

# Documentazione

## Yummy!



# Indice

<b>1 Introduzione</b>	<b>2</b>
<b>2 Pacchetti utilizzati</b>	<b>3</b>
<b>3 Istruzioni per eseguire la build</b>	<b>3</b>
<b>4 Architettura</b>	<b>4</b>
<b>5 Routing</b>	<b>5</b>
<b>6 Views</b>	<b>6</b>
6.1 App	6
6.2 Home	7
6.3 Recipes	8
6.3.1 RecipesCardsGrid	8
6.3.2 RecipesTable	9
6.4 About us	10
6.5 RecipeDetail	11
6.5.1 RecipeCategory	12
6.5.2 RecipeType	12
6.6 Default	13

# 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`

## 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 5 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. **Wrapper** a cui corrisponde il path: /recipes/:number
5. **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

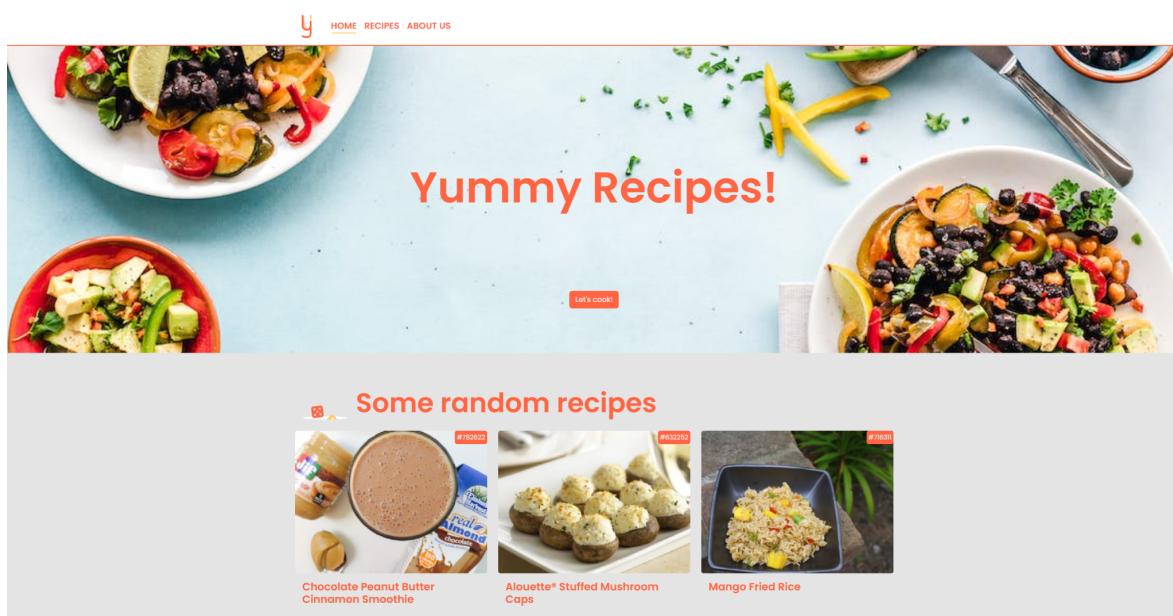


Figura 6.2.1: View Home

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

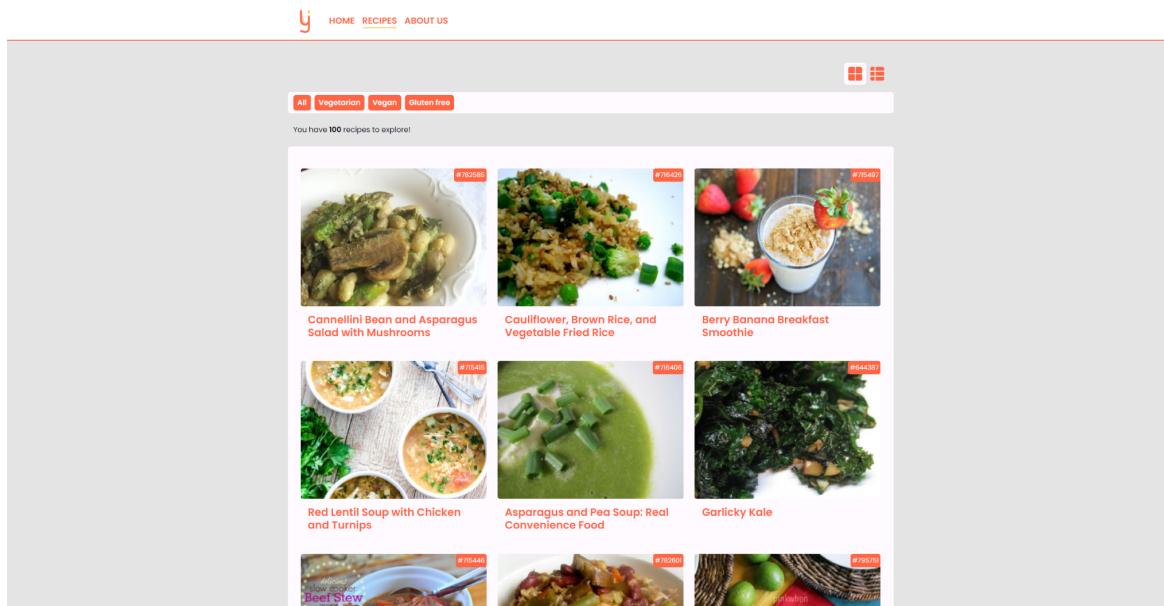


Figura 6.3.1.1: View Recipes, caso Grid

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

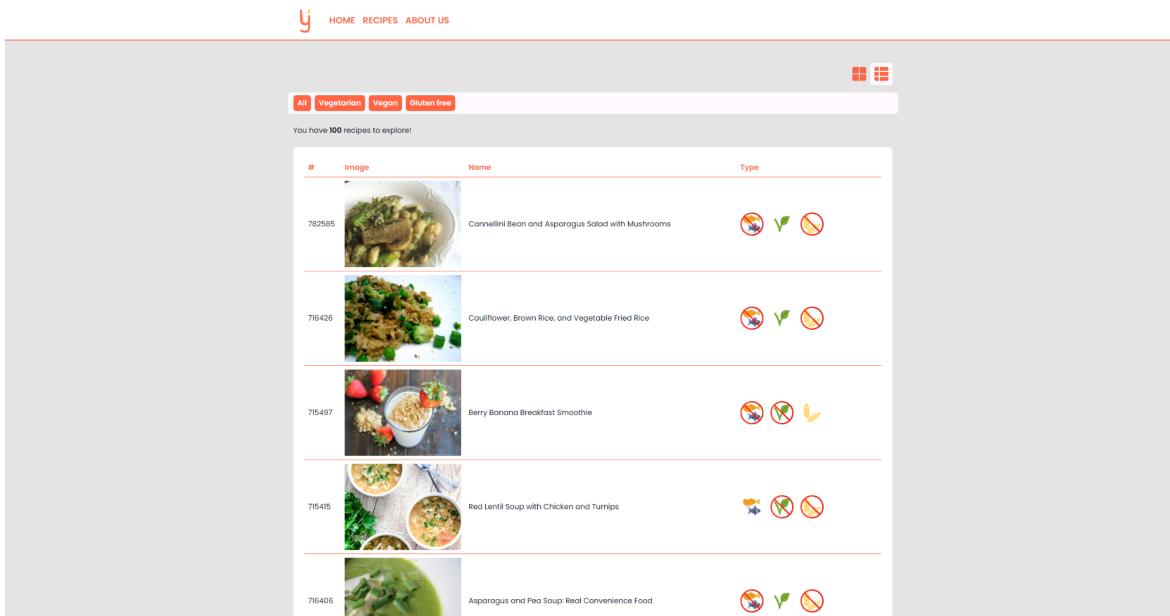


Figura 6.3.2.1: View Recipes, caso Table

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.

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.

## 6.4 About us

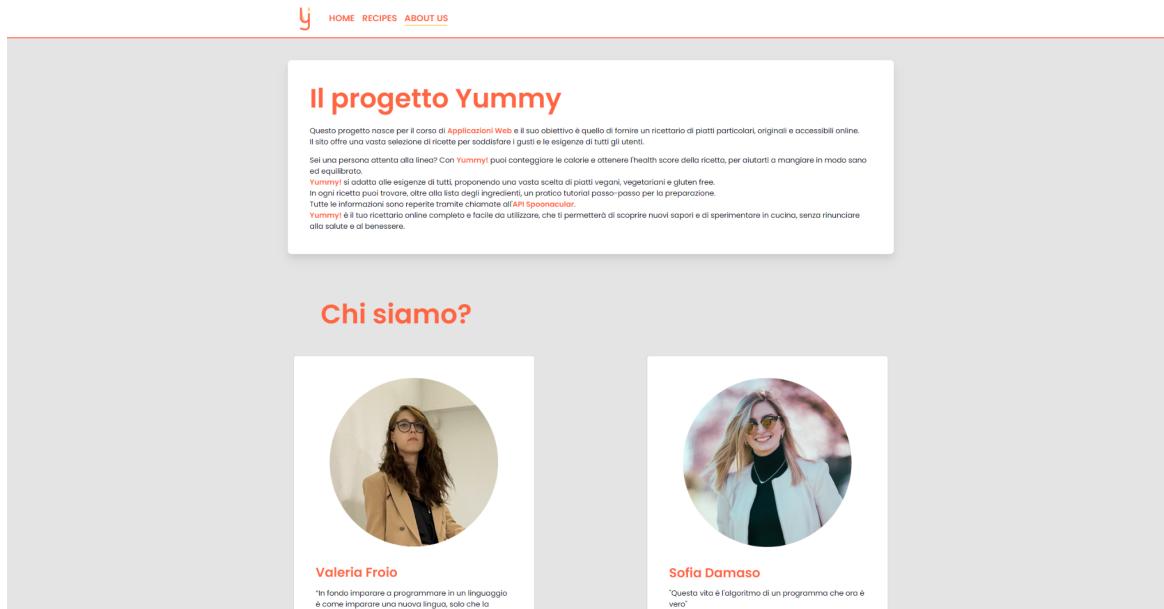


Figura 6.4.1: View About Us

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

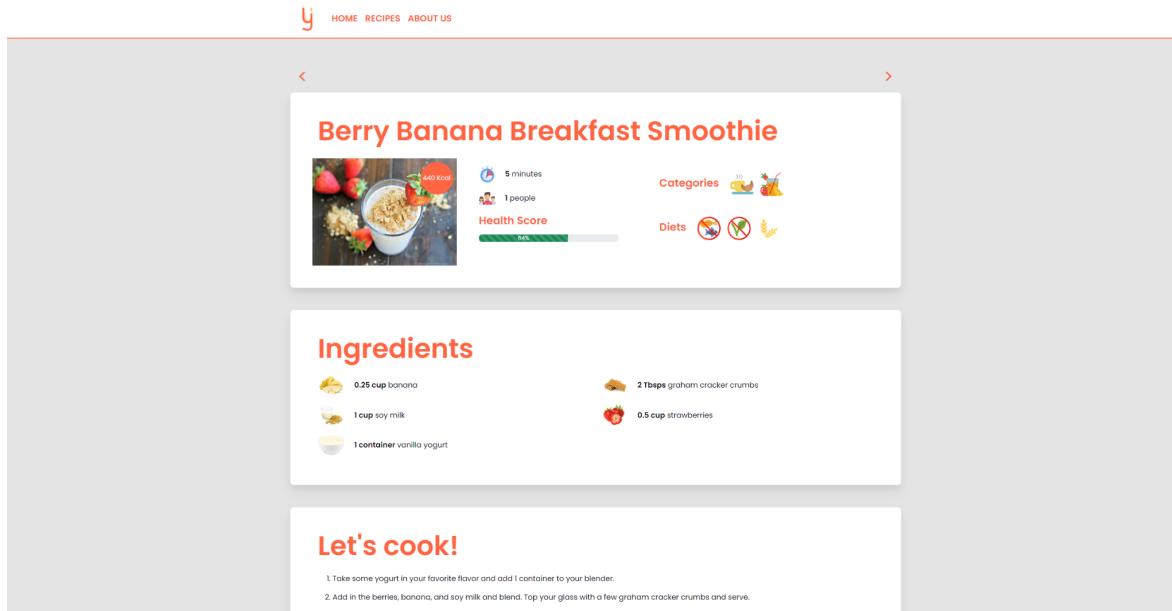


Figura 6.5.1: View RecipeDetail per la ricetta numero 715497

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

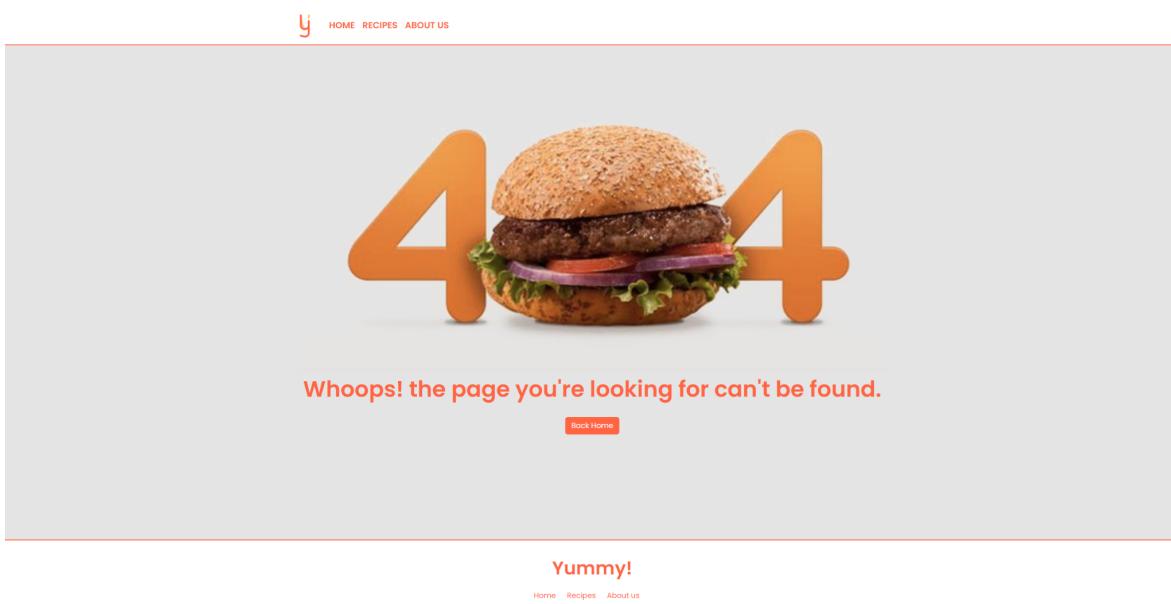


Figura 6.6.1: Pagina di errore costruita dal componente Default

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.