

# INGEGNERIA DEL SOFTWARE

SuperF®igo

Sviluppo Applicazione

Isacco Cenacchi, Enrico Dal Bianco, Pietro De Angeli

Università di Trento

# INDICE DEL DOCUMENTO

<b>1. Scopo del documento</b>	<b>4</b>
<b>2. User Flows</b>	<b>5</b>
<b>3. Application Implementation and Documentation</b>	<b>7</b>
3.1 Project Structure	7
3.2 Project Dependencies	9
3.3 Project Data or DB	10
guests	10
ingredients	10
recipes	11
recipes_images	11
storage	12
users	12
3.4 Project APIs	13
3.4.1 Resources Extraction from the Class Diagram	13
Resources Extraction Diagram:	14
3.4.2 Resources Models	15
Procedure di autenticazione	15
Cambio Password ed Eliminazione Account	15
Ingredienti, Dispensa, Ricette e Info Utente	16
3.5 Sviluppo API	16
3.5.0 server init	17
3.5.1 POST /login	18
3.5.2 POST /register	19
3.5.3 GET /guest_registration	20
3.5.4 POST /change_password	21
3.5.5 DELETE /delete_account	22
3.5.6 GET /user	23
3.5.7 GET /all_ingredients	24
3.5.8 GET /ingredients	24
3.5.9 POST /add	25
3.5.10 DELETE /remove	26
3.5.11 GET /recipes	27
<b>4. API Documentation</b>	<b>28</b>
<b>5. Front End Implementation</b>	<b>31</b>
5.1 Home page	31
5.2 Login	32
5.3 Registrazione	33
5.4 Area riservata	34

5.5 Cambio password	34
5.6 Cerca ingrediente	35
5.7 Aggiungi ingrediente	35
5.8 Dispensa	36
5.8.1 Card ingrediente	36
5.9 Ricette	37
5.9.1 Card ricetta	37
5.10 Info ricetta	38
5.11 Header	39
5.11.1 Header Desktop	39
5.11.1 Header Mobile	39
5.12 Footer	40
5.13 About us	40
<b>6. GitHub Repository and Deployment Info</b>	<b>41</b>
6.1 Struttura GitHub Organization	41
6.2 Deployment locale	41
6.3 Deployment su Vercel (front-end)	41
6.4 Deployment su railway (back-end)	42
<b>7. Testing</b>	<b>43</b>

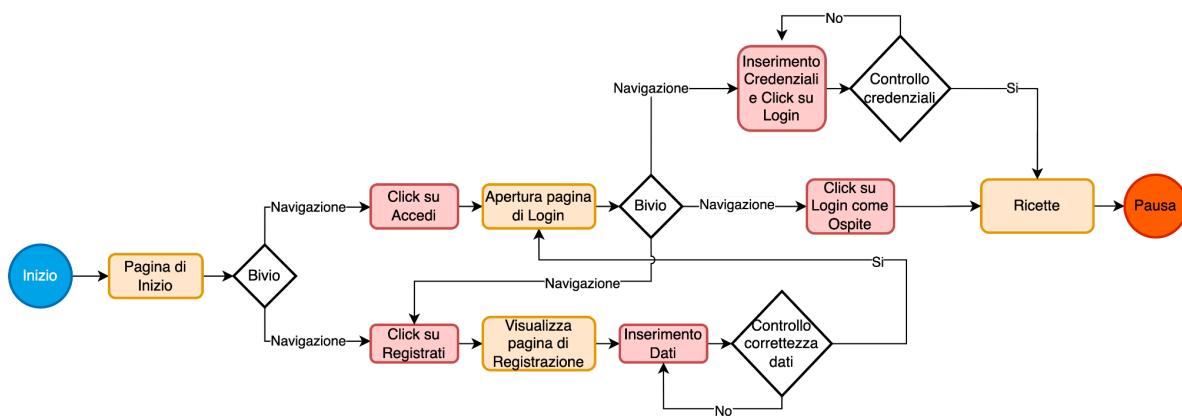
## 1. Scopo del documento

Il presente documento riporta tutte le informazioni necessarie per lo sviluppo di una parte dell'applicazione SuperF®igo. Partendo dalla descrizione degli user flow legate al ruolo dell'utente dell'applicazione, il documento prosegue con la presentazione delle API necessarie (tramite l'API Model e il Modello delle risorse) per poter ricercare gli ingredienti, aggiungerli alla dispensa, modificarli, ed infine le API per poter trovare le Ricette e visualizzarle . 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

In questa sezione del documento di sviluppo riportiamo gli “user flows” di un utente del nostro sito.

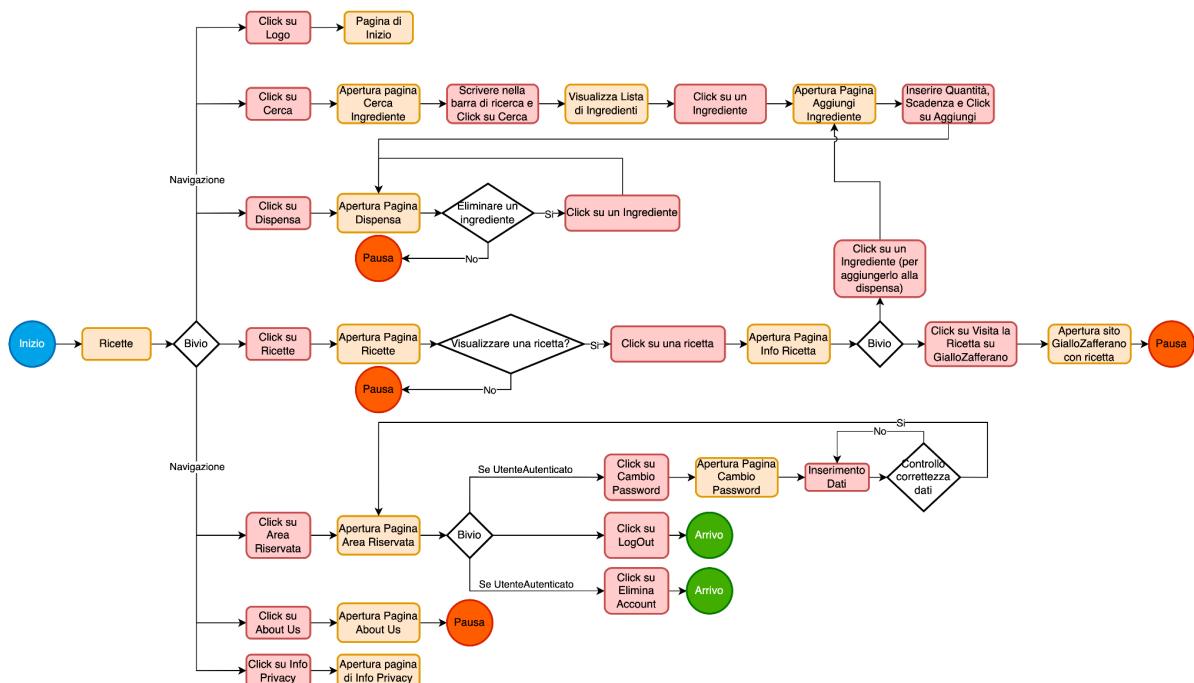
Nel seguente User Flow vengono mostrate le azioni di un utente, il quale, partendo dalla pagina di Inizio, dopo aver eseguito l’accesso, come UtenteAutenticato o come Ospite verrà rediretto alla Pagina delle Ricette. Un utente appena entrato nel nostro sito può decidere se andare sulla pagina di login, quindi effettuare il login come utente autenticato o utente ospite (senza la garanzia che i suoi dati vengano salvati se viene persa la sessione), o sulla pagina registrazione che permette di registrarsi al sito creando l’account e avendo la certezza che le modifiche siano persistenti nel sito.



Nel seguente User Flow invece si mostrano le possibili scelte che può effettuare un utente una volta loggato.

In particolare l'utente può decidere di:

- Cliccare sul Logo per tornare alla pagina di Inizio (Login o Registrazione)
- Cercare un Ingrediente da poi aggiungere alla dispensa
- Visualizzare e modificare la propria dispensa
- Visualizzare le ricette che puo cucinare con gli ingredienti della dispensa
- Aprire la pagina dell'Area Riservata per gestire il proprio account, cambiando password, facendo il logout o eliminando il proprio account
- Aprire la pagina About Us
- Aprire la pagina Info Privacy



### 3. Application Implementation and Documentation

Nelle sezioni precedenti abbiamo identificato varie features, queste dovranno poi essere implementate considerando come il nostro utente finale potrà utilizzarle nel suo flusso applicativo.

L'applicazione è stata sviluppata utilizzando NodeJS. Per la gestione dei dati utilizzeremo invece MongoDB. Sia nel front-end che nel back-end abbiamo fatto un largo uso di javascript.

#### 3.1 Project Structure

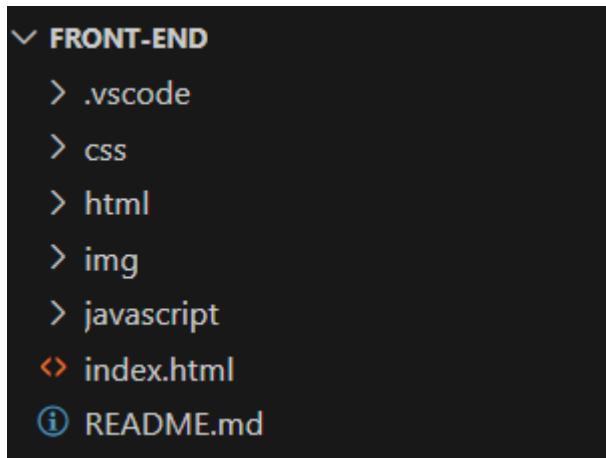
Il sito è composto da 2 directory (in due differenti repository su Github):

- Back-end
- Front-end

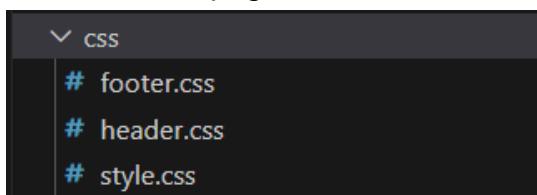
Nella cartella Back-end è presente il file “app.js” dove sono presenti le API di comunicazione tra sito e database, nella cartella test sono presenti gli unit test. E il file swagger.yaml rappresenta la documentazione delle API.

```
> node_modules
  ↘ test
    JS app.test.js
    JS jest.config.cjs
    .gitignore
    JS app.js
    {} globalConfig.json
    JS index.js
    {} package-lock.json
    {} package.json
    {} swagger.yaml
```

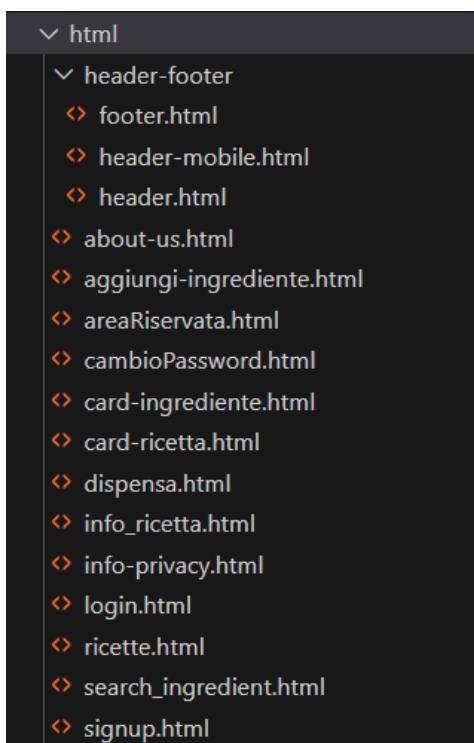
Nella cartella Front-end invece sono presenti i file HTML, CSS e JS per la gestione visiva del sito ed è strutturato in 4 cartelle:



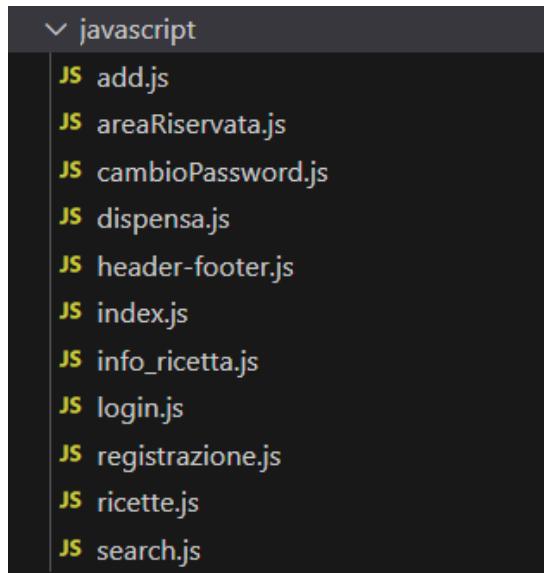
- **css:** contiene 3 file che sono rispettivamente i fogli di stile di header, footer e del resto delle pagine



- **html:** contiene tutte le pagine html del sito ad eccezione di index.html.  
All'interno di html è presente una sottocartella header-footer che contiene il codice html del footer, dell'header desktop e quello mobile



- **img:** contiene l'immagine dell'utente generico (user-icon.jpg) e favicon.ico che sarebbe l'icona del sito
- **javascript:** contiene tutti i file javascript del sito



## 3.2 Project Dependencies

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

- body-parser
- cookie-parser
- cors
- express-session
- express
- mongodb
- swagger-ui-express
- yamljs

### 3.3 Project Data or DB

Per poter gestire tutti i dati utili alla realizzazione del sito SuperF®igo abbiamo definito 5 strutture dati:

- **guests**: per contenere tutte le informazioni riguardanti l'UtenteOspite
- **ingredients**: che contiene tutti gli ingredienti presenti nel database
- **recipes**: che contiene tutte le ricette presenti nel database
- **recipes\_images**: che contiene tutte le immagini delle ricette
- **storage**: che contiene la dispensa di ogni singolo utente (sia Ospite che Autenticato) con all'interno gli ingredienti e la propria data di scadenza
- **users**: per contenere tutte le informazioni riguardanti l'UtenteAutenticato

Come illustrato in questa figura:

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
guests	47	3.81KB	83B	36KB	1	36KB	36KB
ingredients	1688	78.66KB	48B	68KB	1	64KB	64KB
recipes	6216	495.39MB	81.61KB	507.42MB	1	256KB	256KB
storage	12	2.22KB	190B	36KB	1	36KB	36KB
users	12	1.65KB	141B	36KB	1	36KB	36KB

In dettaglio le collezioni sopra citate per descriverne il contenuto:

#### guests

La tabella “guests” contiene “id” una stringa che contiene l’UserId univoco di ogni utente, in questo caso ospite, che serve per identificarlo (per esempio per assegnargli la propria dispensa). Contiene inoltre la data di registrazione per poter eliminare i guests dopo 30gg.

```
_id: ObjectId('64f44d6e497cceabbc3d2510')
id: "83553124511195ad9d620f5e9e6b305c"
register_date: 1693732206131
```

#### ingredients

La tabella “ingredients” contiene “name” una stringa che contiene il nome dell’ingrediente.

```
_id: ObjectId('634434a2ce63bc24789a9ac2')
name: "Confettura di mirtilli"
```

## recipes

La tabella “recipes” contiene “title” con il nome della ricetta, “ingredients” con un array di ingredienti ognuno con la sua quantità che servono per la preparazione della ricetta, “link” una stringa che contiene il link alla ricetta di GialloZafferano ed infine un array “data” che contiene altri dati della ricetta utili per una futura implementazione di nuove funzionalità del sito.

```

_id: ObjectId('64f6f1349fef7a2ebe28f60e')
title: "Spaghetti Cacio e Pepe"
▼ ingredients: Array
  ▼ 0: Object
    Spaghetti: "320 g"
  ▶ 1: Object
  ▶ 2: Object
link: "https://ricette.giallozafferano.it/Spaghetti-Cacio-e-Pepe.html"
▼ data: Array
  ▼ 0: Object
    Difficolta: " Media"
  ▶ 1: Object
  ▶ 2: Object

```

## recipes\_images

Questa tabella contiene l’immagine rappresentante la ricetta, abbiamo deciso di spostare questa informazione in una tabella separata rispetto alle ricette per alleggerire il carico del server quando chiediamo le migliori ricette per un utente.

```

_id: ObjectId('64f6f1349fef7a2ebe28f60d')
title: "Spaghetti Cacio e Pepe"
image: "/9j/4AAQSkZJRgABAQIAJQAlAAD/2wCEAAMDAwMDAwMDAwMEBAQEAYFBQUFBgkGBwYHBg..."
```

## storage

La tabella “storage” contiene “userId” una stringa che contiene l’UserId univoco dell’utente che è proprietario di questa dispensa, “ingredients” un array di ingredienti che rappresenta la dispensa dell’utente e contiene gli ingredienti con la propria data di scadenza.

```
_id: ObjectId('64e07551defabd14818f554b')
userId: "c77a8db37f09d7618ee3ac2c0a69ee3a"
▼ ingredients: Array
  ▼ 0: Object
    name: "Spaghetti"
    expiration: "01/03"
  ▼ 1: Object
    name: "Sugo"
    expiration: "01/03"
```

## users

La tabella “users” contiene “username” una stringa che contiene il nome dell’utente, “psw” una stringa che contiene la password hashata, “email” una stringa che contiene la mail dell’utente ed infine “userId” una stringa che contiene l’UserId univoco di ogni utente e che serve per identificarlo (analogo all’”id” del guest)

```
_id: ObjectId('64f05f91710d4d09f5cd3794')
username: "Isacco"
psw: "ae1a69cf1384950ffe79ef187aae12bf0d1e4a04544625d9944bb271205a0f33"
email: "isacco.cenacchi@studenti.unitn.it"
userId: "530c58ddb255e18dbcf2039a920ef546"
```

Ogni record contiene “\_id” rappresentato da un ObjectId generato casualmente da Mongodb.

## 3.4 Project APIs

In questo capitolo verranno analizzate le API scritte e utilizzate nel corso della scrittura del codice.

### 3.4.1 Resources Extraction from the Class Diagram

Tramite il class diagram identifichiamo le risorse richieste per valutare quali API implementare nel sito.

Abbiamo individuato le seguenti risorse: User e Storage.

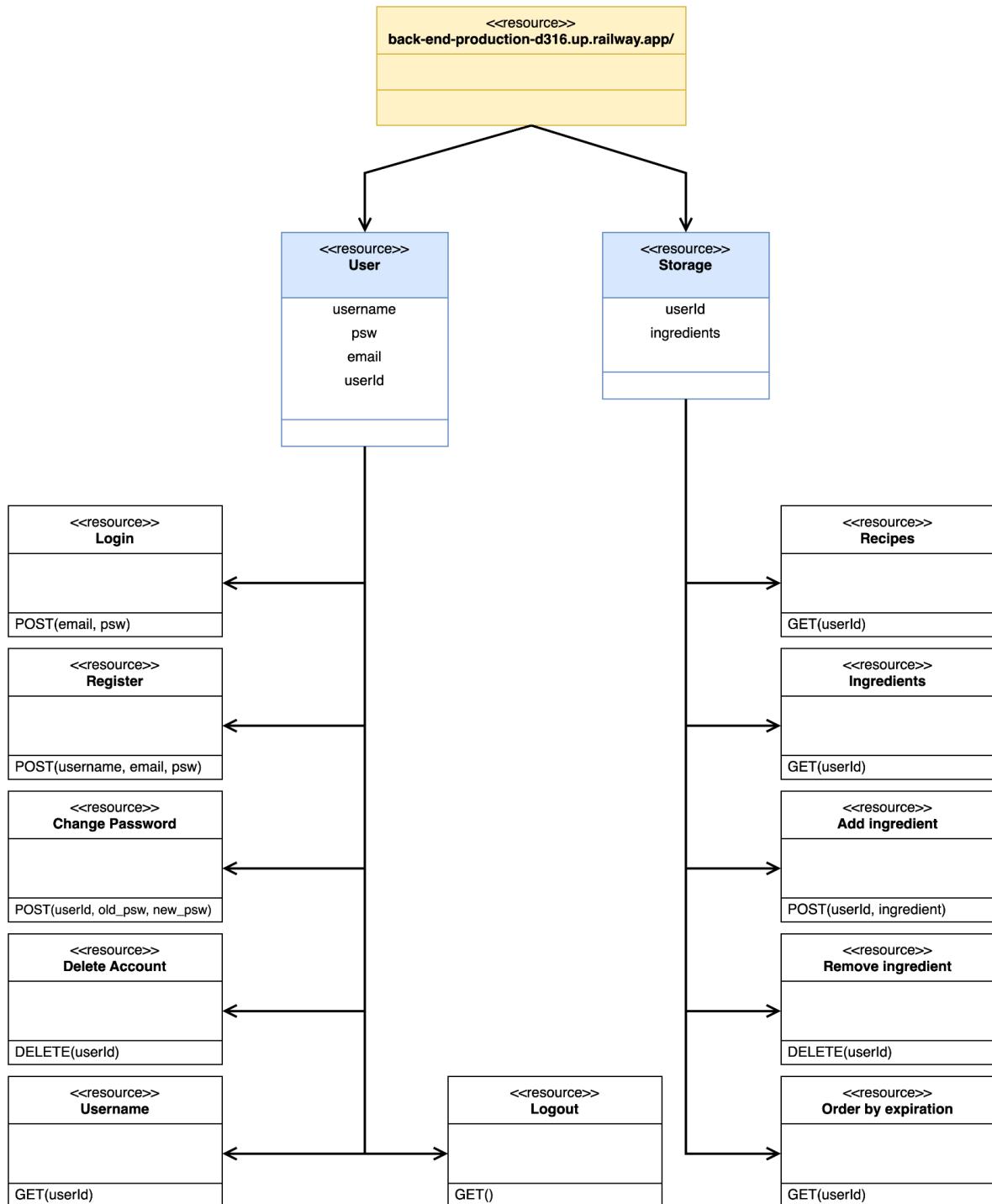
**User** implementa le seguenti API:

- Register (back-end)
- Login (back-end)
- Logout (front-end)
- Change password (back-end)
- Delete account (back-end)
- User (back-end)

Mentre **Storage** le seguenti:

- Recipes (back-end)
- Ingredients (back-end)
- Add ingredient (back-end)
- Remove ingredient (back-end)
- Order by expiration (front-end)

## Resources Extraction Diagram:

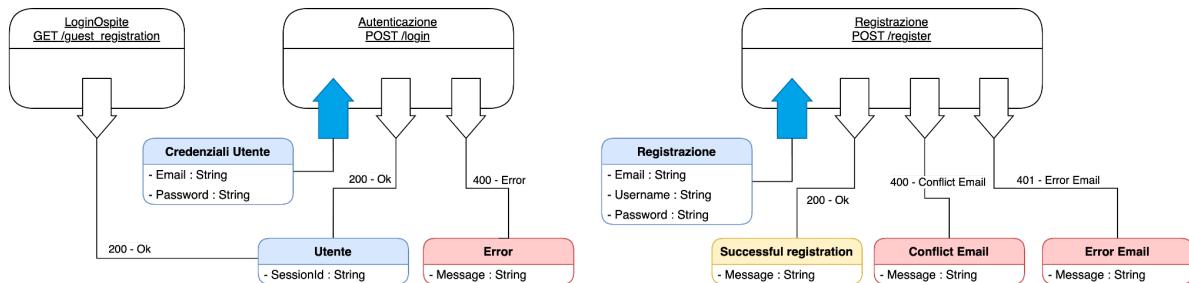


### 3.4.2 Resources Models

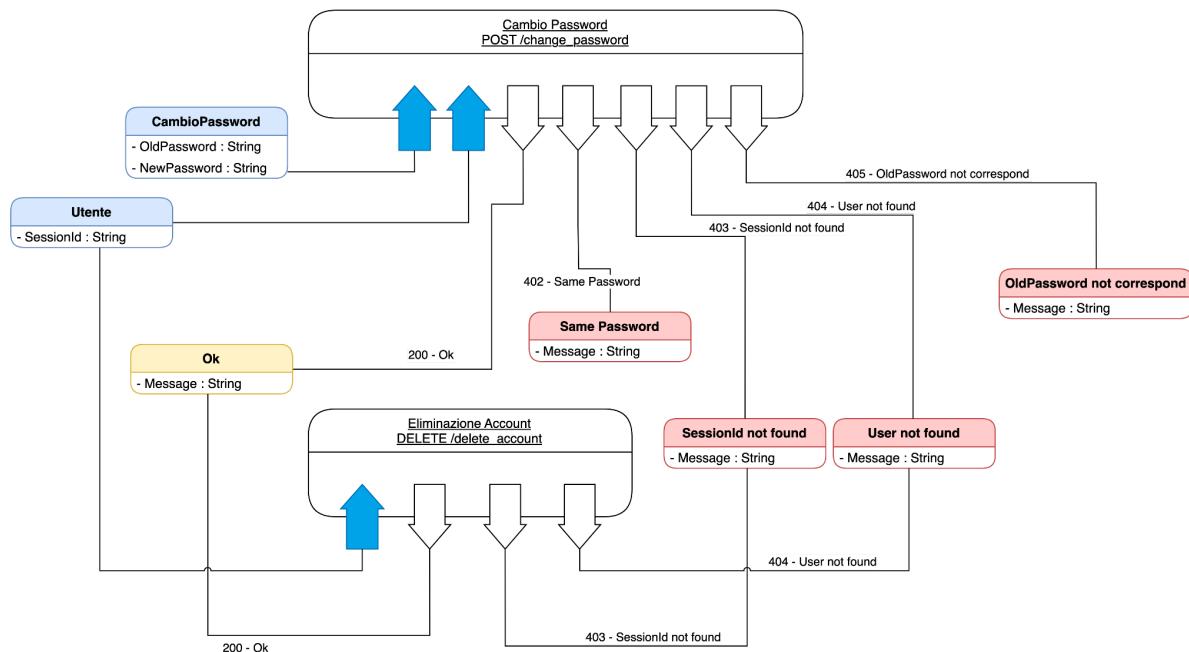
Il Resources Model che viene presentato è una specifica del Resources Extraction. Sono presenti solo le API più importanti per il progetto, insieme ad alcune API aggiuntive che sono necessarie per il funzionamento di quelle principali.

#### Procedure di autenticazione

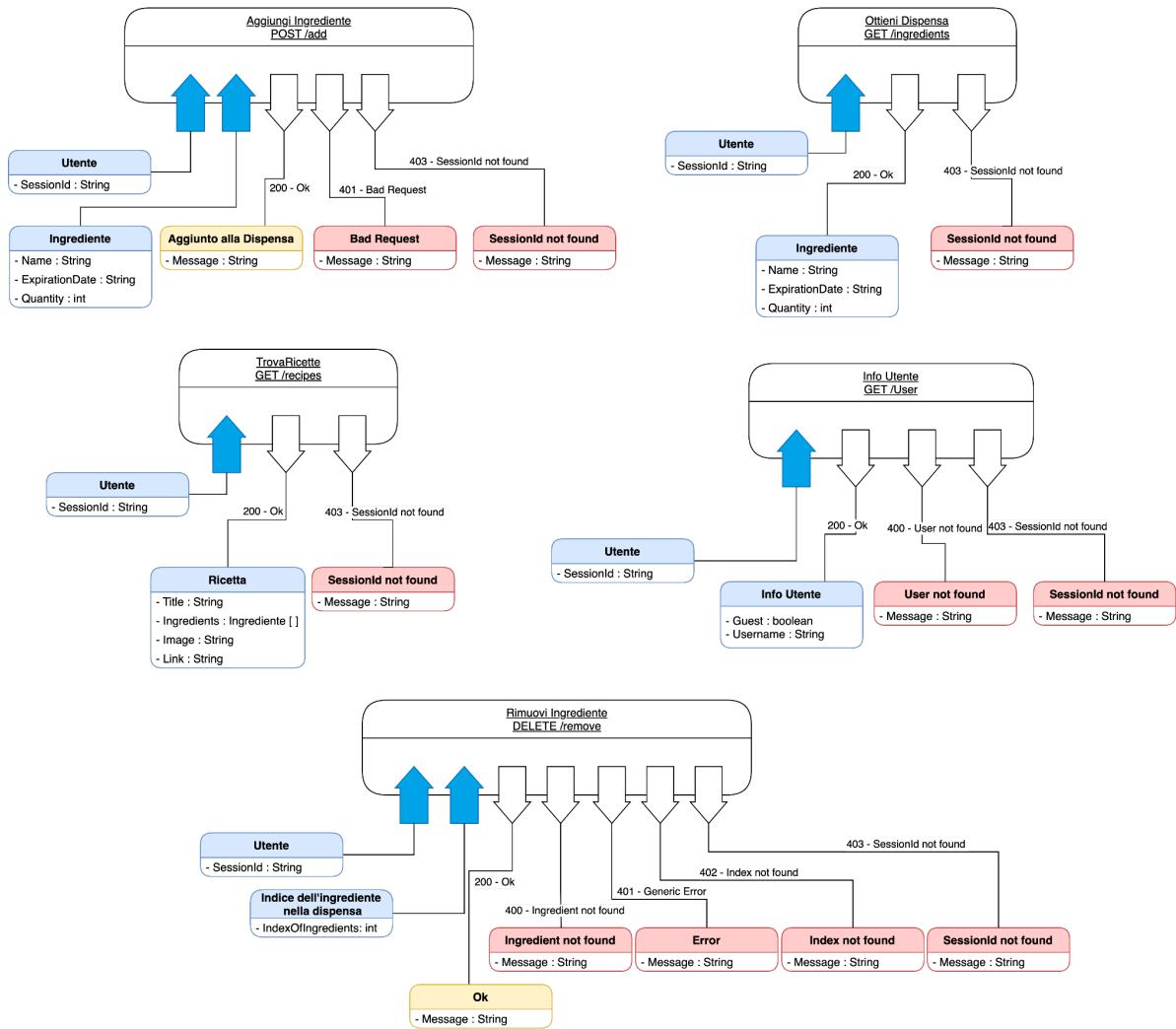
Le procedure di autenticazione non identificano una risorsa ma sono i processi di login che vengono utilizzati sia da UtenteAutenticato (login e register) che da UtenteOspite (guest\_registration) e che servono per accedere al sistema. Tutto ciò è descritto dal Resource Model sotto indicato.



#### Cambio Password ed Eliminazione Account



## Ingredienti, Dispensa, Ricette e Info Utente



## 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 garantire il corretto funzionamento.

### 3.5.0 server init

La funzione connect viene eseguita prima di tutto e serve per inizializzare alcune variabili.

Come prima cosa si connette al DB e prende tutti i riferimenti alle varie collezioni che vengono usate dalle API, poi prende tutti gli id dei guest e gli carica in memoria (per velocizzare le varie operazioni), infine carica tutte le ricette in memoria.

```
1  async function connect(){
2    try{
3      await client.connect();
4      db = client.db(dbName);
5      users = db.collection("users");
6      guests = db.collection("guests");
7      recipes = db.collection("recipes");
8      recipes_images = db.collection("recipes_images");
9      storage = db.collection("storage");
10     ingredients = db.collection("ingredients")
11     let guest_ids = await guests.find().toArray();
12     guest_ids.forEach((e) => {is_guest.add(e.id); sessions[e.id] = {id: e.id, time: Date.now()}});
13     await get_recipes();
14   }
15   catch(e){
16     console.log(e);
17   }
18 }
19
```

### 3.5.1 POST /login

Questa API controlla semplicemente che la password corrispondente alla mail sia corretta, in caso positivo crea una sessione dall'id utente e ritorna il nuovo session id, altrimenti ritorna l'errore 400 (“Wrong Credentials”).

```
1 app.post('/login', async (req, res) => {
2     console.log("/login");
3     let id = null;
4     let email = req.body.email;
5     let psw = req.body.psw;
6     id = await check_credentials(email, psw);
7     if(id != null)
8     {
9         let sid = generate_session_id(id);
10        res.send({sessionId: sid});
11    }
12    else
13    {
14        res.status(400).send("Wrong Credentials");
15    }
16});
```

### 3.5.2 POST /register

Per prima cosa viene controllato se l'email è valida attraverso una regular expression, nel caso non sia valida viene restituito l'errore 401 ("email not valid"). Poi viene controllato se nel DB è già presente quella mail, nel caso sia già presente ritorna l'errore 400 ("email already used"), altrimenti ritorna semplicemente "ok". Il Back End in quest'ultimo caso dovrebbe mandare una mail di verifica per poi inserire registrare effettivamente l'utente, ma per mancanza di conoscenze abbiamo deciso di non implementare questa funzione e di registrare direttamente l'utente.



```
1 app.post('/register', async (req, res) => {
2     console.log("/register");
3     let mail = req.body.mail;
4     let psw = req.body.psw;
5     if(!validateEmail(mail))
6     {
7         res.status(401).send("email not valid");
8         return;
9     }
10    let id = await generate_user_id();
11    let check = await users.findOne({email: mail});
12    if(check == null){
13        //todo: send verification email
14        users.insertOne({username: req.body.username, psw: psw, email: mail, userId: id});
15        res.send("ok")
16        return;
17    }
18    res.status(400).send("email already used");
19});
```

### 3.5.3 GET /guest\_registration

Questa API genera semplicemente un nuovo id e lo aggiunge alla collezione guests, quando inserisco un nuovo guest inserisco anche la data di creazione così da eliminare dopo 30gg, per gli utenti guest l'id utente e l'id sessione sono uguali, quindi creo anche una nuova sessione e restituisco il SID.

```
1 app.get('/guest_registration', async (req, res) => {
2   console.log("/guest_registration");
3
4   let id = await generate_user_id();
5
6   guests.insertOne({id: id, register_date: Date.now()});
7   is_guest.add(id);
8   sessions[id] = {id: id, time: Date.now()};
9   res.send(id.toString());
10});
```

### 3.5.4 POST /change\_password

Riceve in input la nuova e la vecchia password e il SID. Per prima cosa controlla che la nuova password sia diversa dalla vecchia, poi controlla che la sessione sia valida, poi controlla che la vecchia password corrisponda e infine se l'operazione è andata a buon fine ritorna "ok". Questa funzione è utilizzabile solo da utenti registrati (non guest), nel caso un guest chiami questa funzione il codice di ritorno sarà 404 (user not found).

```
1 app.post('/change_password', async (req, res) => {
2     console.log("/change_password");
3     let sid = req.body.sid;
4     let old_psw = req.body.old_psw;
5     let new_psw = req.body.new_psw;
6     if(old_psw == new_psw)
7     {
8         res.status(402).send("the new password is the same as the old one");
9         return;
10    }
11    if(check_session(sid)) {
12        res.status(403).send("session not found");
13        return;
14    }
15    let uid = sessions[sid].id;
16    try{
17        let psw_check = await users.findOne({userId: uid});
18        if(old_psw != psw_check.psw)
19        {
20            res.status(405).send("old password does not correspond");
21            return;
22        }
23        await users.updateOne({ userId: uid }, {$set:{psw: new_psw}});
24        res.send("ok");
25    }
26    catch (error)
27    {
28        console.error(error);
29        res.status(404).send("user not found");
30    }
31 });


```

### 3.5.5 DELETE /delete\_account

Riceve in input solo il SID, controlla che la sessione sia valida ed elimina l'utente e il rispettivo storage. Questa funzione è utilizzabile solo da utenti registrati dato che i guest vengono eliminati automaticamente (da uno script separato) dopo 30gg.

```
1 app.delete('/delete_account', async (req, res) => {
2   console.log("/delete_account");
3   let sid = req.body.sid;
4   if(check_session(sid)) {
5     res.status(403).send("session not found");
6     return;
7   }
8   let uid = sessions[sid].id;
9
10  let response = await users.deleteOne({userId: uid});
11
12  if(response.deletedCount == 1)
13  {
14    await storage.deleteOne({userId: uid});
15    res.send("ok");
16  }
17  else res.status(404).send("user not found");
18});
```

### 3.5.6 GET /user

Riceve come parametro il SID, dopo aver controllato che la sessione sia valida restituiamo un oggetto che indica se l'utente è un guest oppure no insieme all'username, se l'utente è un guest l'username sarà "guest".

```
● ● ●

1 app.get("/user", async (req, res) => {
2   console.log("/user");
3   let sid = req.query.sid;
4   if(check_session(sid)) {
5     res.status(403).send("session not found");
6     return;
7   }
8
9   if(is_guest.has(sid)) res.send({guest: true, username: "guest"});
10  else
11  {
12    let uid = sessions[sid].id;
13    let user = await users.findOne({userId: uid});
14    if(user == null)
15    {
16      res.status(400).send("user not found");
17      return;
18    }
19    res.send({guest: false, username: user.username});
20  }
21})
```

### 3.5.7 GET /all\_ingredients

Restituisce tutti gli ingredienti presenti su tutte le ricette.



```
1 app.get('/all_ingredients', async (req, res) => {
2   console.log("/all_ingredients");
3   res.send(await get_all_ingredients())
4 }
5
```

### 3.5.8 GET /ingredients

Riceve in ingresso il SID, controlla se la sessione è valida e infine restituisce tutti gli ingredienti presenti nella dispensa dell'utente.



```
1 app.get('/ingredients', async (req, res) => {
2   console.log("/ingredients");
3   let sid = req.query.sid;
4   if(check_session(sid)) {
5     res.status(403).send("session not found");
6     return;
7   }
8   let id = sessions[sid].id;
9   let ingredients = await get_ingredients(id);
10  res.send(ingredients);
11});
```

### 3.5.9 POST /add

Riceve in ingresso il SID ed un ingrediente, controlla la sessione e aggiunge l'ingrediente alla dispensa dell'utente.

```
1 app.post('/add', (req, res) => {
2     console.log("/add");
3     let ingredient = req.body.ingredient;
4     let sid = req.body.sid;
5     if(check_session(sid)) {
6         res.status(403).send("session not found");
7         return;
8     }
9     let id = sessions[sid].id;
10    add_ingredient(id, ingredient);
11    res.send("ok");
12});
```

### 3.5.10 DELETE /remove

Riceve in input il SID e un ingrediente, controlla la sessione poi controlla se l'utente ha una dispensa (la dispensa viene creata quando l'utente esegue il primo /add), in questo caso rimuove un ingrediente. La funzione conta anche quanti elementi vengono rimossi, in questo caso saranno sempre 0 o 1 e restituisce il valore in modo da notificare al front-end se l'eliminazione è andata a buon fine.

```
1  app.delete('/remove', async (req, res) => {
2      console.log('/remove');
3      let sid = req.body.sid;
4      let ingredient = req.body.ingredient;
5      console.log(ingredient);
6      if(check_session(sid)) {
7          res.status(403).send("session not found");
8          return;
9      }
10     let id = sessions[sid].id;
11     try{
12         let st = await storage.findOne({userId: id});
13         if(st == null){
14             res.send("storage not found");
15             return;
16         }
17         let old = st.ingredients.length;
18         const index = st.ingredients.findIndex(value => value.name == ingredient);
19         if (index > -1) {
20             st.ingredients.splice(index, 1);
21         }
22         await storage.replaceOne({userId: id}, st);
23         let removed = old-st.ingredients.length;
24         res.send("removed " + removed.toString() + " elements");
25     }
26     catch(error)
27     {
28         console.log(error);
29         res.status(401).send("error");
30     }
31 })
```

### 3.5.11 GET /recipes

Dato il SID restituisce le 10 migliori ricette data la dispensa dell'utente.

```
1 app.get('/recipes', async (req, res) => {
2     console.log("/recipes");
3     let sid = req.query.sid;
4     if(check_session(sid)) {
5         res.status(403).send("session not found");
6         return;
7     }
8     let id = sessions[sid].id;
9     let ret = await get_possible_recipes(id);
10    res.send(ret);
11});
```

## 4. API Documentation

Le API Locali fornite dall'applicazione SuperF®igo e descritte nella sezione precedente sono state documentate utilizzando il modulo di NodeJS chiamato Swagger-UI-React.

In questo modo la documentazione relativa alle API è direttamente disponibile a chiunque possiede il codice sorgente. È stato utilizzato Swagger UI in quanto crea una pagina web dalle definizioni delle specifiche OpenAPI ed è inoltre possibile testarle direttamente dalla pagina web.

In particolare, di seguito viene mostrata la pagina web relativa alla documentazione che presenta le 11 API (GET, POST e DELETE, tre tipi di metodi HTTP diversi) per la gestione 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 del database, 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-docs> se il sito è avviato in locale, altrimenti è reperibile grazie al server tramite il seguente link:

<https://back-end-production-d316.up.railway.app/api-docs/>

Di seguito mostriamo la lista delle api documentate:

Method	Path	Description
POST	/login	Login api.
POST	/register	Register api.
POST	/add	Add ingredient
GET	/ingredients	list of ingredients of the user
GET	/all_ingredients	All ingredients
GET	/recipes	Get possible recipes
GET	/guest_registration	Login for guests
GET	/user	Return the username from a sid
POST	/change_password	Change the user password
DELETE	/remove	Remove an ingredient from the user storage, given the index from the storage
DELETE	/delete_account	Delete the user

Mostriamo inoltre un esempio per ogni tipo di API (POST, GET e DELETE), questi tre metodi differiscono a livello HTML:

**POST /login Login api.**

Used to log in

**Parameters**

No parameters

Request body

application/json

Example Value | Schema

```
[{"email": "esempio@gmail.com", "psw": "HashedPassword"}]
```

**Responses**

Code	Description	Links
200	SID (session id)	No links
	Media type	
	json	
	Controls Accept header.	
	Example Value   Schema	
	"string"	
400	Error	No links
	Media type	
	json	
	Example Value   Schema	
	"Wrong Credentials"	

**GET /ingredients list of ingredients of the user**

Get all the ingredients in the user storage

**Parameters**

Try it out

Name	Description
sid * required	SID string (query) Default value : 000SessionID000 000SessionID000

**Responses**

Code	Description	Links
200	List of ingredients	No links
	Media type	
	json	
	Controls Accept header.	
	Example Value   Schema	
	[{"name": "string", "expiration": "string", "quantity": 0}]	
403	Session not found	No links
	Media type	
	json	
	Example Value   Schema	
	"session not found"	

**DELETE /delete\_account** Delete the user

Used to delete all data of an account

**Parameters**

No parameters

**Request body**

application/json

**Example Value | Schema**

```
[{"sid": "000SessionID000"}]
```

**Responses**

Code	Description	Links
200	Ok code	No links
403	Error	No links

Media type: json

Controls Accept header.

Example Value | Schema

```
"ok"
```

Media type: json

Example Value | Schema

```
"session not found"
```

Di seguito, inoltre, viene riportata la definizione dei modelli utilizzati:

**Schemas**

**Ingredient** ↴ {

- name > [...]
- expiration > [...]
- quantity > [...]

}

**Recipe** ↴ {

- title > [...]
- ingredients > [...]
- image > [...]
- link > [...]
- data > [...]

}

## 5. Front End Implementation

In questa sezione parleremo del front end di Super Frigo, analizzeremo i componenti delle pagine e cosa permettono di fare.

### 5.1 Home page

L'home page di Super frigo presenta una schermata con il nome del nostro sito, e un pulsante per accedere alla pagina di login e registrazione.



## 5.2 Login

La pagina di login contiene un form che permette di inserire mail e password dell'account attraverso i due appositi campi testuali, grazie al tasto login è possibile fare un check delle credenziali. Questa pagina ha anche un tasto "accedi con Google" che permette di accedere usando le credenziali del proprio profilo Google facilitando l'accesso al sito. Se un utente non ha ancora un profilo al sito web è possibile cliccare sul link registrati per poter accedere alla pagina di registrazione. In alternativa è possibile cliccare il tasto "Login come ospite" dove i dati sono salvati solo per la vita della sessione dell'utente.

The screenshot shows a login interface with the following elements:

- Header:** The word "Login" is displayed prominently in large orange text at the top center.
- Text Instructions:** Below the header, there is a block of text in a smaller dark font:

Inserisci la mail e la password per accedere come utente autenticato.  
Clicca login come ospite se non hai un account, altrimenti clicca su Registrati.  
In alternativa puoi effettuare il login con Google.
- Email Input:** A text input field labeled "email:" followed by a placeholder text area.
- Password Input:** A text input field labeled "Password:" followed by a placeholder text area.
- Login Button:** A large orange button containing the text "Login".
- Guest Login Button:** A smaller orange button containing the text "Login come Ospite".
- Registration Link:** Text at the bottom left reading "Non hai un account? [Registrati](#)".

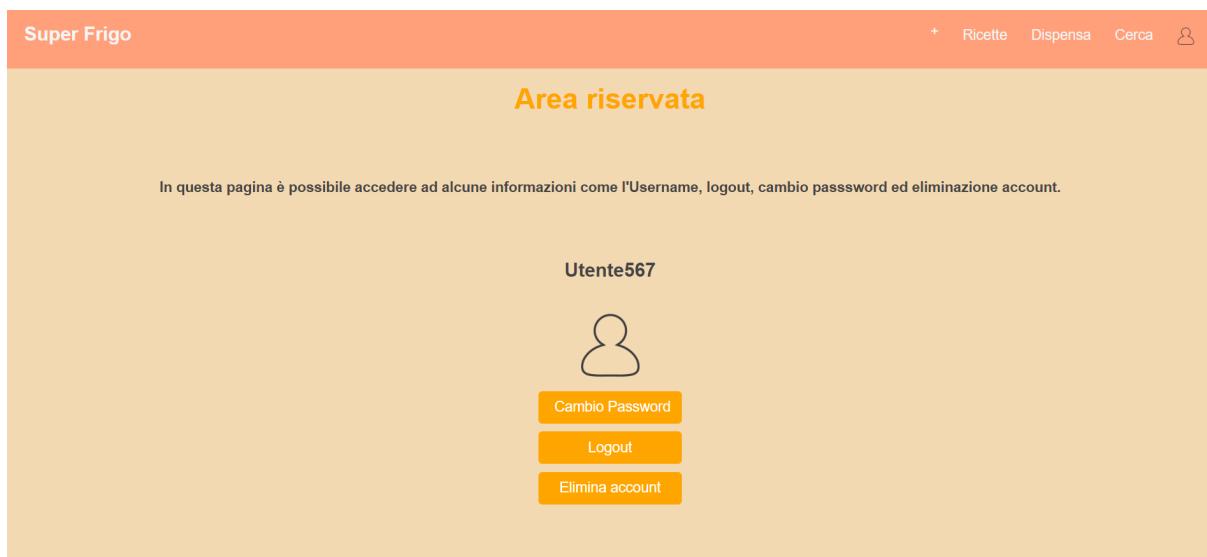
### 5.3 Registrazione

La pagina di registrazione presenta un form che permette di inserire una mail, password, conferma password e username in campi testuali. Una volta terminata la compilazione (tutti i campi sono obbligatori) l'utente può premere il bottone “Registrati” per completare la registrazione.

The screenshot shows a registration form with a light orange background. At the top center, the word "Registrazione" is written in large, bold, orange letters. Below it, a sub-instruction "Inserisci la tua mail, le password e un username che dovrai ricordarti" is displayed in a smaller, dark gray font. The form consists of four input fields: "Email:" with a white input box, "Password:" with a white input box, "Conferma Password:" with a white input box, and "Username:" with a white input box. At the bottom center of the form is a yellow rectangular button with the text "Registrati" in white. The entire form is set against a light orange background.

## 5.4 Area riservata

La pagina dell'area riservata raccoglie alcune informazioni dell'utente. La pagina mostra un'immagine di un omino stilizzato per rappresentare un utente. Subito sotto troviamo l'username dell'utente come semplici campi testuali. Infine abbiamo dei button: "Cambio password" che permette di cambiare la password dell'account, "logout" che fa sì che si esca dalla sessione dell'utente e "Elimina account" che elimina tutti i dati dell'account dai nostri database.



## 5.5 Cambio password

Presenta un form che permette di inserire vecchia password, nuova password e conferma nuova password in campi testuali. Per inviare i dati è presente il tasto conferma.

## 5.6 Cerca ingrediente

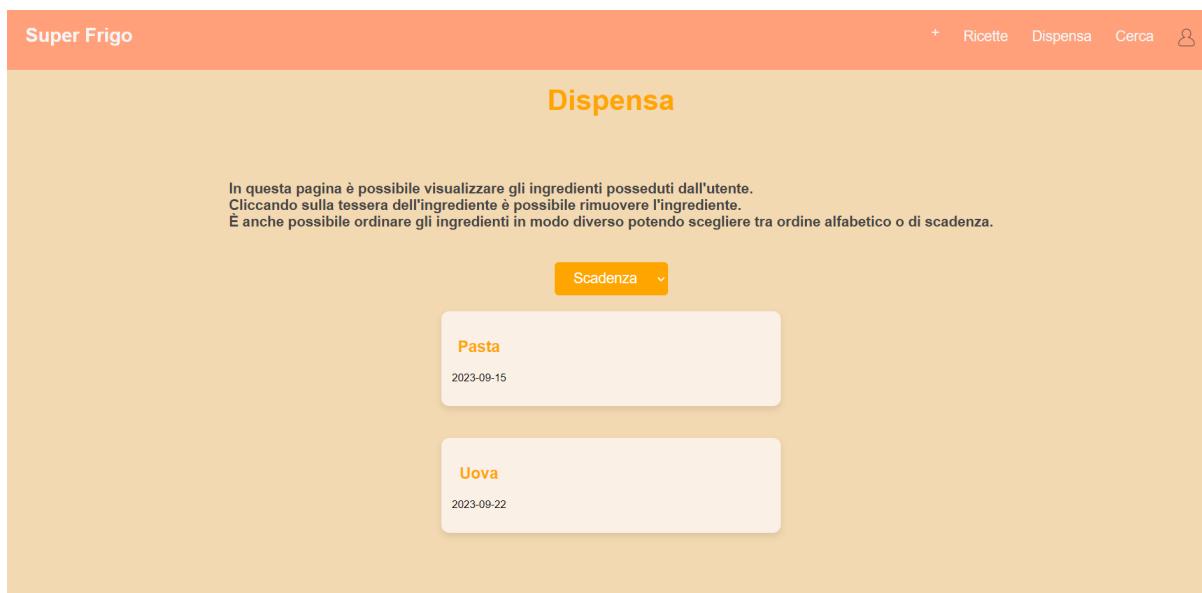
In questa pagina è presente una barra di ricerca dove è possibile cercare un ingrediente scrivendo qualcosa nell'input testuale e cliccando sulla lente di ingrandimento per effettuare la ricerca. Verranno mostrate all'utente delle card ingrediente (5.8.1) per cibi che corrispondono alla ricerca fatta. Cliccando su una card è possibile passare alla pagina 5.7.

## 5.7 Aggiungi ingrediente

Questa pagina mostra il nome dell'ingrediente che si sta aggiungendo. Poi è presente un campo quantità che permette di definire il numero di ingredienti da aggiungere. Infine mostra anche la scadenza dell'ingrediente con un campo data. Una volta inseriti i campi necessari l'utente può premere un tasto “Aggiungi” per salvare le modifiche.

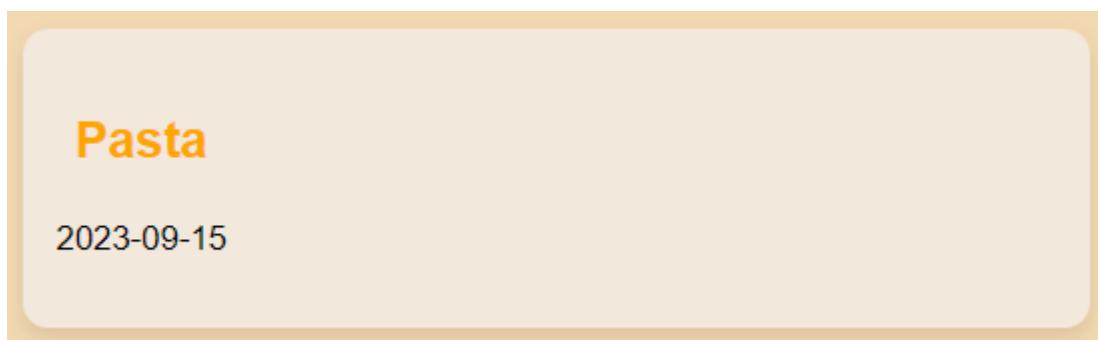
## 5.8 Dispensa

La pagina Dispensa contiene solamente una lista di ingredienti mostrati in una card. Se non è presente alcun ingrediente verrà mostrata la scritta “Non è presente alcun ingrediente”. È possibile rimuovere un ingrediente cliccando su una card e cliccando “ok” sull'avviso mostrato, altrimenti “annulla”. È anche possibile selezionare un ordinamento per gli ingredienti mostrati. Infatti premendo sul menù a tendina si può scegliere tra ordine alfabetico o di scadenza degli ingredienti.



### 5.8.1 Card ingrediente

Questa card rettangolare mostra solo nome e scadenza dell'ingrediente nella dispensa come campo testuale.



## 5.9 Ricette

In questa pagina è possibile visualizzare le prime 10 ricette che puoi cucinare con il minor numero di ingredienti mancanti. Le ricette sono mostrate all'interno di card. È possibile cliccare sulla card per avere informazioni su quella ricetta (5.8).

The screenshot shows a mobile application interface for 'Super Frigo'. At the top, there is a navigation bar with the title 'Super Frigo' on the left and icons for '+', 'Ricette', 'Dispensa', 'Cerca', and a user profile icon on the right. Below the navigation bar, the main content area has a light orange background. The title 'Le nostre migliori ricette' is centered at the top of this area. A descriptive text below it states: 'In questa pagina è possibile visualizzare le prime 10 ricette che puoi cucinare con il minor numero di ingredienti mancanti. Clicca su una tessera per ottenere più dettagli della ricetta.' Two rectangular cards are displayed, each containing a small image of a dish and its name. The first card, titled 'Uova sode', shows a bowl of soft-boiled eggs with salad. The second card, titled 'Uova alla coque', shows three eggs in colorful egg cups (blue, white, yellow) with their yolks partially broken.

### 5.9.1 Card ricetta

Questa card rettangolare mostra il nome della ricetta come campo testuale e la foto.



## 5.10 Info ricetta

In questa pagina vengono mostrate più informazioni della ricetta come gli ingredienti che la la compongono (mostrati come 5.8.1). È possibile cliccare sulla card dell'ingrediente per poterlo aggiungere alla propria dispensa. È presente anche un link alla pagina di Giallozafferano per avere informazioni più specifiche sulla preparazione della ricetta.

The screenshot shows a mobile application interface for 'Super Frigo'. At the top, there's a red header bar with the text 'Super Frigo' on the left and navigation icons on the right. Below the header, the title 'Info ricetta' is displayed in bold black text. A descriptive text in Italian follows: 'Questa pagina mostra informazioni sulla ricetta, in particolare ti permette di vedere da quali ingredienti è composta e aggiungerli cliccando sulla tessera. Cliccando sul link invece si viene redirezionati alla pagina di GialloZafferano.' Below this, the name of the recipe, 'Tortellini', is shown in large, bold, orange text. Underneath the title, there is a blue underlined link: 'Visita la ricetta su GialloZafferano'. Three cards are listed below the link, each containing an ingredient name in orange text: 'Farina 0', 'Uova', and 'Lanza di maiale'. Each card has a small circular icon with a plus sign in the top-left corner, indicating it can be added to a shopping list.

## 5.11 Header

L'header è una barra arancione che mostra il nome del sito SuperF®igo (cliccando il quale si viene portati alla home page) e i collegamenti alle pagine. cliccando su “+” viene aperto un menu a tendina verso il basso che permette di accedere alle pagine di Login (5.2), Area Riservata (5.4) e About us. Gli altri tasti sono per collegarsi alla pagina Ricette(5.9), Dispensa (5.8) e Cerca ingrediente (5.6).

### 5.11.1 Header Desktop

La versione desktop dell'header posiziona la barra in testa alla pagina, nonostante lo scorrimento del contenuto la barra rimarrà fissa in alto. Il menù a tendina si apre verso il basso.



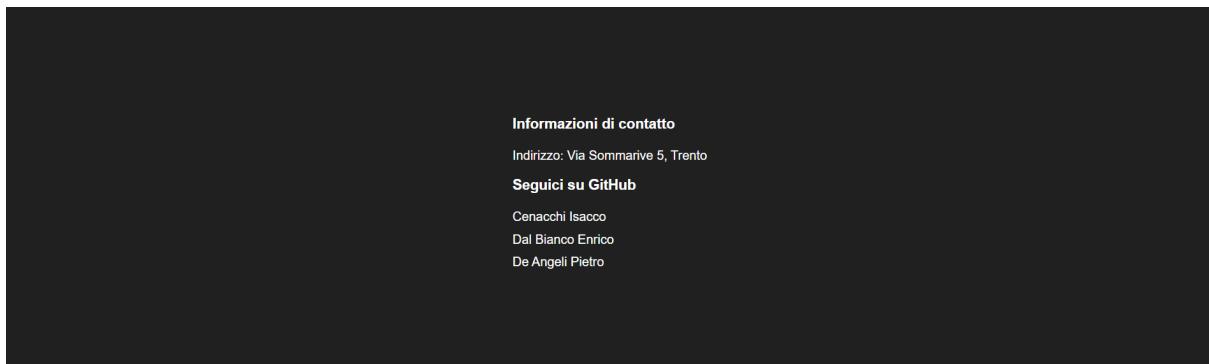
### 5.11.1 Header Mobile

La versione mobile invece posiziona la barra in fondo alla pagina e nonostante lo scorrimento del contenuto la barra rimarrà fissa in basso. Il menù a tendina si apre verso l'alto.



## 5.12 Footer

Il footer è presente solo nella versione desktop del sito, in quella mobile le informazioni presenti nel footer saranno presenti nella pagina About us (5.13) in modo più dettagliato. Consiste in una barra grigio scura con alcune informazioni sul nostro gruppo di progetto T54 di ingegneria del software scritte in formato testuale.



## 5.13 About us

About us è la pagina che dà alcune informazioni in formato testuale sul gruppo T54. È formato da 3 righe che contengono i nomi dei 3 componenti del gruppo (Link al nostro profilo GitHub) e le nostre email universitarie.

A screenshot of the "About us" page from the "Super Frigo" website. The page has a light orange header with the site name "Super Frigo" and navigation links for "Ricette", "Dispensa", "Cerca", and a user icon. The main content area has a light beige background and features the title "About us" in orange at the top. Below it, the text "T54 - Ingegneria del software" is displayed. Three horizontal lines separate the names of the team members: Cenacchi Isacco (email: isacco.cenacchi@studenti.unitn.it), Dal Bianco Enrico (email: enrico.dalbianco@studenti.unitn.it), and De Angeli Pietro (email: pietro.deangeli@studenti.unitn.it).

## 6. GitHub Repository and Deployment Info

Gli account dei membri del gruppo sono:

- Isacco Cenacchi
- Enrico Dal Bianco
- Pietro De Angeli

### 6.1 Struttura GitHub Organization

Nell'organizzazione di GitHub sono presenti 3 repositories:

- **Front End:** usata per lo sviluppo del Front End;
- **Back End:** usata per lo sviluppo del Back End;
- **Deliverables:** viene usata per la consegna dei deliverables.

Per una descrizione più dettagliata delle repository FrontEnd e BackEnd vedere [3.1 PROJECT STRUCTURE](#).

### 6.2 Deployment locale

Per avviare l'applicazione in locale per quanto riguarda il Front End noi abbiamo usato Visual Studio Code con l'estensione Live Server che funziona da web server. Possono essere usate alternative come XAMPP o qualsiasi altro web server. La cartella principale del Front End è “Front-end” che contiene index.html.

In caso si voglia avviare in locale anche il Back End noi abbiamo disabilitato CORS dal nostro browser per comodità usando l'estensione Chrome “Allow CORS: Access-Control-Allow-Origin”.

Per installare le dipendenze basta utilizzare il comando “npm install” mentre per avviare il Back End basta utilizzare il comando “node ./index.js” e il server ascolterà sulla porta 3000, bisogna inoltre specificare una variabile d'ambiente chiamata uri ovvero il link per connettersi al DB.

### 6.3 Deployment su Vercel (front-end)

L'applicazione web è stata rilasciata sulla piattaforma di hosting gratuita Vercel.com all'indirizzo <https://superfrigo.vercel.app/>

Per provare il sito è necessario premere login nel caso in cui si sia già registrati in precedenza o nel caso in cui si voglia effettuare il login come ospite. Altrimenti è necessario andare sulla pagina di registrazione.

La repository del sito contenente frontend e backend si trova su GitHub al link:  
<https://github.com/SE-T54/>

## 6.4 Deployment su railway (back-end)

Le API sono state pubblicate grazie al servizio host di Railway, opportunamente configurato per fare il deployment automatico quando viene effettuato un nuovo push sul branch master del back-end.

Il link delle API è: <https://back-end-production-d316.up.railway.app/>

Il link per la repository: <https://github.com/SE-T54/Back-end>

## 7. Testing

È stato utilizzato il framework Jest per effettuare degli unit-test automatici su alcune delle API realizzate. Gli unit-test sono collocati nella cartella test del progetto. Per effettuare i test in locale è necessario definire la variabile d'ambiente: NODE\_OPTIONS="--experimental-vm-modules".

Per testare le API basta lanciare il comando "npm test". In seguito l'elenco delle API testate con la descrizione dei test:

- POST /login

```

1  describe("GET /login", () => {
2      describe("given a valid email and password", () => {
3          let sid;
4          test("should respond with a 200 status code", async () => {
5              const response = await request(app.app).post("/login").send({
6                  email: "test@gmail.com",
7                  psw: "test"
8              });
9              expect(response.statusCode).toBe(200);
10             expect(response.headers['content-type']).toEqual(expect.stringContaining("json"));
11             expect(response.body.sessionId).toBeDefined();
12         })
13         test("login with same user", async () => {
14             const response = await request(app.app).post("/login").send({
15                 email: "test@gmail.com",
16                 psw: "test"
17             });
18             expect(response.statusCode).toBe(200);
19             expect(response.headers['content-type']).toEqual(expect.stringContaining("json"));
20             expect(response.body.sessionId).toBeDefined();
21             sid = response.body.sessionId;
22         })
23         test("GET /user for non authenticated user", async () => {
24             let response = await request(app.app).get("/user?sid="+sid).send();
25             expect(response.statusCode).toBe(200);
26             expect(response.body.guest).toBeDefined();
27             expect(response.body.username).toBeDefined();
28             expect(response.body.guest).toBe(false);
29             expect(response.body.username).toBe("test");
30         })
31     })
32     describe("given an invalid email or password", () => {
33         test("should respond with a 400 status code", async () => {
34             const response = await request(app.app).post("/login").send({
35                 email: "test@gmail.com",
36                 psw: "wrong_password"
37             });
38             expect(response.statusCode).toBe(400);
39         })
40     })
41   })
42 })
43 })

```

- POST /register

```
1  describe("POST /register", () => {
2      describe("given a valid username and password", () => {
3          test("should respond with a 200 status code", async () => {
4              const response = await request(app.app).post("/register").send({
5                  mail: "example@example.com",
6                  username: "example_username",
7                  psw: "password"
8              });
9              expect(response.statusCode).toBe(200);
10         })
11     })
12     describe("given an already used email", () => {
13         test("should respond with a 400 status code", async () => {
14             const response = await request(app.app).post("/register").send({
15                 mail: "test@gmail.com",
16                 username: "example_username",
17                 psw: "password"
18             });
19             expect(response.statusCode).toBe(400);
20             expect(response.text).toBe("email already used");
21         })
22     })
23     describe("given an invalid email", () => {
24         test("should respond with a 401 status code", async () => {
25             const response = await request(app.app).post("/register").send({
26                 mail: "invalid_email",
27                 username: "example_username",
28                 psw: "password"
29             });
30             expect(response.statusCode).toBe(401);
31             expect(response.text).toBe("email not valid");
32         })
33     })
34 })
```

- POST /change\_password

```
1  describe("POST /change_password", () => {
2      test("the new password is the same as the old one", async () => {
3          const login = await request(app.app).post("/login").send({
4              email: "example@example.com",
5              psw: "password"
6          });
7          sid = login.body.sessionId;
8          let response = await request(app.app).post("/change_password").send({
9              sid: sid,
10             old_psw: 'password',
11             new_psw: "password"
12         })
13         expect(response.statusCode).toBe(402);
14     })
15     test("invalid session id", async () => {
16         let response = await request(app.app).post("/change_password").send({
17             sid: -1,
18             old_psw: 'password',
19             new_psw: "password2"
20         })
21         expect(response.statusCode).toBe(403);
22     })
23     test("wrong old password", async () => {
24         let response = await request(app.app).post("/change_password").send({
25             sid: sid,
26             old_psw: 'passwor',
27             new_psw: "password"
28         })
29         expect(response.statusCode).toBe(405);
30     })
31     test("ok", async () => {
32         let response = await request(app.app).post("/change_password").send({
33             sid: sid,
34             old_psw: 'password',
35             new_psw: "password2"
36         })
37         expect(response.statusCode).toBe(200);
38     })
39 })
```

- DELETE /delete\_account

```
1  describe("DELETE /delete_account", () => {
2      describe("given valid sid", () => {
3          test("should respond with a 200 status code", async () => {
4              const response = await request(app.app).delete("/delete_account").send({
5                  sid: sid
6              });
7              expect(response.statusCode).toBe(200);
8          })
9      })
10     describe("given invalid sid", () => {
11         test("should respond with a 200 status code", async () => {
12             const response = await request(app.app).delete("/delete_account").send({
13                 sid: -1
14             });
15             expect(response.statusCode).toBe(403);
16         })
17     })
18 })
```

- GET /guest\_registration

```
1  describe("GET /guest_registration", () => {
2      test("should respond with a 200 status code and a new SID", async () => {
3          const response = await request(app.app).get("/guest_registration").send();
4          expect(response.statusCode).toBe(200);
5          expect(response.text).toBeDefined();
6          sid = response.text;
7      })
8      test("check if is a guest", async () => {
9          const check = await request(app.app).get("/user?sid="+sid).send();
10         expect(check.statusCode).toBe(200);
11         expect(check.body.guest).toBeDefined();
12         expect(check.body.guest).toBe(true);
13     })
14 })
```

- GET /recipes

```
1 describe("GET /recipes", () => {
2     test("valid SID", async () => {
3         const response = await request(app.app).get("/recipes?sid="+sid).send();
4         expect(response.statusCode).toBe(200);
5         expect(response.body.length).toBe(10);
6     })
7     test("invalid SID", async () => {
8         const response = await request(app.app).get("/recipes?sid=-1").send();
9         expect(response.statusCode).toBe(403);
10    })
11})
```

- GET /ingredients

```
1 describe("GET /ingredients", () => {
2     test("invalid sid", async () => {
3         let response = await request(app.app).get("/ingredients?sid="-1).send();
4         expect(response.statusCode).toBe(403);
5     })
6 })
```

- POST /add

```
1  describe("POST /add", () => {
2      test("valid SID and valid ingredient", async () => {
3          const response = await request(app.app).post("/add").send({
4              sid: sid,
5              ingredient: {
6                  name: "test",
7                  expiration: "01/01/2100",
8                  quantity: 1
9              }
10         })
11         expect(response.statusCode).toBe(200);
12         expect(response.text).toBe("ok");
13         let ingredients = (await request(app.app).get("/ingredients?sid="+sid).send()).body;
14         expect(ingredients.some(x => x.name == "test")).toBe(true);
15     })
16     test("invalid SID", async () => {
17         const response = await request(app.app).post("/add").send({
18             sid: -1,
19             ingredient: {
20                 name: "test",
21                 expiration: "01/01/2100",
22                 quantity: 1
23             }
24         })
25         expect(response.statusCode).toBe(403);
26     })
27     test("invalid ingredient format", async () => {
28         const response = await request(app.app).post("/add").send({
29             sid: sid,
30             ingredient: {
31                 noname: "aaa",
32                 noexpiration: "01/01/2100"
33             }
34         })
35         expect(response.statusCode).toBe(401);
36     })
37 })
```

- DELETE /remove

```

 1  describe("DELETE /remove", () => {
 2      test("Valid SID and valid index", async () => {
 3          for(let i=0;i<3;i++)
 4              await request(app.app).post("/add").send({sid:sid,ingredient:{name:"test"+i,expiration:"01/01/2100",quantity:1}});
 5
 6          const response = await request(app.app).delete("/remove").send({
 7              sid: sid,
 8              ingredient: 0
 9          });
10          expect(response.statusCode).toBe(200);
11          let ingredients = (await request(app.app).get("/ingredients?sid="+sid).send()).body;
12          expect(ingredients.length).toBe(3);
13      })
14      test("invalid SID and valid index", async () => {
15          const response = await request(app.app).delete("/remove").send({
16              sid: -1,
17              ingredient: 0
18          });
19          expect(response.statusCode).toBe(403);
20      })
21      test("valid SID and invalid index", async () => {
22          const response = await request(app.app).delete("/remove").send({
23              sid: sid,
24              ingredient: 3
25          });
26          expect(response.statusCode).toBe(402);
27      })
28      test("storage not created yet", async () => {
29          const login = await request(app.app).get("/guest_registration").send();
30          let tsid = login.text;
31          const response = await request(app.app).delete("/remove").send({
32              sid: tsid,
33              ingredient: 0
34          });
35          expect(response.statusCode).toBe(400);
36      })
37  })

```

app.js	<div style="width: 90.98%;"></div>	90.98%	212/233	84.28%	59/70	89.28%	25/28	92.27%	203/220
--------	------------------------------------	--------	---------	--------	-------	--------	-------	--------	---------

Come si può vedere dall'immagine sopra i test coprono il 92.27% delle righe di codice, le righe mancanti sono errori che non dovrebbero mai verificarsi e alcune cose che non hanno bisogno di una verifica.