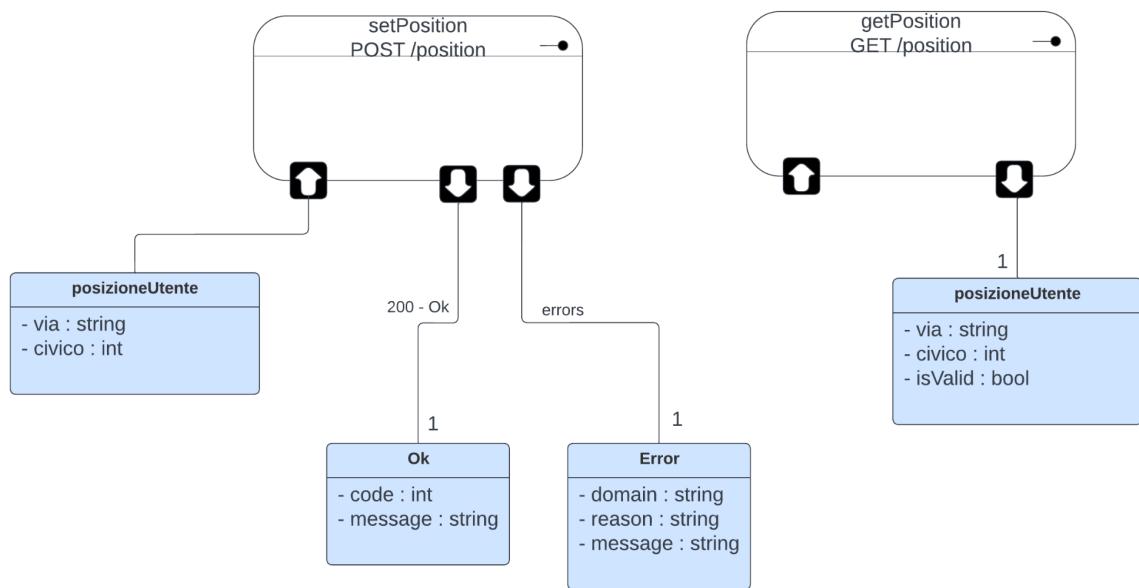
- **applicaSconto:** Utilizza il metodo POST con input la Pietanza da scontare. Se tutto va a buon fine viene notificato all'utente che lo sconto è stato applicato attraverso il codice 200 - OK.



## Posizione

La risorsa Posizione implementa le seguenti API, descritte attraverso il Resource Model sotto indicato:

- **setPosition**: Utilizza il metodo POST con input `posizioneUtente`. Se la via dell'utente esiste viene notificato all'utente la conferma della posizione con annesso codice 200 - Ok. Viene notificato un messaggio di errore nel caso la via non esista.
- **getPosition**: Utilizza il metodo GET e restituisce `posizioneUtente`.



## 3.5 Sviluppo API

Nel Resources Model sopra sono mostrate diverse API, ma solo quelle più importanti per il progetto in corso sono state sviluppate ed implementate per garantirne il corretto funzionamento.

### 3.5.1 Login

Il login viene gestito dal modulo “next-auth” che permette di fare il login tramite più metodi diversi, tra cui il metodo con credenziali che e’ quello che e’ stato implementato in questo caso.

```
CredentialsProvider({
  name: "credentials",
  credentials: {
    email: null,
    password: null,
  },
  async authorize(credentials) {
    const error = "Invalid email or password";
    await dbConnect();
    // check user existance
    const result = await UserSchema.findOne({ email: credentials.email });
    if (!result) {
      throw new Error(error);
    }
    const checkPassword = await compare(
      credentials.password,
      result.password
    );

    // incorrect password
    if (!checkPassword || result.email !== credentials.email) {
      throw new Error(error);
    }

    return result;
  },
})
```

### 3.5.2 Register

Questa API permette la registrazione di un nuovo utente: controlla innanzitutto che vengano rispettati i requisiti sulle stringhe come imposto in front-end, poi controlla che l'utente non sia già registrato e procede a salvare il nuovo utente nel database assieme all'hash della password per motivi di sicurezza.

```
async function signupHandler(req: NextApiRequest, res: NextApiResponse<Data>) {
  if (req.method !== "POST") {
    return res.status(405).send({
      success: false,
      error: "HTTP method not valid only POST Accepted",
    });
  }

  const schema = Yup.object().shape({
    name: regSchema.name,
    surname: regSchema.name,
    email: regSchema.email,
    password: regSchema.password,
  });

  try {
    await schema.validate(req.body);

    const { name, surname, email, password, isGestore } = req.body;
    await dbConnect();

    const hypoteticUser = await UserSchema.findOne({ email }).exec();
    if (hypoteticUser && hypoteticUser.email === email) {
      return res.status(409).send({
        success: false,
        error: "Email already in use",
      });
    }
    const isUnitn = email.endsWith("@studenti.unitn.it");

    await UserSchema.create({
      id: uuidv4(),
      name,
      surname,
      email,
      password: await hash(password, 12),
      isGestore,
      isUnitn,
    });

    return res.status(201).send({ success: true, isUnitn });
  } catch (error) {
    return res
      .status(500)
      .send({ success: false, error: (error as Error).message });
  }
}
```

### 3.5.3 Thumbnails

Questa API viene utilizzata per richiedere le informazioni base di una quantità (num) determinata alla richiesta di ristoranti, in modo da visualizzarle nella pagina che mostra la lista dei ristoranti registrati sul sito (/ristoranti).

```
async function thumbnailsHandler(
  req: NextApiRequest,
  res: NextApiResponse<Data>
) {
  if (req.method !== "GET") {
    return res.status(405).send({
      success: false,
      error: "HTTP method not valid only GET Accepted",
    });
  }
  const start = parseInt(req.query.start as string);
  const num = parseInt(req.query.num as string);

  dbConnect();
  const ristoranti = await RistoranteSchema.find({})
    .skip(start)
    .limit(num)
    .exec();
  const ristorantiData = ristoranti.map((ristorante) => {
    return {
      id: ristorante.id,
      nome: ristorante.nome,
      indirizzo: ristorante.indirizzo,
      telefono: ristorante.telefono,
      email: ristorante.email,
      immagine: ristorante.immagine,
      tipologia: ristorante.tipologia,
    };
  });

  return res.status(200).json({ success: true, ristoranti: ristorantiData });
}
```

### 3.5.4 My Restaurants

Questa API viene utilizzata quando un gestore vuole visualizzare tutti i suoi ristoranti, l'API verifica che sia eseguita da un gestore e poi ritorna tutti i ristoranti che hanno quel gestoreId.

```
async function myrestaurantHandler(
  req: NextApiRequest,
  res: NextApiResponse<Data>
) {
  if (req.method !== "GET") {
    return res.status(405).send({
      success: false,
      error: "HTTP method not valid only GET Accepted",
    });
  }

  const session = await unstable_getServerSession(req, res, authOptions);
  if (!session?.user?.isGestore) {
    return res.status(401).send({
      success: false,
      error: "Unauthorized",
    });
  }

  dbConnect();
  const ristoranti = await RistoranteSchema.find({
    gestoreId: session.user.id,
  }).exec();

  return res.status(200).json({ success: true, ristoranti });
}
```

### 3.5.5 Get Menu

Questa API serve a recuperare le informazioni e tutti i piatti di menù dato l'id.

```
async function getMenu(req: NextApiRequest, res: NextApiResponse<Data>) {  
  const { id } = req.query;  
  if (!id) {  
    return res.status(400).send({  
      success: false,  
      error: "id not provided",  
    });  
  }  
  
  dbConnect();  
  const menu = await MenuSchema.findOne({ id }).exec();  
  if (!menu) {  
    return res.status(404).send({  
      success: false,  
      error: "Menu not found",  
    });  
  }  
  
  return res.status(200).send({ success: true, menu });  
}
```

### 3.5.6 Add Menu

Questa API serve per aggiungere un menu a un dato ristorante. Prima di tutto vengono controllate le credenziali dell'utente che fa la richiesta per controllare che sia un gestore, poi viene controllata anche la presenza di tutti i parametri fondamentali per la creazione del menu, infine viene salvato il menu nel database e aggiunto l'id del nuovo menu al ristorante che lo possiede.

```
async function addMenu(req: NextApiRequest, res: NextApiResponse<Data>) {
  const session = await unstable_getServerSession(req, res, authOptions);
  if (!session?.user?.isGestore) {
    return res.status(401).send({
      success: false,
      error: "Unauthorized",
    });
  }
  const { nome, ristoranteId, piatti } = req.body;
  if (!nome || !ristoranteId || !piatti) {
    return res.status(400).send({
      success: false,
      error: "Insufficient arguments",
    });
  }

  dbConnect();
  try {
    const piattiData = piatti.map((piatto: Omit<PietanzaDocType, "id">) => ({
      id: uuidv4(),
      ...piatto,
    }));
    const menu = await MenuSchema.create({
      id: uuidv4(),
      nome,
      ristoranteId,
      piatti: piattiData,
    });
    await RistoranteSchema.updateOne(
      { id: ristoranteId },
      { $push: { menuIds: menu.id } }
    ).exec();
    return res.status(200).send({ success: true, menuItem: menu.id });
  } catch (error) {
    return res.status(400).send({
      success: false,
      error: error.message,
    });
  }
}
```

### 3.5.7 Edit Menu

Questa API serve per modificare un menu di un ristorante che si possiede. Bisogna fornire l'id del menu e i campi che si desidera modificare: prima di tutto vengono controllate le credenziali dell'utente che fa la richiesta per controllare che sia il gestore proprietario del ristorante a cui appartiene il menù, in seguito i campi forniti verranno sovrascritti a quelli già presenti nel database, verrà poi restituito il menu modificato.

```
async function editMenu(req: NextApiRequest, res: NextApiResponse<Data>) {
  const session = await unstable_getServerSession(req, res, authOptions);
  const { id, nome, piatti } = req.body;
  if (!id) {
    return res.status(400).send({
      success: false,
      error: "id is required",
    });
  }

  await checkPermission(session, id, res);

  dbConnect();
  const menu = await MenuSchema.findOne({ id }).exec();
  if (nome) menu.nome = nome;
  if (piatti) menu.piatti = piatti;
  await menu.save();

  return res.status(200).send({ success: true, menu });
}
```

### 3.5.8 Delete Menu

Questa API permette di eliminare un menu di un ristorante. Fornito l'id del menù da eliminare, prima di tutto vengono controllate le credenziali dell'utente che fa la richiesta per controllare che sia il gestore proprietario del ristorante a cui appartiene il menù, in seguito il menù verrà eliminato dal database e verrà anche eliminato il suo id dalla lista menulds del ristorante a cui appartiene.

```
async function deleteMenu(req: NextApiRequest, res: NextApiResponse<Data>) {
  const session = await unstable_getServerSession(req, res, authOptions);
  const id = req.query.id as string;
  if (!id) {
    return res.status(400).send({
      success: false,
      error: "id is required",
    });
  }

  await checkPermission(session, id, res);

  dbConnect();
  await MenuSchema.deleteOne({ id }).exec();
  await RistoranteSchema.updateOne(
    { gestoreId: session.user.id },
    { $pull: { menuIds: id } }
  ).exec();
  return res.status(200).send({ success: true });
}
```

### 3.5.9 Get Restaurant Info

Questa API serve a recuperare tutte le informazioni di un ristorante dato il suo id.

```
async function getRestaurantInfo(
  req: NextApiRequest,
  res: NextApiResponse<Data>
) {
  const { id } = req.query;
  if (!id) {
    return res.status(400).send({
      success: false,
      error: "Missing id",
    });
  }

  dbConnect();
  const restaurant = await RistoranteSchema.findOne({ id: id });
  if (!restaurant) {
    return res.status(404).send({
      success: false,
      error: "Restaurant not found",
    });
  }

  return res.status(200).send({ success: true, ristorante: restaurant });
}
```

### 3.5.10 Add Restaurant

Questa API serve per creare un nuovo ristorante. Prima di tutto vengono controllate le credenziali dell'utente che fa la richiesta per controllare che sia un gestore, poi viene controllata anche la presenza di tutti i parametri fondamentali per la creazione del ristorante, infine viene salvato il ristorante nel database.

```
async function addRestaurant(req: NextApiRequest, res: NextApiResponse<Data>) {
  const session = await unstable_getServerSession(req, res, authOptions);
  if (!session?.user?.isGestore) {
    return res.status(401).send({
      success: false,
      error: "Unauthorized",
    });
  }

  const {
    nome,
    indirizzo,
    orariApertura,
    telefono,
    sito,
    email,
    descrizione,
    tipologia,
    immagine,
    valutazione,
  } = req.body;
  // [NOTE] should use YUP
  if (
    !nome ||
    !indirizzo ||
    !orariApertura ||
    !telefono ||
    !sito ||
    !email ||
    !descrizione ||
    !tipologia ||
    !immagine
  ) {
    return res.status(400).send({
      success: false,
      error: "Missing fields",
    });
  }
}

dbConnect();
try {
  const restaurant = await RistoranteSchema.create({
    id: v4(),
    nome,
    indirizzo,
    orariApertura,
    telefono,
    sito,
    email,
    descrizione,
    tipologia,
    gestoreId: session.user.id,
    immagine,
    valutazione: valutazione ? valutazione : 0,
    menuIds: [],
  });
  return res.status(200).send({ success: true, ristoranteId: restaurant.id });
} catch (error) {
  return res.status(400).send({
    success: false,
    error: error.message,
  });
}
}
```

### 3.5.11 Edit Restaurant

Questa API serve per modificare un ristorante che si possiede. Bisogna fornire l'id del ristorante e i campi che si desidera modificare: prima di tutto vengono controllate le credenziali dell'utente che fa la richiesta per controllare che sia il gestore proprietario del ristorante in questione, in seguito i campi forniti verranno sovrascritti a quelli già presenti nel database, verrà poi restituito il ristorante modificato.

```
async function editRestaurant(req: NextApiRequest, res: NextApiResponse<Data>) {
  const session = await unstable_getServerSession(req, res, authOptions);
  const {
    id,
    nome,
    indirizzo,
    orariApertura,
    telefono,
    sito,
    email,
    descrizione,
    tipologia,
    immagine,
    valutazione,
  } = req.body;
  if (!session?.user?.isGestore) {
    return res.status(401).send({
      success: false,
      error: "Unauthorized",
    });
  }
  if (!id) {
    return res.status(400).send({
      success: false,
      error: "id is required",
    });
  }
  dbConnect();
  const restaurant = await RistoranteSchema.findOne({
    id,
    gestoreId: session.user.id,
  });
  if (!restaurant) {
    return res.status(404).send({
      success: false,
      error: "Restaurant not found",
    });
  }
  if (nome) restaurant.nome = nome;
  if (indirizzo) restaurant.indirizzo = indirizzo;
  if (orariApertura) restaurant.orariApertura = orariApertura;
  if (telefono) restaurant.telefono = telefono;
  if (sito) restaurant.sito = sito;
  if (email) restaurant.email = email;
  if (descrizione) restaurant.descrizione = descrizione;
  if (tipologia) restaurant.tipologia = tipologia;
  if (valutazione) restaurant.valutazione = valutazione;
  if (immagine) restaurant.immagine = immagine;
  await restaurant.save();

  return res.status(200).send({ success: true, ristorante: restaurant });
}
```

### 3.5.12 Delete Restaurant

Questa API permette di eliminare un ristorante. Fornito l'id del ristorante da eliminare, prima di tutto vengono controllate le credenziali dell'utente che fa la richiesta per controllare che sia il gestore proprietario del ristorante in questione, in seguito il ristorante verrà eliminato dal database.

```
async function deleteRestaurant(
  req: NextApiRequest,
  res: NextApiResponse<Data>
) {
  const session = await unstable_getServerSession(req, res, authOptions);
  const id = req.query.id as string;
  if (!session?.user?.isGestore) {
    return res.status(401).send({
      success: false,
      error: "Unauthorized",
    });
  }

  if (!id) {
    return res.status(400).send({
      success: false,
      error: "id is required",
    });
  }
}

dbConnect();
const restaurant = await RistoranteSchema.findOne({
  id,
  gestoreId: session.user.id,
});
if (!restaurant) {
  return res.status(404).send({
    success: false,
    error: "Restaurant not found",
  });
}
await MenuSchema.deleteMany({ id: { $in: restaurant.menuIds } });
await restaurant.delete();

return res.status(200).send({ success: true });
}
```

## 4. API documentation

Le API Locali fornite dall'applicazione Hungry Everywhere e descritte nella sezione precedente sono state documentate utilizzando il modulo NodeJS chiamato Swagger UI React. In questo modo la documentazione relativa alle API è direttamente disponibile a chiunque veda il codice sorgente. Per poter generare l'endpoint dedicato alla presentazione delle API è stato utilizzato Swagger UI in quanto crea una pagina web dalle definizioni delle specifiche OpenAPI.

In particolare, di seguito viene mostrata la pagina web relativa alla documentazione che presenta le 11 API (GET, POST, PUT e DELETE) per la gestione dei dati dell'applicazione. Le API di tipo GET vengono generalmente utilizzate per richiedere dati al backend dell'applicazione, le API POST vengono usate per inserire dati all'interno dell'applicazione web (generalmente nel database), le API PUT vengono usate per modificare dati già esistenti e, infine, le API di tipo DELETE vengono usate per eliminare dati dal database.

L'endpoint da invocare per raggiungere la seguente documentazione è: <http://localhost:3000/api-doc> se il sito è avviato in locale, altrimenti è reperibile su [Vercel](#).

The screenshot shows the Swagger UI interface for the Hungry Everywhere API. At the top, there's a red header bar with the API Doc logo and a Sign in button. Below the header, the title "Hungry Everywhere API Documentation" is displayed, along with a "1.0" version indicator and a green "OpenAPI" link. The main content area is organized into three sections:

- Auth**: Authentication endpoints. This section contains five API entries:
  - POST /api/auth/signout**: Log out.
  - GET /api/auth/csrf**: Get CSRF token.
  - POST /api/auth/callback/credentials**: Log in with credentials.
  - GET /api/auth/session**: Get session.
  - POST /api/auth/signup**: Register a new user.
- Menu**: Menu management. This section contains four API entries:
  - GET /api/menu**: Get a menu.
  - POST /api/menu**: Add a menu.
  - PUT /api/menu**: Update a menu.
  - DELETE /api/menu**: Delete a menu.
- Restaurant**: Restaurant management. This section contains two API entries:
  - GET /api/restaurant**: Get the info of a restaurant.
  - POST /api/restaurant**: Add a new restaurant.

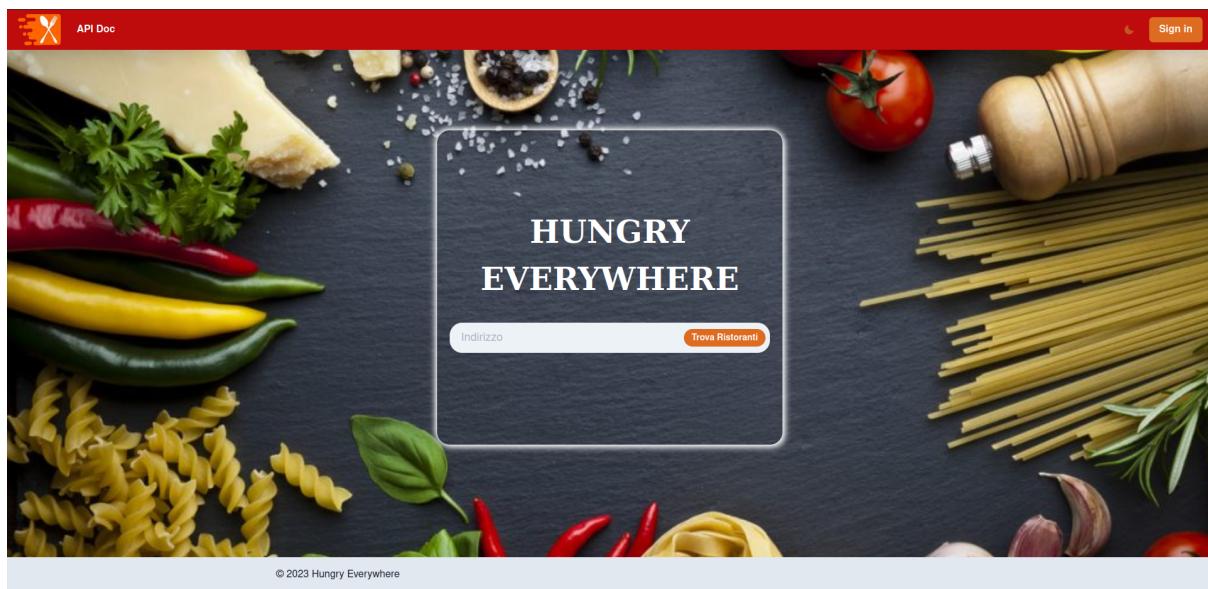
## 5. FrontEnd Implementation

Il FrontEnd offre diverse funzionalità in base all'utente che ne fa uso. Ulteriori dettagli saranno forniti nel corso del seguente capitolo.

### 5.1 - HomePage

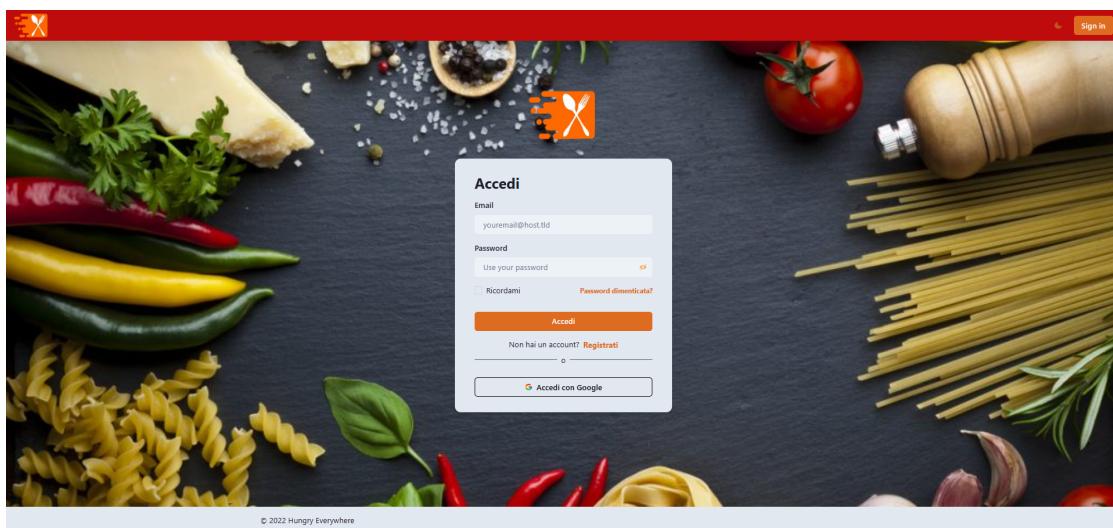
La schermata principale include una NavBar, presente su tutte le pagine, che contiene il logo che porta alla homepage, il pulsante per cambiare tema (chiaro o scuro) e il tasto di login. Il tasto di cambio tema non è implementato bene in tutte le pagine. Inoltre, vi è un footer con il nome dell'applicazione.

La schermata principale include anche un campo per inserire l'indirizzo e un pulsante per cercare i ristoranti nella zona specificata. È presente anche un tasto di login che permette di accedere alla pagina di accesso.



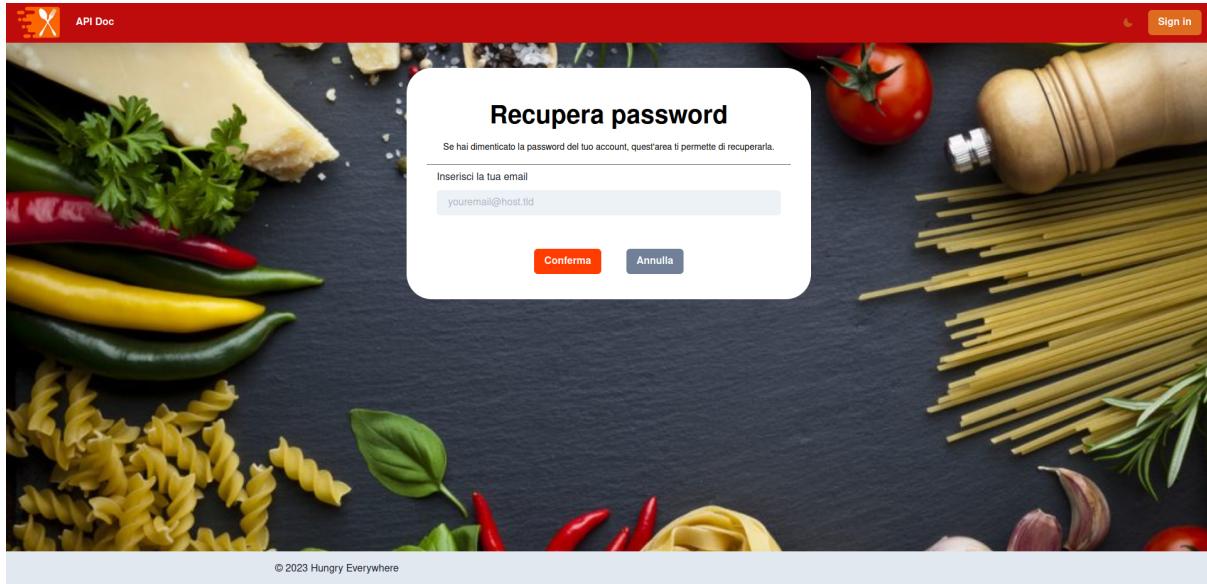
## 5.2 - Pagina di Login

La schermata di login include un form in cui è possibile inserire l'email e la password (con un pulsante per visualizzare la password inserita). C'è anche un tasto per salvare le credenziali inserite e un link accanto che porta alla pagina di recupero password. Il tasto "accedi" verifica che le credenziali inserite siano corrette. Inoltre, c'è un link per accedere alla pagina di registrazione e un pulsante per accedere con il proprio account Google.



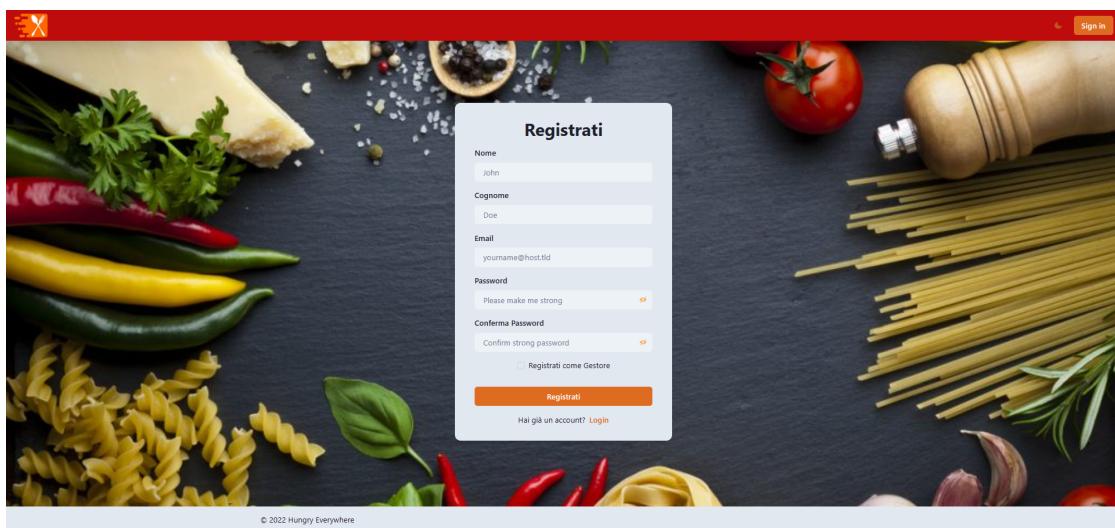
## 5.3 - Recupero password

La schermata di recupero password include un form in cui si inserisce l'email utilizzata durante la registrazione. Premendo il tasto "Conferma" verrà inviata una email all'utente. Premendo il tasto "Annulla" si torna alla homepage.



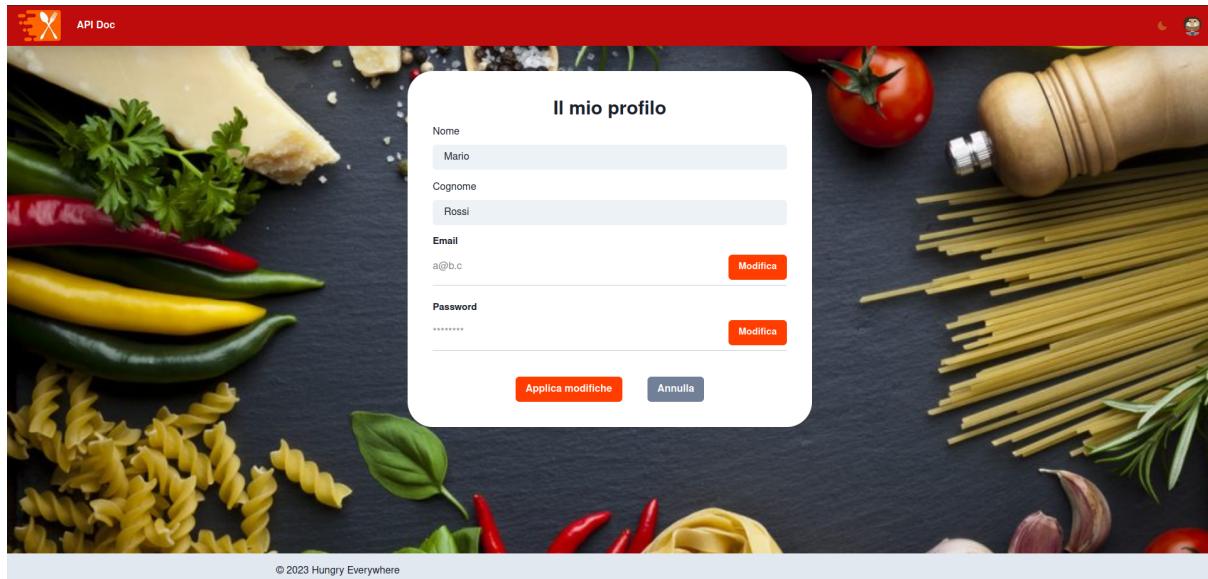
## 5.4 - Registrati

La schermata di registrazione include un form dove è possibile inserire il proprio nome, cognome, email e password. Ci sono anche campi di conferma password per aumentare la sicurezza e per evitare errori di battitura. I campi per inserire la password hanno pulsanti per mostrare la password inserita. Inoltre, c'è una casella di controllo per registrarsi come gestore. Premendo il pulsante "Registrati" ci si registra al sito. In basso c'è un link per tornare alla pagina di login se si ha già un account.



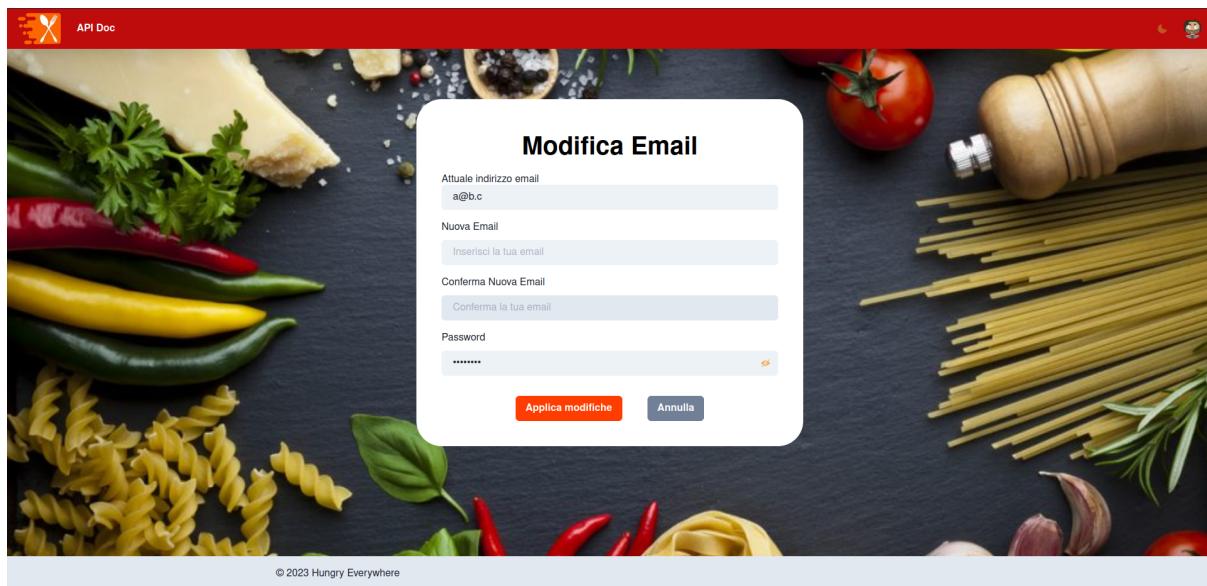
## 5.5 - Il mio profilo

La schermata di riepilogo del profilo mostra i campi nome e cognome che possono essere modificati direttamente dall'utente. Ci sono anche i campi email e password, con un pulsante "Modifica" che reindirizza alla pagina di modifica selezionata. Nella parte inferiore del form ci sono i pulsanti per applicare o annullare le modifiche.



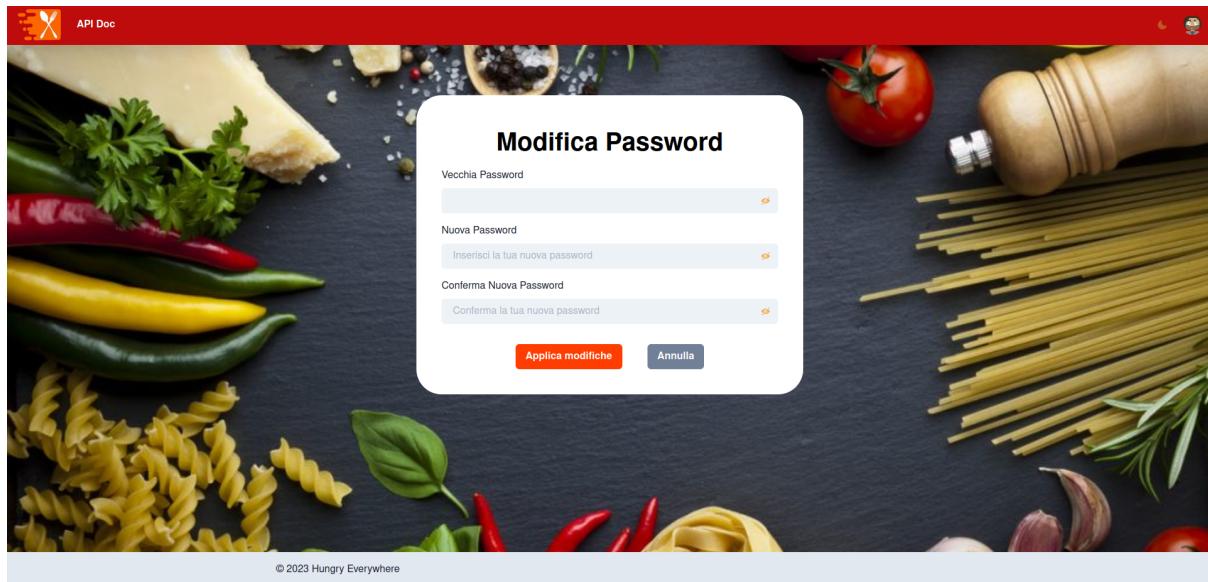
## 5.6 - Modifica email

La schermata di modifica email include un form con la propria email e i campi per modificarla. Per confermare le modifiche, è necessario inserire la password. Nella parte inferiore del form ci sono i pulsanti per applicare o annullare le modifiche.



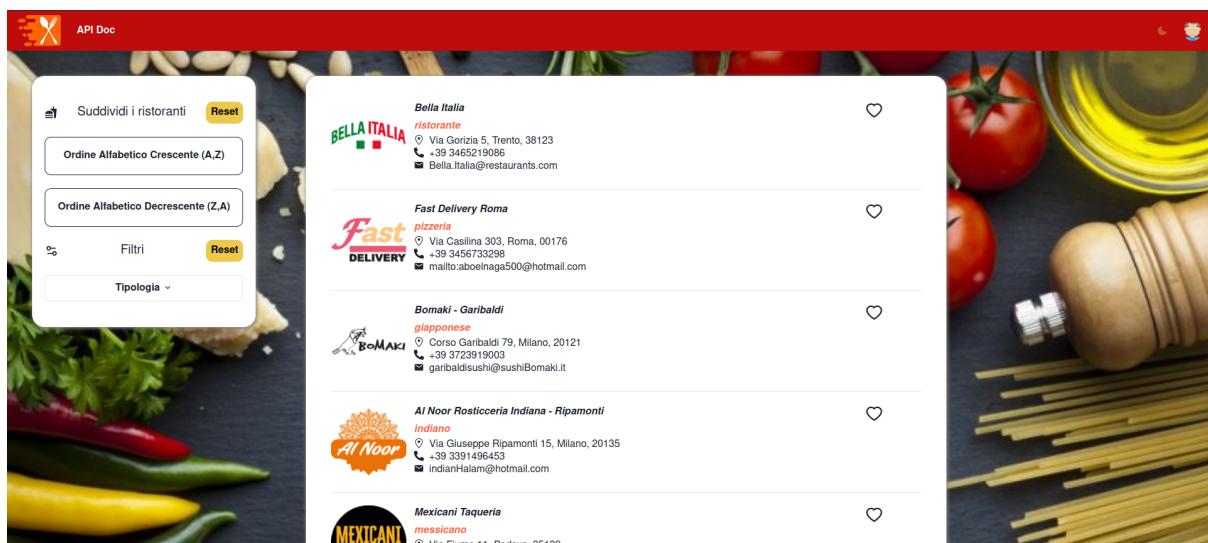
## 5.7 - Modifica password

La schermata di modifica password include un form con i campi per inserire la vecchia password, la nuova password e la conferma di quest'ultima. Nella parte inferiore del form ci sono i pulsanti per applicare o annullare le modifiche.



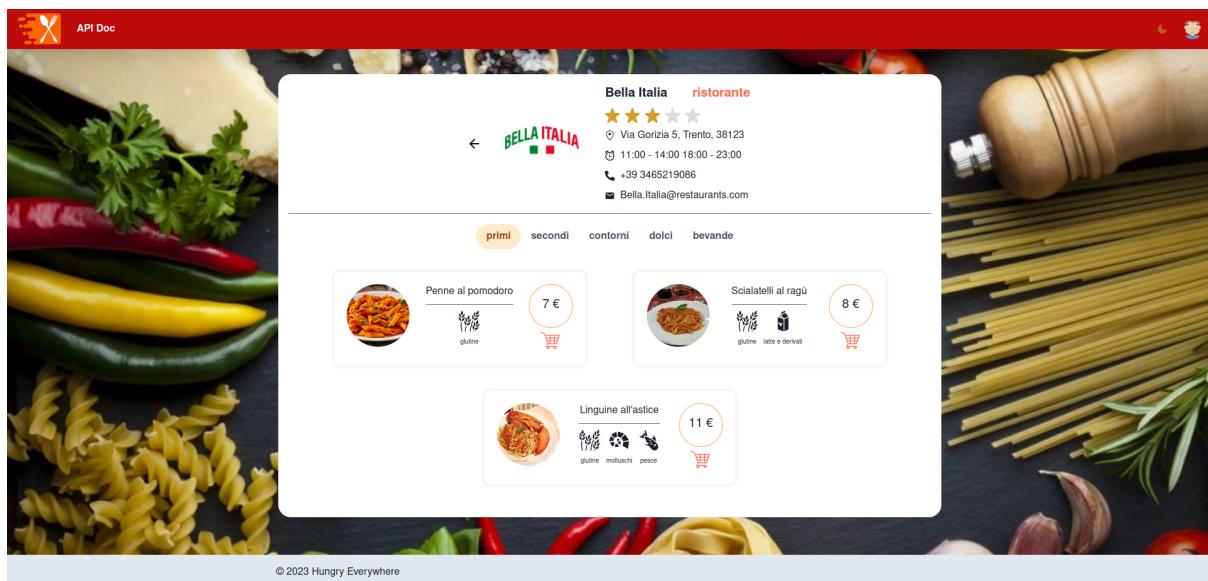
## 5.8 - Ristoranti vicini (cliente)

La schermata è divisa in due parti: la parte sinistra contiene due pulsanti per ordinare i ristoranti in ordine alfabetico od ordinarli tramite un menù a tendina in base alla topologia. C'è anche un tasto reset per eliminare i filtri selezionati. Al centro sono presenti i ristoranti con il nome, la tipologia e le informazioni. Per ogni ristorante, in alto a destra, è presente un pulsante per aggiungere il ristorante ai preferiti.



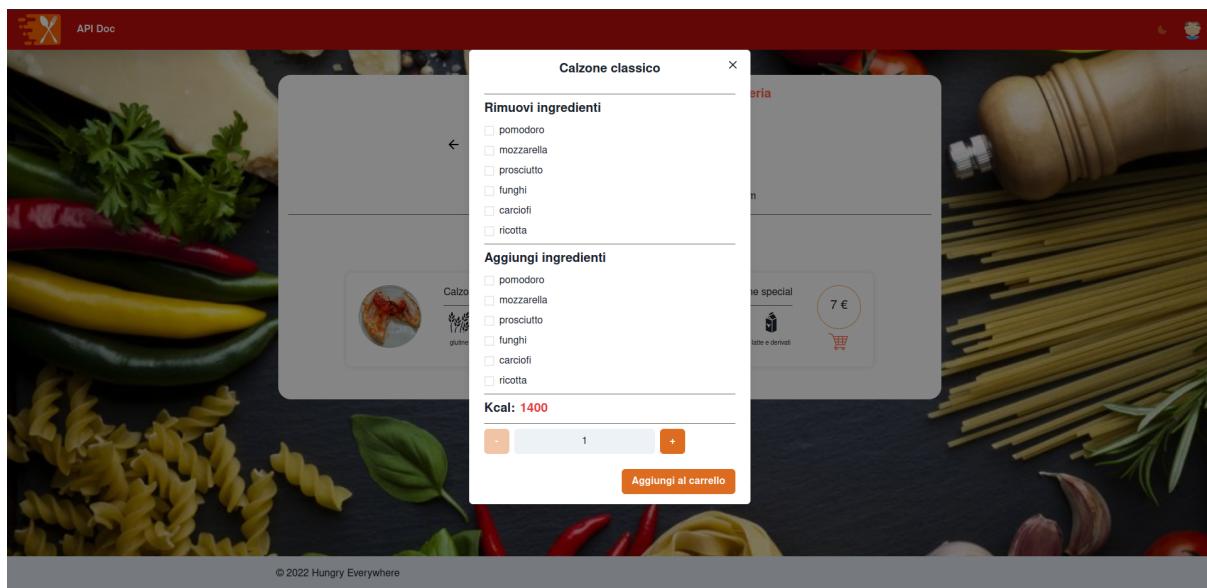
## 5.9 - Dettagli ristorante (cliente)

La schermata del ristorante presenta informazioni dettagliate sul ristorante, come il nome, la tipologia di cucina, la valutazione, l'indirizzo, gli orari di apertura, il numero di telefono e l'indirizzo email, oltre a una foto del ristorante. Nella seconda metà della pagina è presente il menù, suddiviso in diverse sezioni. Ogni sezione include diverse pietanze, complete di immagine, nome, nome, allergeni e prezzo.



## 5.10 - Focus pietanza

La schermata che si apre dopo aver premuto il pulsante carrello nella pagina del singolo ristorante include il nome della pietanza selezionata, gli ingredienti rimovibili e quelli aggiungibili. Inoltre, vi sono le calorie e un campo di input per scegliere la quantità desiderata. Il pulsante "Aggiungi al carrello" aggiunge la pietanza al carrello.

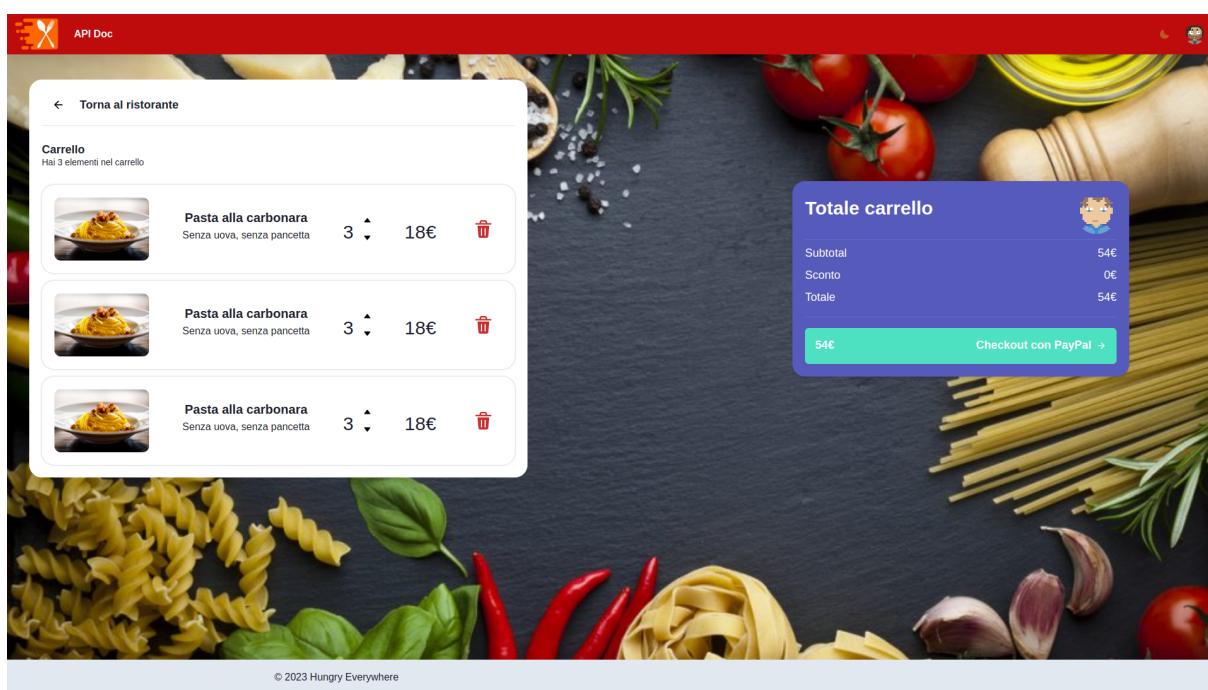


## 5.11 - Carrello

La schermata del carrello è suddivisa in due parti.

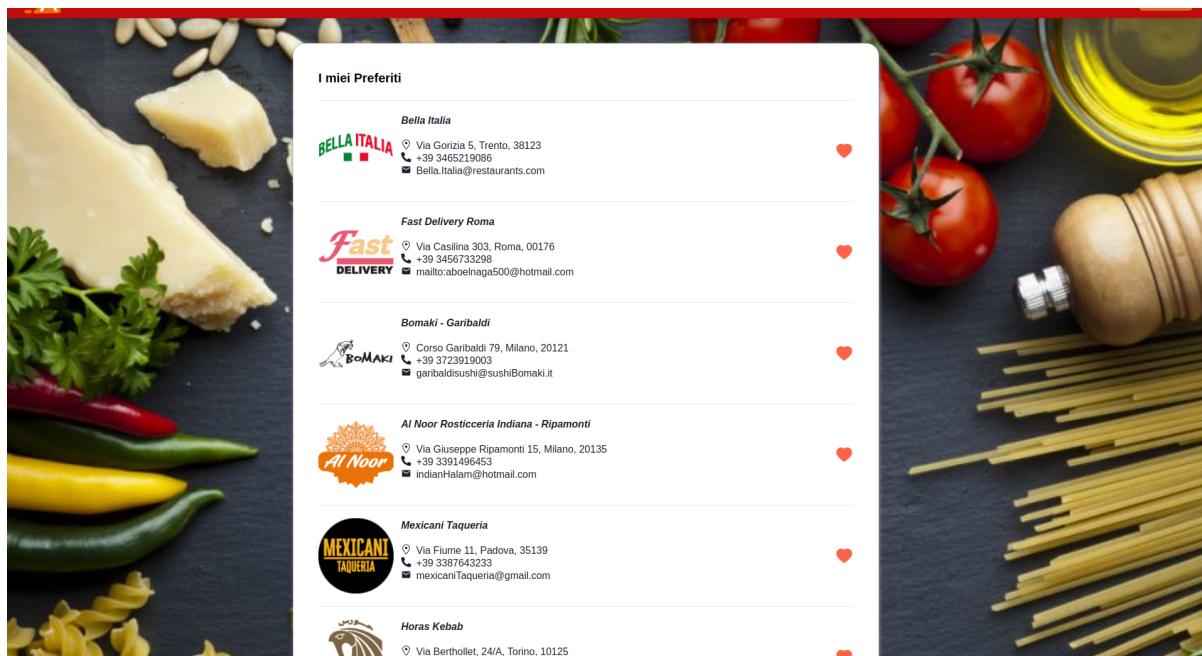
A sinistra c'è il riepilogo del carrello, dove è presente un pulsante per tornare al ristorante, un indicatore del numero di elementi nel carrello (ogni piatto viene considerato come un elemento) e le pietanze selezionate in precedenza. Ogni pietanza ha il nome e ulteriori dettagli, oltre a una quantità modificabile (minimo 1), il costo totale e un cestino per eliminarla.

A destra c'è una sezione che riepiloga il totale del carrello, comprensivo di subtotale, sconti applicati e totale finale. In fondo c'è un pulsante per pagare con PayPal.



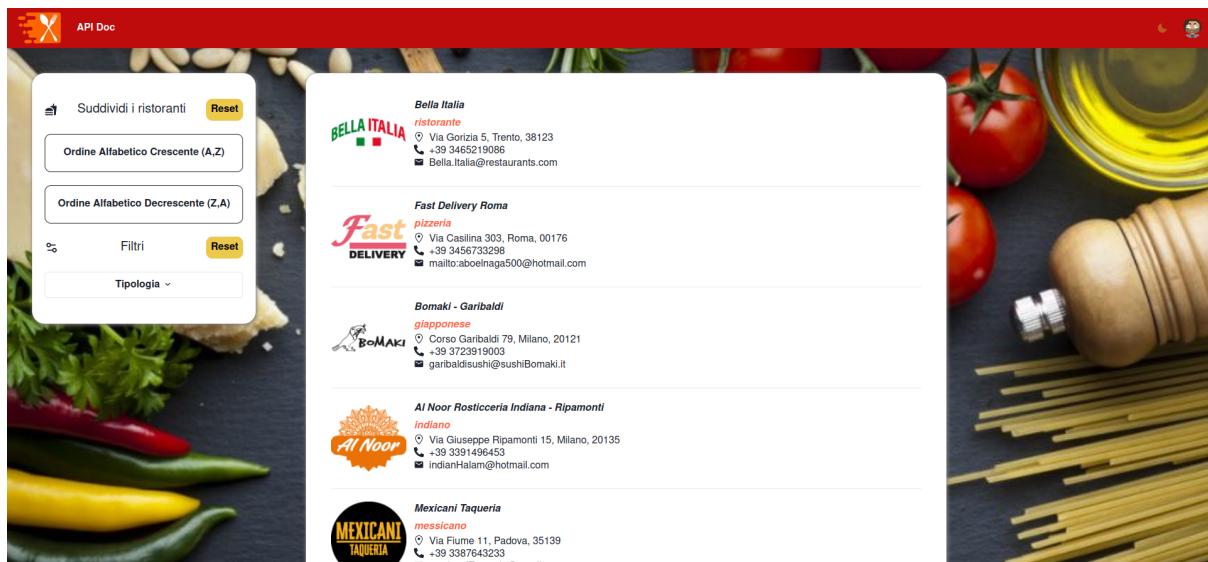
## 5.12 - I miei preferiti

La schermata dei ristoranti preferiti mostra all'utente autenticato l'elenco dei ristoranti a cui ha assegnato la preferenza, consentendo un accesso più rapido a un ristorante specifico. È possibile rimuovere uno o più ristoranti dall'elenco dei preferiti cliccando sull'icona del cuore accanto a ogni ristorante.



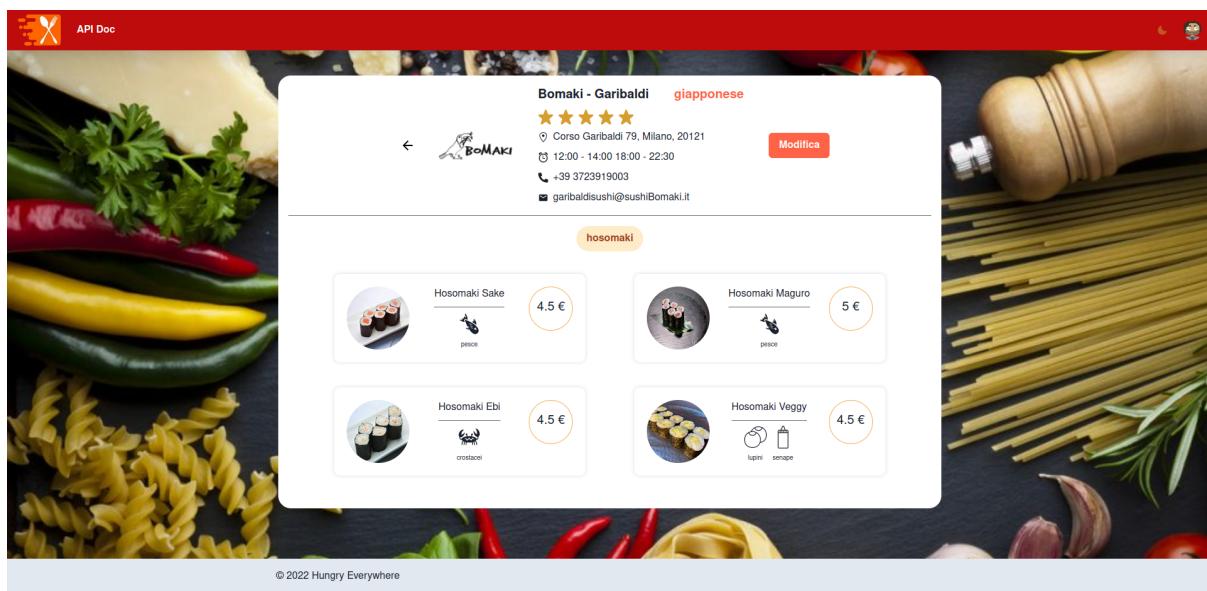
## 5.13 - Ristoranti vicini

La schermata è divisa in due parti: la parte sinistra contiene due pulsanti per ordinare i ristoranti in ordine alfabetico od ordinarli tramite un menù a tendina in base alla topologia. C'è anche un tasto reset per eliminare i filtri selezionati. Al centro sono presenti i ristoranti con il nome, la tipologia, l'immagine e le informazioni.



## 5.14 - Dettagli ristorante (gestore)

La schermata del ristorante presenta informazioni dettagliate sul ristorante, come il nome, la tipologia di cucina, la valutazione, l'indirizzo, gli orari di apertura, il numero di telefono e l'indirizzo email, oltre a una foto del ristorante. Nella seconda metà della pagina è presente il menù, suddiviso in diverse sezioni. Ogni sezione include diverse pietanze, complete di immagine, nome, allergeni e prezzo. Il gestore ha a disposizione un pulsante "Modifica" che gli consente di apportare modifiche alle informazioni relative a un determinato ristorante.



## 5.15 - Schermata dei ristoranti del gestore

La schermata dei ristoranti del gestore visualizza un elenco di tutti i ristoranti che un gestore gestisce, complete di nome, tipologia di cucina, immagine, l'indirizzo, il numero di telefono e l'indirizzo email. È possibile rimuovere uno o più ristoranti dall'elenco cliccando sull'icona del cestino accanto a ogni ristorante. Il gestore può anche aggiungere nuovi ristoranti all'elenco utilizzando l'icona apposita presente alla fine della schermata.

**I miei Ristoranti**

<b>BELLA ITALIA</b> ristorante Bella Italia Via Gorizia 5, Trento, 38123 +39 3465219086 Bella.Italia@restaurants.com	
<b>Fast Delivery Roma</b> pizzeria Fast DELIVERY Via Casilina 303, Roma, 00176 +39 3456733298 mailto:aboehnaga500@hotmail.com	
<b>Bomaki - Garibaldi</b> giapponese BoMAKI Corso Garibaldi 79, Milano, 20121 +39 3723919003 garibaldisushi@sushiBomaki.it	
<b>Al Noor Rosticceria Indiana - Ripamonti</b> indiano Al Noor Via Giuseppe Ripamonti 15, Milano, 20135 +39 3391496453 indianHalham@hotmail.com	
<b>MEXICANI TAQUERIA</b> messicano MEXICANI TAQUERIA Via Fiume 11, Padova, 35139 +39 338764233 mexicanTaqueria@gmail.com	
<b>Horas Kebab</b> kebab HORAS Via Berthollet, 24/A, Torino, 10125 +39 3519987633 HorasKebab@yahoo.it	
<b>ZEN</b> poke Zen Poke Corso Padova 55, Vicenza, 36100 +39 3998765335 ZenpokeVI@pokesushi.it	
<b>MARK BURGER</b> fast food MARK BURGER Via Arenaccia 205, Napoli, 80141 +39 3490023343 markburgernapoli@gmail.com	

**+**

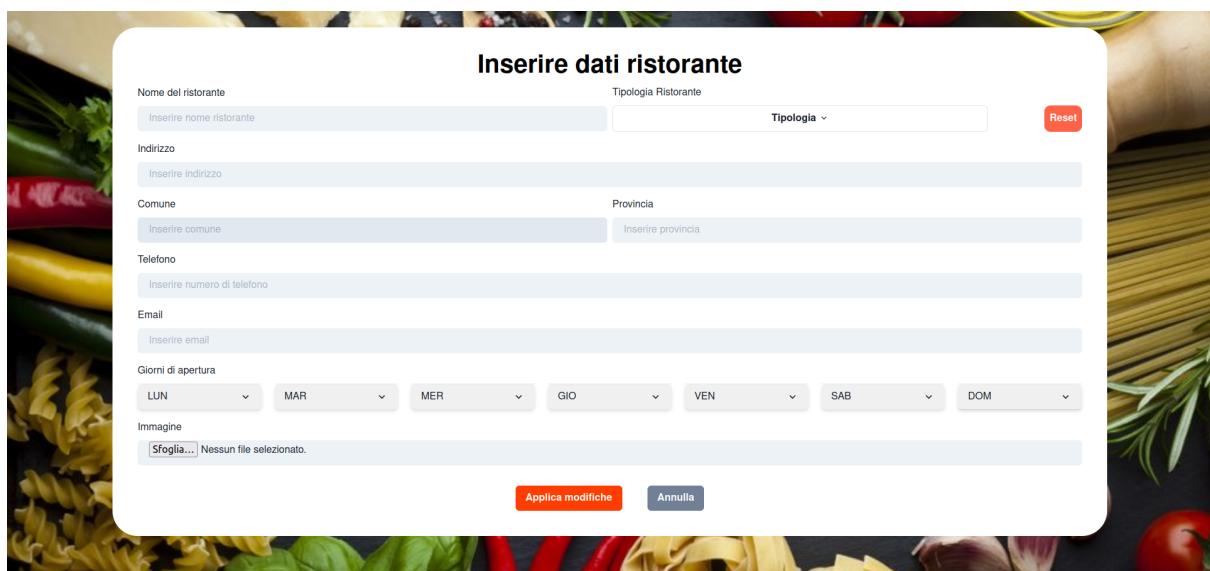
## 5.16 - Elimina ristorante

La schermata che si apre dopo aver premuto sul cestino chiede al gestore se è sicuro di voler eliminare il ristorante corrispondente. La finestra di dialogo presenta due opzioni: "Elimina" e "Annulla". Se il gestore seleziona "Elimina", l'oggetto viene eliminato. Se viene selezionata "Annulla", l'oggetto non viene eliminato.



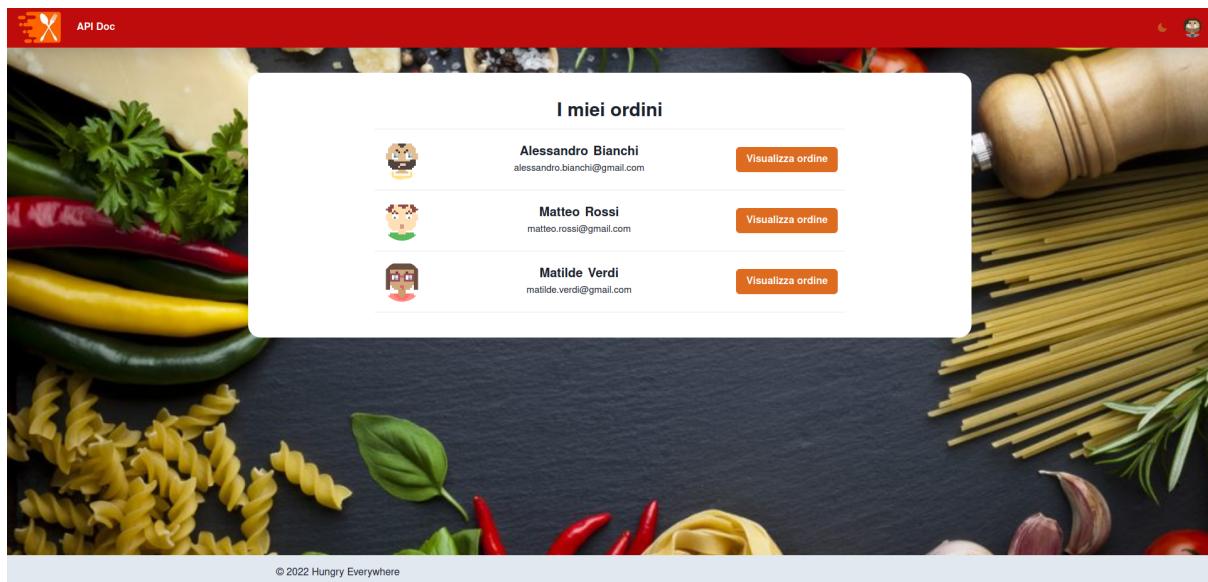
## 5.17 - Schermata di aggiunta/modifica ristorante

Quando il gestore seleziona il pulsante "Modifica" presente nella sezione [5.14](#) o il pulsante "+" presente nella sezione [5.15](#), viene visualizzata una nuova schermata contenente un form da compilare con le informazioni del ristorante. Nel form, è possibile selezionare la tipologia del ristorante tramite una tendina e specificare gli orari di apertura e chiusura per ogni giorno della settimana. È inoltre possibile caricare un'immagine del ristorante. In fondo al form, ci sono due pulsanti: uno per confermare le modifiche apportate e uno per annullarle.



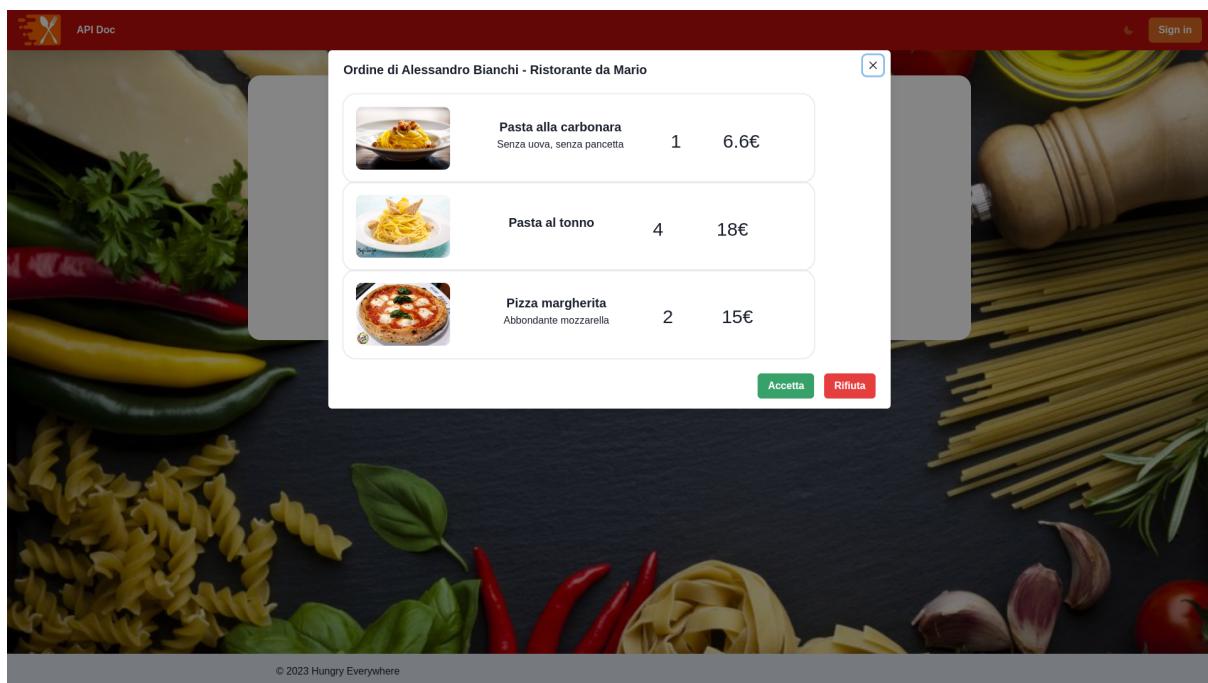
## 5.18 - Riepilogo ordini del gestore

La schermata di riepilogo degli ordini permette a ogni gestore di gestire gli ordini ricevuti dai suoi ristoranti. Ogni cliente è elencato con il nome, il cognome e l'indirizzo email, e il gestore può visualizzare i dettagli degli ordini effettuati facendo clic sul tasto "Visualizza ordine" accanto a ogni cliente.



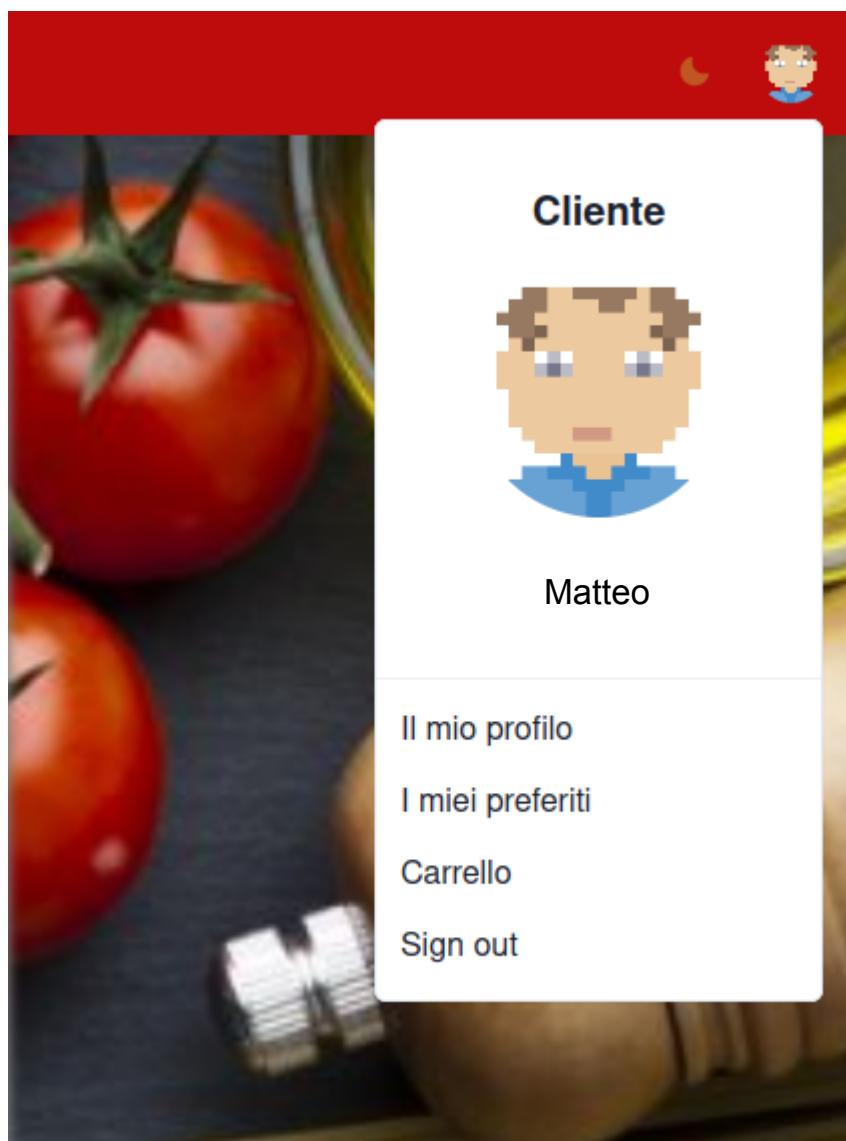
## 5.19 - Focus sull'ordine

Dopo aver premuto il pulsante "Visualizza ordine" presente al punto [5.18](#), viene visualizzata una schermata che mostra il nome del cliente che ha effettuato l'ordine e il nome del ristorante da cui ha scelto i piatti. La schermata include anche un riepilogo dell'ordine. Se il gestore della piattaforma seleziona il pulsante "Rifiuta", l'ordine viene rifiutato. Se viene selezionato il pulsante "Accetta", l'ordine viene accettato.



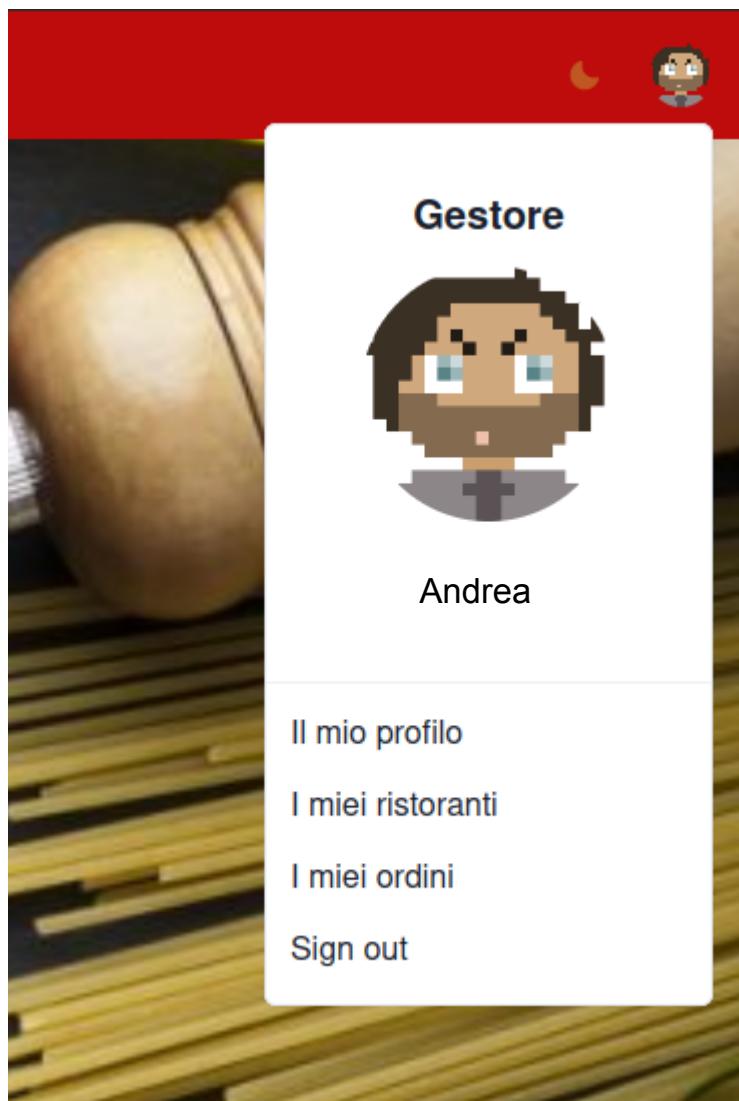
## 5.20 - Pagine accessibili dal cliente

Nella barra di navigazione, è presente un'icona raffigurante il cliente. Selezionandola, si accede alla schermata che mostra il tipo di utente connesso, il suo nome e le pagine a cui è possibile accedere. Nella stessa schermata, è presente un pulsante intitolato "Sign out" che consente di terminare la sessione dell'utente.



## 5.21 - Pagine accessibili dal gestore

Nella barra di navigazione, è presente un'icona raffigurante il gestore. Selezionandola, si accede alla schermata che mostra il tipo di utente connesso, il suo nome e le pagine a cui è possibile accedere. Nella stessa schermata, è presente un pulsante intitolato "Sign out" che consente di terminare la sessione dell'utente.



## 6. GitHub Repository and Deployment Info

Gli account dei membri del gruppo sono:

- Lorenzo Bevilacqua: <https://github.com/ardubev16>
- Alessandro Sartore: <https://github.com/Sartor02>
- Luca Tecchio: <https://github.com/LT-27>

### 6.1 Struttura GitHub Organization

Nell'organizzazione di GitHub sono presenti 3 repositories:

- **Documents:** inizialmente veniva usata per la stesura dei Deliverables, in seguito e' stata usata soltanto per tenere traccia delle ore di lavoro e per salvare tutti i diagrammi in alta risoluzione (formato svg);
- **Deliverables:** viene usata per la consegna dei deliverables;
- **Codebase:** viene usata per lo sviluppo dell'applicativo web, contiene sia Front End che Back End.

### 6.2 Struttura Codebase

La repository Codebase è strutturata in modo molto simile a un progetto Nextjs con Typescript con qualche aggiunta, per una descrizione più dettagliata vedere [3.1 PROJECT STRUCTURE](#).

## 6.3 Deployment locale

Per avviare l'applicazione in locale e' necessario avere *nodejs* e *npm* installati, e' inoltre necessario inserire le seguenti variabili d'ambiente nel file *.env.local*:

- **NEXTAUTH\_URL** e **NEXT\_PUBLIC\_SITE\_URL**, assegnando il seguente valore: *http://localhost:3000*
- **NEXTAUTH\_SECRET**, assegnando come valore una stringa generata col seguente comando “*openssl rand -hex 32*” oppure andando sul sito <https://generate-secret.now.sh/32>
- **MONGODB\_URI**, assegnando l'URI con credenziali valide per un database su MongoDB col seguente formato: “*mongodb+srv://<username>:<password>@<url>/<db\_name>*”. Per utilizzare il database già popolato mostrato negli screenshot precedenti riferirsi a [6.4 Deployment su Vercel](#).

E' necessario poi eseguire i seguenti comandi:

- *npm i*: serve per installare tutte le dependencies del progetto
- *npm run build*: serve per costruire le pagine del sito in modo che siano pronte a essere servite a chi fa la richiesta
- *npm run start*: serve per avviare il server che metterà a disposizione il sito web all'indirizzo <http://localhost:3000>

## 6.4 Deployment su Vercel

L'applicazione web è stata rilasciata sulla piattaforma di hosting gratuita [Vercel.com](https://vercel.com) al seguente indirizzo:  
<https://hungry-everywhere.vercel.app/>.

Ci sono degli utenti attivi che possono essere utilizzati nel caso in cui non si desideri creare un account per provare il sito:

Cliente: [alberto.rossi@gmail.com](mailto:alberto.rossi@gmail.com)

Alberto\_2023

Gestore: [marco.verdi@gmail.com](mailto:marco.verdi@gmail.com)

Marco\_2023

## 7. Testing

È stato utilizzato il framework Jest per effettuare degli unit-test automatici su alcune delle API realizzate. Gli unit-test sono locati nella cartella `__tests__` del progetto. Le restanti API sono state testate a mano con Postman perché essendo API autenticate si sono verificate delle difficoltà nell'implementazione dei test in Jest.

### 7.1 API testate con Jest

- Signup
- getMenu
- getRistorante
- thumbnails

Segue il risultato del testing automatico e uno snippet di codice responsabile del testing.

File	% Stmt	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	54.63	35.29	56.75	52.5	
lib	92.18	87.5	66.66	94.44	
dbConnect.ts	88.88	75	100	88.23	18, 44
typings.ts	100	100	100	100	
yupSchemas.ts	85.71	100	25	100	
lib/models	100	100	100	100	
menu.ts	100	100	100	100	
pietanza.ts	100	100	100	100	
ristorante.ts	100	100	100	100	
user.ts	100	100	100	100	
pages/api	33.12	20	52.94	34.75	
menu.ts	29.03	17.64	37.5	29.31	74-80, 198-370, 378-384
restaurant.ts	23.07	8.33	50	25.37	214-491, 502-508
thumbnails.ts	94.11	85.71	100	93.75	90
pages/api/auth	56	33.33	28.57	57.44	
[...nextauth].ts	28.57	0	0	30.76	135-174
signup.ts	90.9	75	100	90.47	74, 112
Test Suites: 4 passed, 4 total					
Tests: 24 passed, 24 total					
Snapshots: 0 total					
Time: 7.522 s					
Ran all test suites.					

```
describe("POST /api/auth/signup", () => {
  it("should return 500, no password", async () => {
    const { req, res } = createMocks({ method: "POST" });
    await signupHandler(req, res);
    expect(res.statusCode).toBe(500);
    expect(res._getData()).toEqual({
      success: false,
      error: "Password is essential!",
    });
  });
  it("should return 500, password too short", async () => {
    const { req, res } = createMocks({
      method: "POST",
      body: { password: "foo" },
    });
    await signupHandler(req, res);
    expect(res.statusCode).toBe(500);
    expect(res._getData()).toEqual({
      success: false,
      error: "Password must be at least 8 characters",
    });
  });
  it("should return 500, password too long", async () => {
    const { req, res } = createMocks({
      method: "POST",
      body: { password: "f".repeat(257) },
    });
    await signupHandler(req, res);
    expect(res.statusCode).toBe(500);
    expect(res._getData()).toEqual({
      success: false,
      error: "Password too long! (max: 256)",
    });
  });
});
```

## 7.2 API testate con Postman

- signin, signout, session, csrf
  - myrestaurants
  - addRistorante, editRistorante, deleteRistorante
  - addMenu, editMenu, deleteMenu

Seguono due esempi di testing manuale utilizzando Postman.

The screenshot shows the Postman application interface. At the top, it displays the URL "localhost:3000/api/menu" and the method "POST". The "Body" tab is selected, showing a JSON payload representing a menu item. The response tab shows a successful 200 OK status with a response body containing a success message and a menu ID.

POST localhost:3000/api/menu

Params Authorization Headers (10) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 [JSON]
2   {
3     "name": "primi",
4     "tipologia": "primi",
5     "ristoranteId": "6d87b60f-1c65-4406-948f-3046a0a0d5de",
6     "piatti": [
7       {
8         "name": "pasta al tonno",
9         "prezzo": 5.28,
10        "allergeni": [],
11        "calorie": 668,
12        "immagine": "null",
13        "ingredienti": ["pasta", "tonno"],
14        "isDisponibile": true
15      },
16      {
17        "name": "pasta alla carbonara",
18        "prezzo": 6.50,
19        "allergeni": ["uova e derivati", "latte e derivati"],
20        "calorie": 900,
21        "immagine": "null",
22        "ingredienti": ["pasta", "uova", "guanciale", "pecorino", "pepe"],
23        "isDisponibile": true
24      }
25    ]
26  }
```

Body Cookies (3) Headers (8) Test Results

Status: 200 OK Time: 158 ms Size: 772 B Save Response

Pretty Raw Preview Visualize JSON

```
1 [JSON]
2   {
3     "success": true,
4     "menuId": "9f21f5db-23d2-4139-96f0-c2d7407d1285"
5   }
```

DELETE localhost:3000/api/menu?id=9f21f5db-23d2-4139-96f0-c2d7407d1285 Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> id	9f21f5db-23d2-4139-96f0-c2d7407d1285	
Key	Value	Description

Body Cookies (3) Headers (8) Test Results

Status: 401 Unauthorized Time: 156 ms Size: 776 B Save Response

Pretty Raw Preview Visualize JSON ↻

```
1 { "success": false, 2   "error": "Unauthorized or menu not found" 3 } 4
```