

Documentazione

Yummy!



Indice

| | |
|---|----------|
| 1 Introduzione | 2 |
| 2 Pacchetti utilizzati | 3 |
| 3 Istruzioni per eseguire la build | 4 |
| 4 Architettura | 5 |
| 5 Routing | 6 |
| 6 Views | 7 |
| 6.1 App | 7 |
| 6.2 Home | 8 |
| 6.3 Recipes | 9 |
| 6.3.1 RecipesCardsGrid | 9 |
| 6.3.2 RecipesTable | 10 |
| 6.4 About us | 11 |
| 6.5 RecipeDetail | 13 |
| 6.5.1 RecipeCategory | 14 |
| 6.5.2 RecipeType | 14 |
| 6.6 Default | 15 |
| 6.7 Profile | 16 |

1 Introduzione

Il progetto **Yummy!** è relativo a un'applicazione web che è un ricettario online, nel quale è possibile accedere a 100 ricette e ai relativi dettagli.

Per ognuna di esse si hanno diverse informazioni di dettaglio: tempo di preparazione, numero di persone, tipologia di piatto e di dieta (tra vegana, vegetariana e senza glutine), health score, lista degli ingredienti e step per la preparazione della ricetta.

All'interno della lista di ricette è possibile filtrare sulla base delle diete presenti e ottenere due visualizzazioni alternative, sotto forma di elenco di card o in formato tabellare.



2 Pacchetti utilizzati



Per creare l'applicazione sono stati usati i seguenti pacchetti **npm**:

- **reactstrap**: dà la possibilità di utilizzare bootstrap con React
`npm install --save bootstrap reactstrap`
- **react-router-dom**: permette di creare un sistema di routing all'interno di un'applicazione web
`npm install --save react-router-dom`
- **clsx**: utilizzato per gestire il CSS nei rendering condizionali
`npm install --save clsx`
- **lottie-web**: Lottie è una libreria mobile che permette di inserire animazioni json in pagine web con l'utilizzo di Bodymovin
`npm install lottie-web`
- **react-icons**: fornisce dei set di icone
`npm install --save clsx`
- **firebase**: permette di gestire l'autenticazione
`npm install firebase`
- **@firebase/firestore**: permette di salvare in un database i dati degli utenti. La struttura di questo database è: collections > documents. Ogni document può contenere diversi campi.
`npm install @firebase/firestore`
- **@mui/material** e **@mui/icons-material**: permettono di creare componenti personalizzate
`npm install @mui/material`
`npm install @mui/icons-material`

3 Istruzioni per eseguire la build

E' stata creata una build eseguibile direttamente attraverso i seguenti step:

1. Utilizzare il comando cd all'interno del Prompt per posizionarsi nella cartella contenente la cartella build
2. Eseguire i seguenti comandi:
 - a. npm install -g serve
 - b. npx serve -s build
3. Incollare il localhost address sul browser

4 Architettura

Per lo sviluppo di **Yummy!** è stata scelta l'API **Spoonacular**, a cui sono state fatte diverse chiamate, in particolare per ottenere la lista degli ingredienti e le kilocalorie della ricetta presenti nella pagina di dettaglio di ogni ricetta, mentre le altre informazioni presenti nella stessa pagina sono state ricavate dal file food.json. Tale file JSON è stato scaricato a seguito di una chiamata alla API con l'utilizzo di **Postman**, in modo tale da ottenere più velocemente i dati e limitare il numero di chiamate all'API stessa, dato che nella sua versione gratuita il numero di chiamate consentite è limitato.

L'architettura dell'applicazione si basa su tre pagine:

- Home
- Recipes
- About us

Dalla pagina Recipes è, ovviamente, possibile accedere ai dettagli di ogni singola ricetta presente, mentre dalla Home è possibile accedere ai dettagli di 6 ricette che sono proposte randomicamente.

In caso di errori o di richiesta di una pagina inesistente verrà visualizzata una pagina di errore.



POSTMAN

5 Routing

Il sistema di routing, come detto in precedenza, è stato gestito tramite **react-router-dom**.

Sono state create 6 Route, a cui vengono associati i seguenti componenti:

1. **Home** a cui corrisponde il path: /
2. **Recipes** a cui corrisponde il path: /recipes
3. **About** a cui corrisponde il path: /about
4. **Profile** a cui corrisponde il path: /profile
5. **Wrapper** a cui corrisponde il path: /recipes/:number
6. **Default** a cui corrisponde il path: /*

Le ultime due Route sono state inserite al fine di gestire correttamente il caso in cui l'utente inserisca un path non corretto o un identificatore non presente.

Il componente Wrapper permette di controllare che l'identificatore inserito dall'utente esista e, solo in quel caso, indirizzarlo alla pagina di dettaglio della ricetta corrispondente, altrimenti verrà mostrata una pagina d'errore.

Il componente Default invece gestisce la richiesta di accesso ad un path non valido perché porta ad una pagina inesistente nel sito.

6 Views

6.1 App

Questa view contiene la gestione del routing che fa variare, a seconda del path, la pagina children di **MainTemplate**. Con questo componente è infatti possibile avere Header e Footer sempre visibili sulla pagina e far variare il contenuto interno che dipende dalla pagina in cui ci si trova.

Il componente **Header** si occupa di gestire la navbar, in particolare su schermi più piccoli del breakpoint md (ovvero 768px) si ha un hamburger menù creato tramite i componenti Collapse e Nav di reactstrap, mentre in caso di schermi più grandi viene visualizzata una navbar orizzontale classica. All'interno della navbar ogni NavItem è stato costruito con NavLink.



Figura 5.1.1: Header

Il componente **Footer** contiene una navigazione, dove ogni elemento è stato costruito tramite il componente NavLink, inoltre sono stati inseriti il logo dell'Università degli Studi di Milano-Bicocca e il logo di e-learning con i relativi link. È stato inoltre inserito il componente **ScrollButton**, che permette di far comparire nella pagina in cui ci si trova un bottone che riporta all'inizio della pagina dopo che l'utente ha effettuato uno scroll verso il basso, facilitandogli così la risalita.



Figura 6.1.2: Footer e ScrollButton

6.2 Home

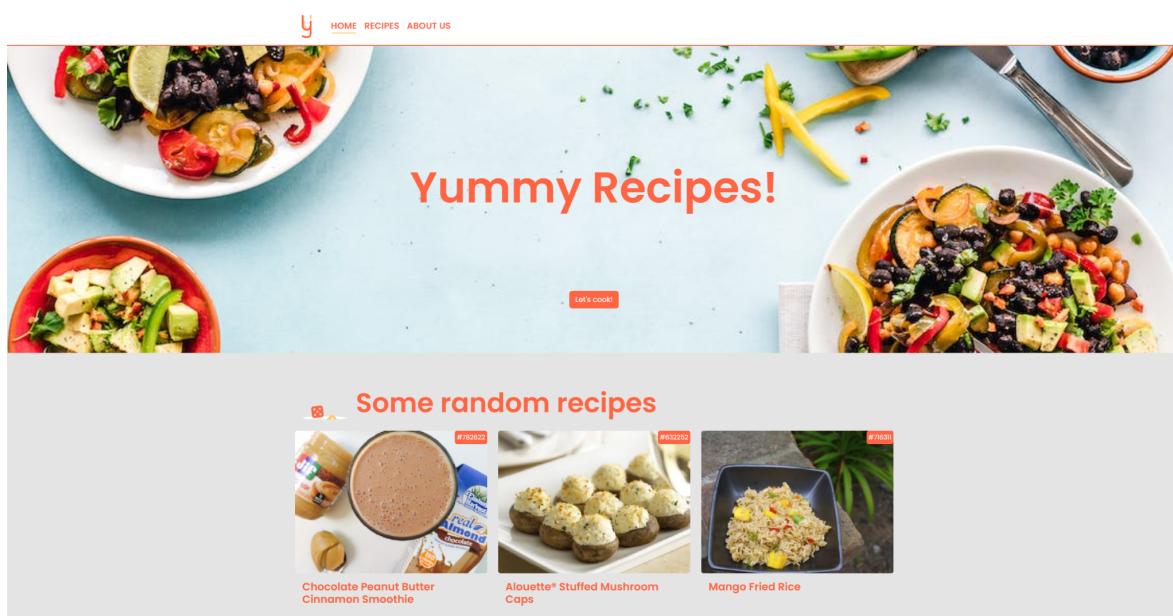


Figura 6.2.1: View Home

Questa view mostra 6 ricette estratte randomicamente dalla JSON statico contenente la lista delle ricette, questa funzionalità è stata sviluppata tramite l'utilizzo di `Math.floor()` e `Math.random()` per la generazione di numeri casuali tra 0 e 99.

L'animazione dei dadi viene caricata nella home tramite la funzione `loadAnimation()` richiamata all'interno dell'hook `useEffect(function, [])` che viene eseguito una sola volta.

La funzione `loadAnimation()` carica l'animazione Lottie contenuta nel file `dado.json` dalla cartella degli assets statici e ne definisce le caratteristiche (container, renderer, loop e autoplay).

6.3 Recipes

Questa view gestisce la visualizzazione alternativa della lista di ricette sotto forma di grid o table attraverso il rendering condizionale, che si basa sul cambiamento dello stato `displayGrid`.

Al click sull'icona a forma di griglia `displayGrid` avrà valore true e porterà al rendering del componente **RecipesCardsGrid**, mentre al click sull'icona a forma di tabella `displayGrid` assumerà valore false e porterà al rendering del componente **RecipesTable**.

Nel rendering condizionale viene usato anche `clsx` per gestire lo stile sulla base del valore dello stato.

6.3.1 RecipesCardsGrid

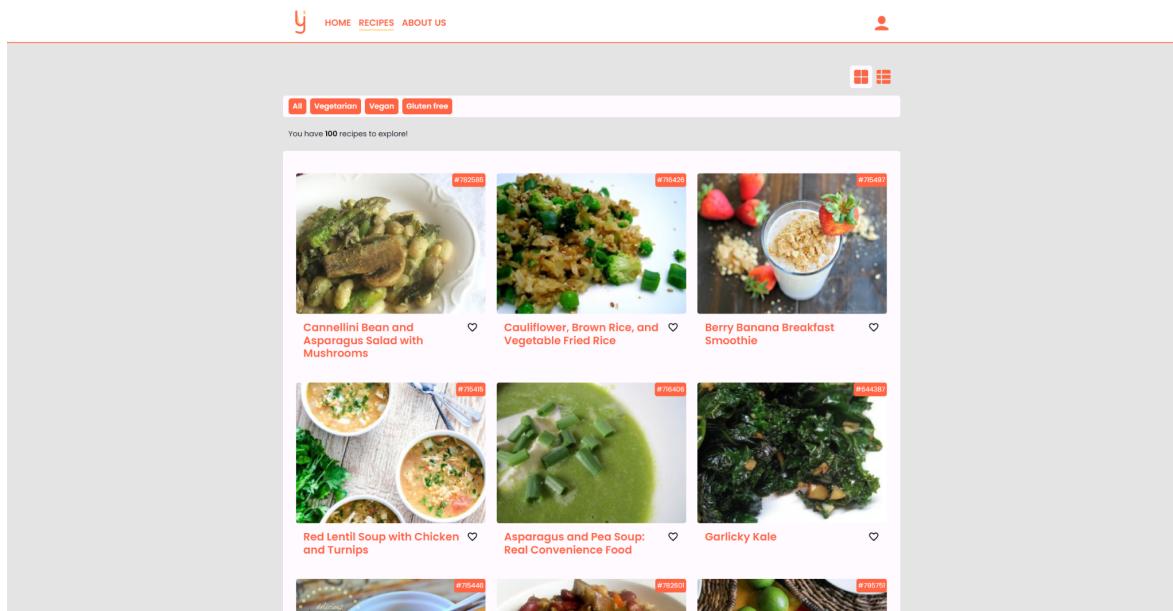


Figura 6.3.1.1: View Recipes, caso Grid

Questo componente gestisce la visualizzazione sotto forma di griglia di tutte le ricette presenti all'interno del JSON statico `food.json`.

La griglia è composta da card realizzate attraverso il componente **RecipeCard**. L'immagine di ogni RecipeCard funge da collegamento con la pagina di dettaglio della ricetta corrispondente grazie all'utilizzo del componente `NavLink` e la relativa `props to`.

6.3.2 RecipesTable

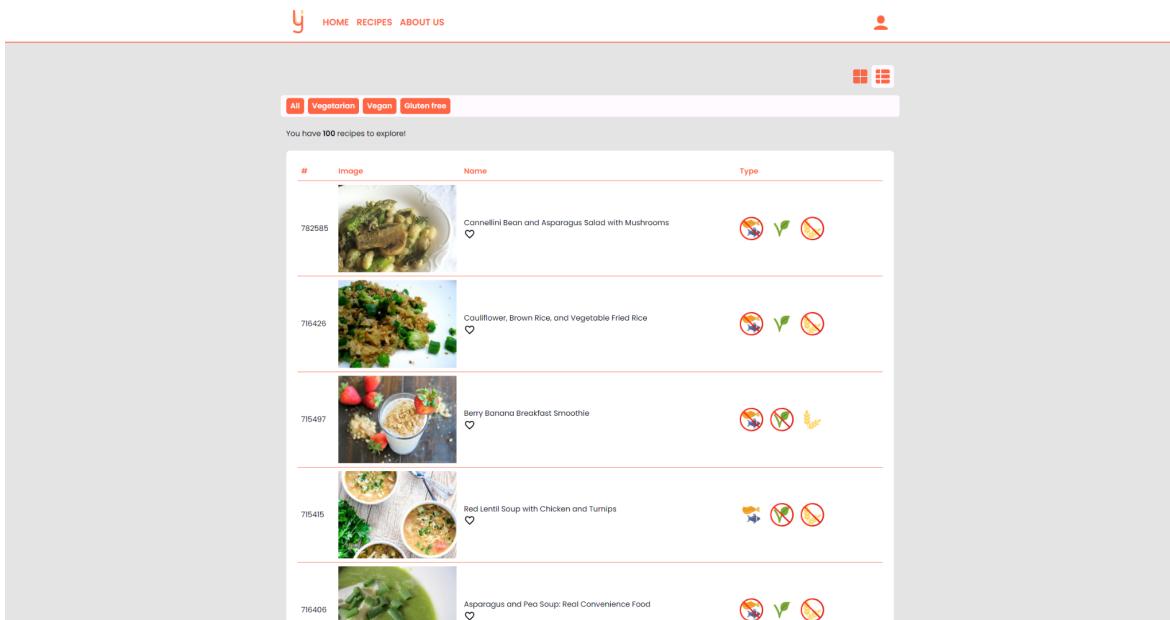


Figura 6.3.2.1: View Recipes, caso Table

Questo componente gestisce la visualizzazione sotto forma di tabella di tutte le ricette presenti all'interno del JSON statico food.json.

In particolare per ogni ricetta viene costruita una riga contenente l'identificatore, l'immagine, il nome, le icone corrispondenti alla tipologia di dieta e un link per accedere alla pagina di dettaglio della ricetta stessa (questo link è visualizzato solo quando facciamo hover sulla riga di una ricetta).

Entrambi i componenti RecipesCardsGrid e RecipesTable contengono un sistema che permette di filtrare le ricette visualizzate sulla base del tipo di dieta selezionato e un sistema per salvare le ricette preferite.

È possibile filtrare rispetto alla dieta vegana, a quella vegetariana e a quella senza glutine ed è ovviamente anche possibile visualizzare tutte le ricette senza alcun filtro cliccando su "All".

I bottoni per il filtraggio vengono creati e gestiti all'interno del componente **Filter**, a cui viene passata la funzione `filterItem()` che effettua il filtraggio dell'elenco di ricette in base alla dieta selezionata, la funzione `setItem()` per modificare lo stato e l'array `menuItems` che contiene tutte le possibili diete così da poter generare correttamente i bottoni corrispondenti.

Il vero e proprio filtraggio viene, invece, gestito all'interno di RecipesCardsGrid e RecipesTable attraverso la funzione `filterItem()` che aggiorna lo stato `item`, ovvero la lista ogni volta che si clicca su uno dei bottoni.

Per salvare le ricette preferite dell'utente viene usato il componente **CheckboxRecipe**, che ha come props l'utente e l'id della ricetta.

Nella view **Recipes** viene salvato l'utente corrente in uno stato user, quest'ultimo viene settato alla chiamata della funzione `onAuthStateChanged` contenuta a sua volta in una `useEffect(function, [])`.

Per gestire il fatto che sia stata selezionata o meno la ricetta viene utilizzato lo stato checked, che inizialmente per ogni ricetta ha valore false. Viene eseguita inizialmente una `useEffect(function, [])` per controllare se la ricetta è stata salvata e, in tal caso, cambierà il valore di checked a true.

La checkbox è stata realizzata tramite il componente **checkbox** di `@mui/material` e `@mui/icons-material`.

La funzione `handleCheck` viene invocata quando si ha un cambiamento nella checkbox. Al suo interno viene gestito lo stato checked in modo coerente con la visualizzazione della checkbox.

Se la checkbox è checked, la ricetta viene aggiunta all'array dei preferiti tramite la funzione `addFav`, se invece diventa unchecked la ricetta viene rimossa tramite la funzione `removeFav`.

Il documento viene aggiornato utilizzando le funzioni `updateDoc`, `arrayUnion` e `arrayRemove`.

Se un utente aggiunge per la prima volta ai preferiti una ricetta viene creato un nuovo documento contenente il suo uid e l'array delle ricette salvate.

6.4 About us

The screenshot shows the 'About Us' page of the Yummy website. At the top, there's a navigation bar with a logo, 'HOME', 'RECIPES', and 'ABOUT US' (which is underlined). Below the navigation, there's a section titled 'Il progetto Yummy' with a detailed description of the project's purpose, target audience, and features like a health score and ingredient lists. This section is enclosed in a light gray box. Below this, there's a heading 'Chi siamo?' followed by two profiles: Valeria Fraio and Sofia Damaso, each with a circular photo, their names, and a short quote at the bottom.

Il progetto Yummy

Questo progetto nasce per il corso di [Applicazioni Web](#) e il suo obiettivo è quello di fornire un ricettario di piatti particolari, originali e accessibili online. Il sito offre una vasta selezione di ricette per soddisfare i gusti e le esigenze di tutti gli utenti. Sei uno persona attento alla linea? Con **Yummy** puoi conteggiare le calorie e ottenere l'health score della ricetta, per aiutarti a mangiare in modo sano ed equilibrato. **Yummy** si adatta alle esigenze di tutti, proporviendo una vasta scelta di piatti vegani, vegetariani e gluten free. In ogni ricetta puoi trovare, oltre alla lista degli ingredienti, un prezioso tutorial passo-passo per la preparazione. Tutte le informazioni sono reperibili tramite chiaviote dell'[API Spoonacular](#). **Yummy** è il tuo ricettario online completo e facile da utilizzare, che ti permetterà di scoprire nuovi sapori e di sperimentare in cucina, senza rinunciare alla salute e al benessere.

Chi siamo?

Valeria Fraio
"In fondo imparare a programmare in un linguaggio è come imparare una nuova lingua, solo che la

Sofia Damaso
"Questa vita è l'algoritmo di un programma che ora è vero"

Figura 6.4.1: View About Us

Questa view riporta una breve descrizione e due card di presentazione del team Yummy!.

Queste card contengono un'immagine, una citazione, la laurea conseguita, gli hobby e le informazioni di contatto delle sviluppatri.

In fondo alla pagina sono inoltre presenti due button, uno permette di scaricare il file della documentazione, mentre l'altro permette di accedere al repository Github contenente il codice sorgente del progetto.

6.5 RecipeDetail

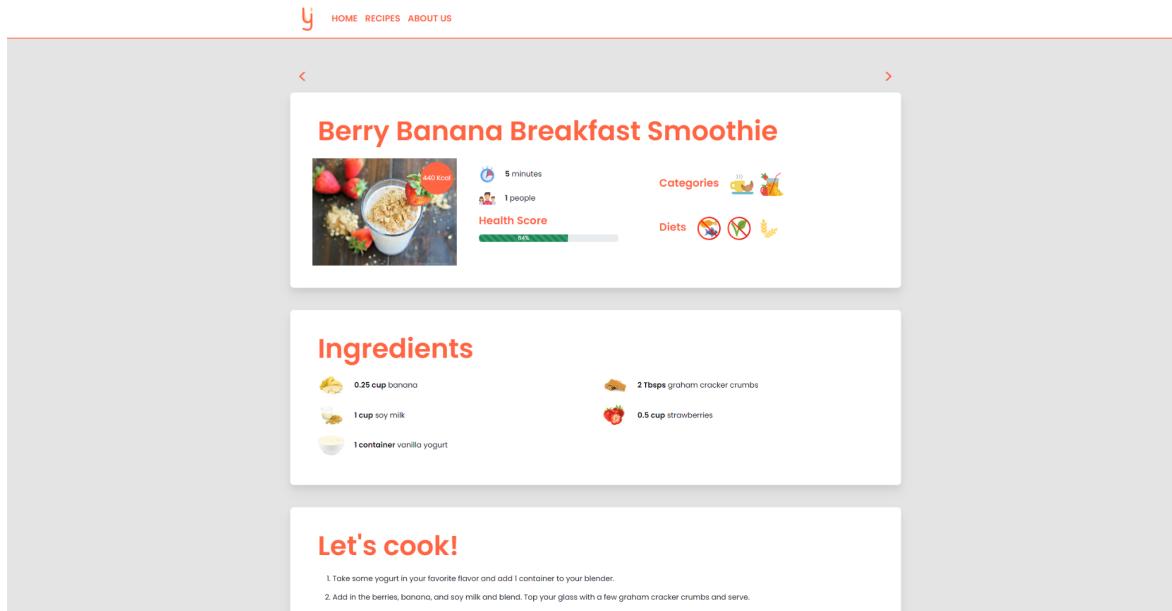


Figura 6.5.1: View RecipeDetail per la ricetta numero 715497

Questa view contiene le informazioni di dettaglio della ricetta, utilizza due fetch per ottenere rispettivamente il valore delle kilocalorie e la lista degli ingredienti dall'API di Spoonacular, mentre tutti gli altri dati presenti nella pagina (numero di persone, tempo di preparazione, gli step della ricetta e l'health score) vengono prelevati dal JSON statico.

Per visualizzare la percentuale di health score è stato utilizzato il componente Progress.

Le due fetch sopracitate sono state inserite all'interno di due useEffect(), esse modificano il valore dello stato corrispondente, se il componente è già stato montato, tramite il setter dello stato.

Entrambe le useEffect() vengono eseguite al variare dell'identificativo della ricetta.

All'inizio di ogni pagina di dettaglio è presente una piccola navigazione che permette di passare alla ricetta successiva oppure alla precedente tramite il componente NavLink e la relativa props to.

Sono stati inseriti i relativi controlli per controllare che esista la ricetta a cui vogliamo passare, poiché la prima ricetta non possiede una precedente e l'ultima non possiede una ricetta successiva.

Per quanto riguarda la categoria della ricetta (ad esempio main dish o side dish) e la tipologia di dieta vengono usati rispettivamente i componenti

RecipeCategory e **RecipeType**.

In entrambi i componenti per far comprendere meglio il significato delle icone è stato utilizzato un UncontrolledTooltip con posizionamento bottom che permette di inserire una label quando si fa hover sull'icona.

6.5.1 RecipeCategory

Questo componente riceve come props l'identificativo della ricetta, in modo tale da poter accedere ai suoi dettagli nel file JSON.

I valori del file food.json utilizzati sono contenuti in un array dishTypes che viene utilizzato nel componente per creare un oggetto contenente coppie chiave-valore, dove la chiave è il nome della categoria e il valore è un booleano, ciò permette di fare un rendering condizionale grazie all'operatore ternario, al fine di visualizzare le icone che rappresentano le categorie della ricetta.

Le possibili categorie per ogni piatto sono le seguenti:

- Appetizer
- Breakfast
- Dessert
- Drink o Beverage
- Main Dish
- Side Dish

6.5.2 RecipeType

Questo componente riceve come props l'identificativo della ricetta, in modo tale da poter accedere ai suoi dettagli nel file JSON.

I valori del file food.json utilizzati sono coppie chiave-valore che vengono utilizzate nel componente per creare un oggetto in maniera similare a quello presente in RecipeCategory, ciò permette di fare un rendering condizionale grazie all'operatore ternario, al fine di visualizzare le icone che rappresentano le diete (vegana, vegetariana e senza glutine).

Per ogni dieta è presente un'icona visualizzata nel caso in cui la ricetta appartenga a quella dieta e un'altra nel caso opposto.

6.6 Default

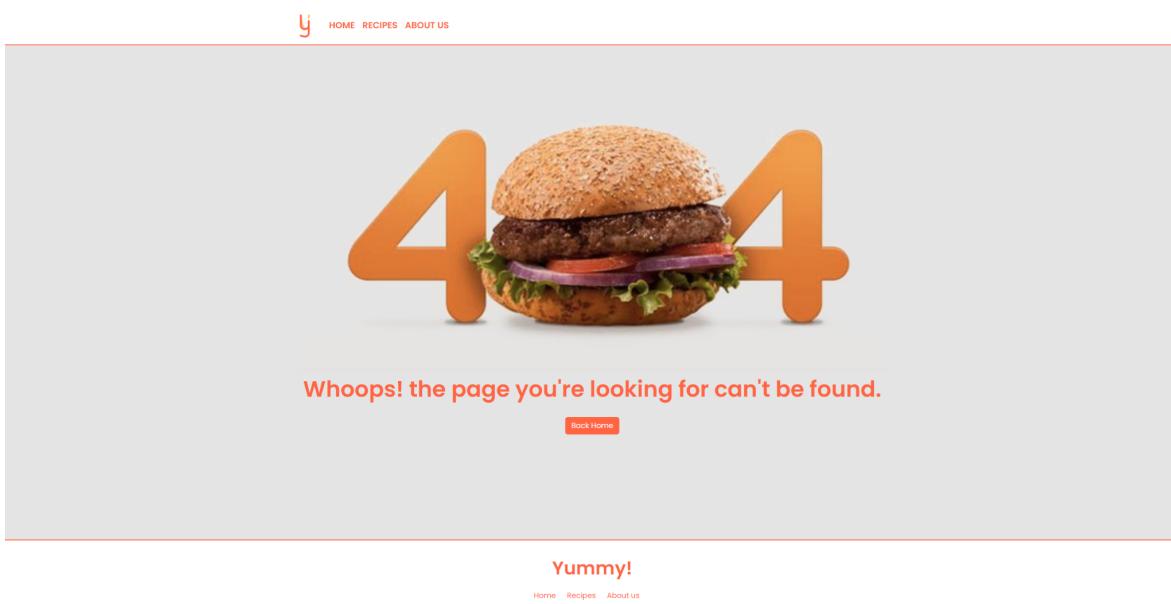


Figura 6.6.1: Pagina di errore costruita dal componente Default

Il componente **Default** si occupa della costruzione della pagina visualizzata in caso di errore, nei casi descritti nella sezione relativa al Routing.
Questo componente costruisce la pagina di errore 404 con un'immagine, il testo relativo all'errore e un bottone che riporta alla Home tramite il componente NavLink e la props to.

6.7 Profile

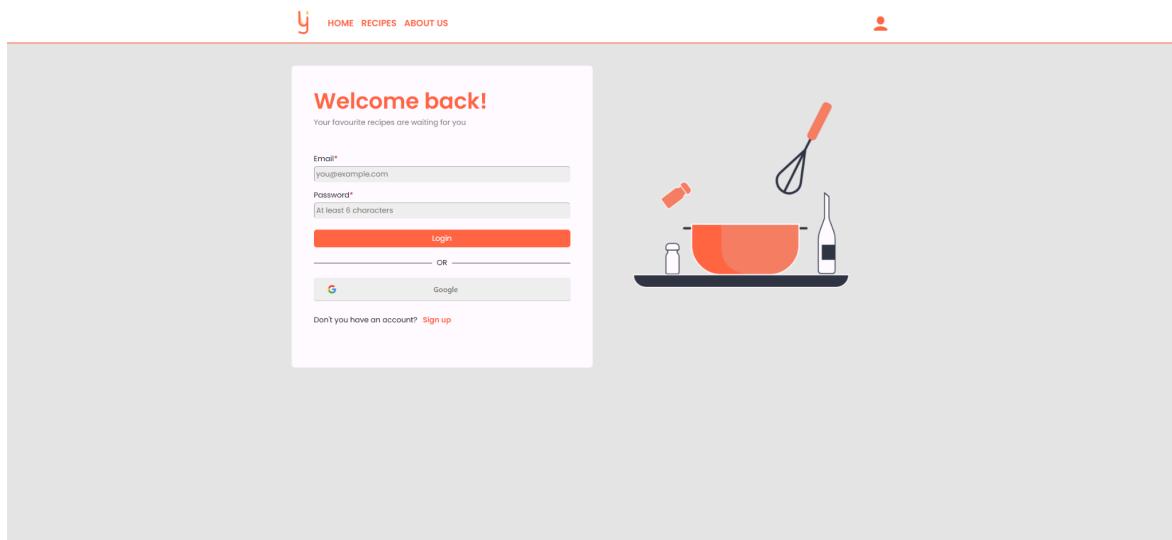


Figura 6.7.1: View Profile, visualizzazione iniziale

Questa view rappresenta il profilo dell’utente con i suoi dati e le ricette salvate una volta che si è autenticato. Per far ciò abbiamo utilizzato Firebase e Firestore. Nel file `firebase.js` viene inizializzato Firebase con i config forniti dal sito stesso e vengono esportati `auth`, `signInWithGoogle` e `firestore`, che ci permettono di utilizzare l’autenticazione, fare il login tramite profilo Google e l’utilizzo del database Firestore. All’interno di `signInWithGoogle` viene utilizzata la funzione `signInWithPopup` di Firebase al fine di ottenere un popup in cui l’utente può inserire le proprie informazioni.

All’interno della view l’utente può registrarsi e/o fare il login, con la propria email o con un account Google. Per registrarsi è necessario compilare un form che appare una volta cliccato il bottone “Sign up”. L’apertura e la chiusura del Collapsible sono gestite tramite lo stato `isOpen`, il quale ha inizialmente valore false.

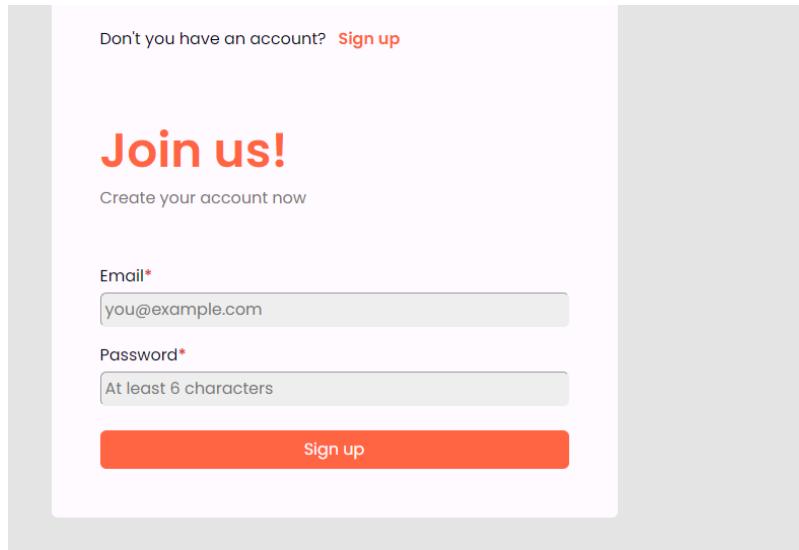


Figura 6.7.1: View Profile, visualizzazione del Sign up

Quando un utente si registra, sia tramite Google che tramite email, vengono salvati i suoi dati (username, email e immagine del profilo) nel localStorage attraverso la funzione `setItem`. Nel caso l'immagine del profilo non sia presente o sia corrotta viene visualizzata un'immagine di default tramite la funzione `profileDefaultImage`, che viene importata dal file `utility.js`.

Nella parte destra della pagina appare un'animazione che è stata inserita analogamente a quella nella Home e viene gestita tramite la funzione `loadAnimation()` richiamata all'interno dell'hook `useEffect(function, [])`, che viene eseguito una sola volta. Questa funzione carica l'animazione contenuta nel file `cooking.json` dalla cartella degli assets statici e ne definisce le caratteristiche (`container`, `renderer`, `loop` e `autoplay`). Nella stessa `useEffect` è stata inserita la chiamata alla funzione `onAuthStateChanged` di Firebase, alla quale viene passata l'oggetto `auth` per l'autenticazione e l'utente corrente.

Ogni volta che si effettua il submit in uno dei due form presenti (login o registrazione) i campi vengono svuotati attraverso la funzione `clearFields`.

Al fine di gestire in maniera ottimale l'autenticazione sono state usate 3 diverse funzioni asincrone:

- **Register:** aspetta che venga eseguita la funzione `createUserWithEmailAndPassword` che crea un utente utilizzando email e password tramite. Tutte le operazioni sono inserite all'interno di un `try & catch`.

- **Login:** analogamente alla registrazione, aspetta che venga eseguita la funzione `signInWithEmailAndPassword` che permette a un utente di accedere con email e password. Anche in questo caso le operazioni svolte sono inserite all'interno di un `try & catch`.
- **Logout:** attende il risultato della funzione `signOut`.

Le tre funzioni `createUserWithEmailAndPassword`,
`signInWithEmailAndPassword` e `signOut` sono state importate da
`firebase/auth`.

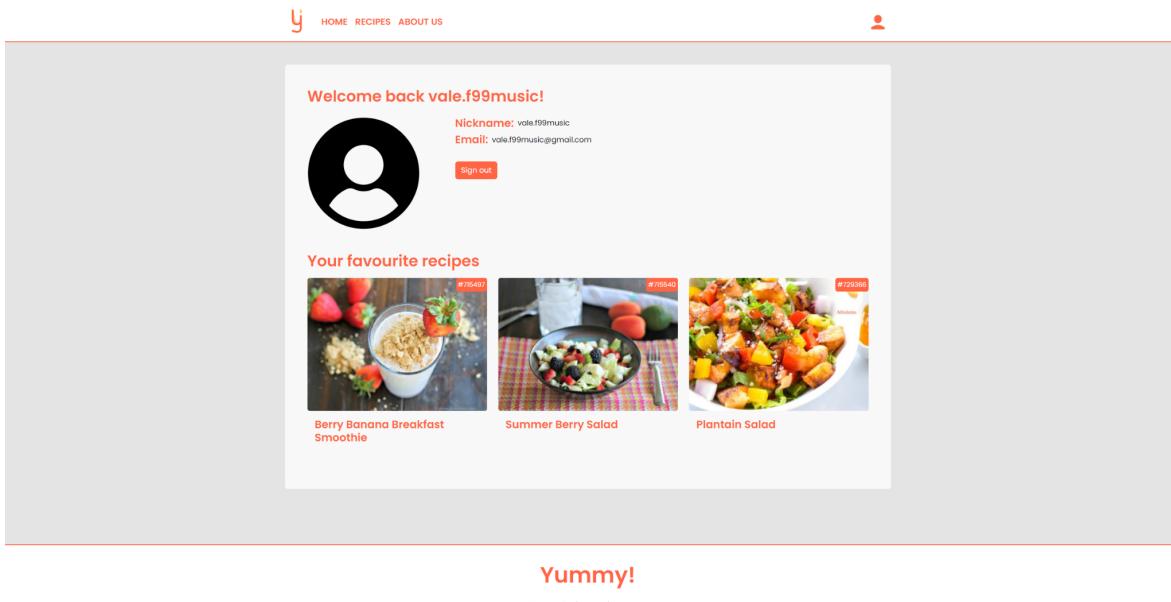


Figura 6.7.2: View Profile, visualizzazione del profilo con ricette salvate

Quando l'utente si è registrato o ha fatto il login viene aggiornato lo stato user, quindi viene richiamata la `useEffect(function, [user])` che controlla il database, in particolare la collection `users`, per capire se l'utente ha già ricette salvate e, in tal caso, le visualizza. Per far ciò vengono comparati tutti gli uid contenuti nei documenti della collection con quello dell'utente che attualmente è loggato. Gli uid utilizzati sono generati automaticamente da Firebase in fase di autenticazione.

Per mostrare le ricette salvate nei preferiti, se presenti, viene utilizzato prima il metodo `filter` per filtrare la lista di tutte le ricette sulla base di quelle presenti nel database e successivamente con il metodo `map` vengono create le card vere e proprie da visualizzare (tramite il componente **RecipeCard**).

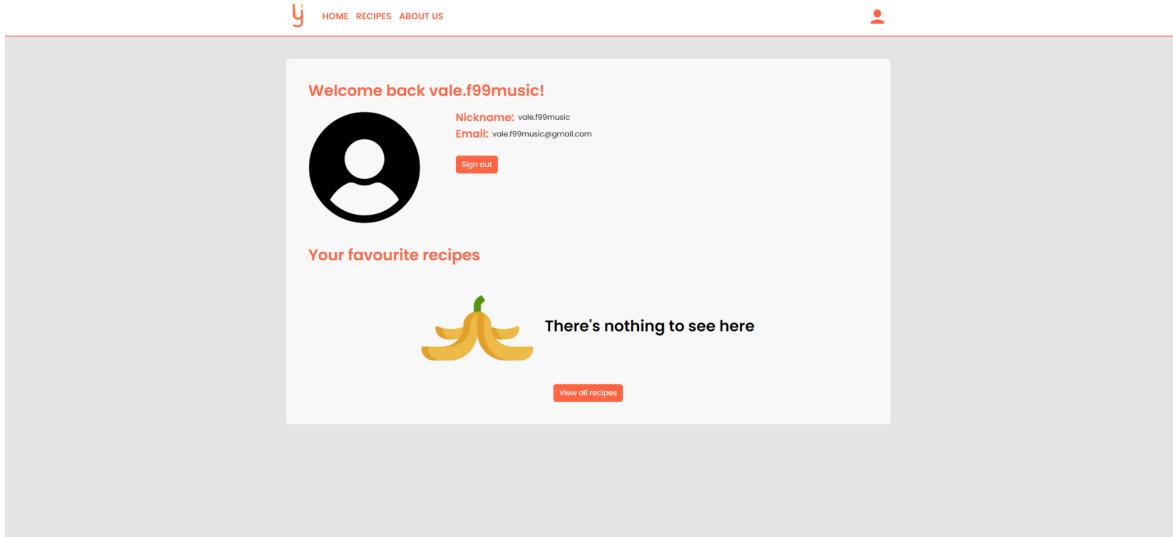


Figura 6.7.3: View Profile, visualizzazione del profilo senza ricette salvate

Se non sono presenti delle ricette salvate viene mostrata un'illustrazione e la scritta "There's nothing to see here" e un bottone che riporta alla pagina delle ricette.