

React Anwendung

User

bitte merke folgende Aufbau: wir haben einen public ordner mit dem Inhalt einer Index.html [<!DOCTYPE html>

```
<html lang="de">
```

```
<head>
```

```
<meta charset="utf-8" />
```

```
<link rel="icon" href="%PUBLIC_URL%/favicon.ico" /> <!-- Quelle: https://icon-icons.com/de/symbol/Essen-gesund-Obst-Gem%C3%BCse/233071 -->
```

```
<meta name="viewport" content="width=device-width, initial-scale=1" />
```

```
<meta name="description" content="Webapplikation zum Finden von Rezepten" />
```

```
<meta name="keywords" content="Rezepte, Lebensmittel">
```

```
<meta name="author" content="André Schlichting, Jan Gruner, Daniel Kunst, Saskia-Maria Mellies, Christoph Rudolph, Theo Prill" />
```

```
<meta name="theme-color" content="#000000" />
```

```
<link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
```

```
<!--
```

```
  manifest.json provides metadata used when your web app is installed on a
```

```
  user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
```

```
-->
```

```
<link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
```

```
<!--
```

```
  Notice the use of %PUBLIC_URL% in the tags above.
```

```
  It will be replaced with the URL of the `public` folder during the build.
```

```
  Only files inside the `public` folder can be referenced from the HTML.
```

```
  Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
```

```
  work correctly both with client-side routing and a non-root public URL.
```

Learn how to configure a non-root public URL by running ``npm run build``.

-->

`<title>Zutatenzauber</title>`

`</head>`

`<body>`

`<noscript>Bitte aktivieren Sie JavaScript, um die Webanwendung zu nutzen.</noscript>`

`<div id="root"></div>`

`<!--`

This HTML file is a template.

If you open it directly in the browser, you will see an empty page.

You can add webfonts, meta tags, or analytics to this file.

The build step will place the bundled scripts into the `<body>` tag.

To begin the development, run ``npm start`` or ``yarn start``.

To create a production bundle, use ``npm run build`` or ``yarn build``.

-->

`</body>`

`</html>`

**], des Weiteren haben wir ein ordner src mit ordner components mit Inhalt der Datei Header.jsx [import
Navbar from "./Navbar";**

function NavBar(props) {

return (

<ul className="nav nav-tabs">

<li className="nav-item">

Gerichte

```
    </a>
```

```
  </li>
```

```
<li className="nav-item">
```

```
  <a className="nav-link" href="/">
```

```
    Cocktails
```

```
  </a>
```

```
</li>
```

```
</ul>
```

```
);
```

```
}
```

```
function HeadLine(props) {
```

```
  return (
```

```
    <div className="headline-container">
```

```
      <h1 className="display-2">Zutatenzauber</h1>
```

```
    </div>
```

```
  );
```

```
}
```

```
function Header(props) {
```

```
  return (
```

```
    <div className="header-container">
```

```
      <HeadLine />
```

```
      <div className="nav-container">
```

```
        <NavBar />
```

```
</div>
```

```
</div>
```

```
);
```

```
}
```

```
export default Header;
```

```
] + InputDish.jsx
```

```
[import { useState } from "react";
```

```
/*function FilterDish() {
```

```
  return (
```

```
    <div className="filter-container">
```

```
      <fieldset>
```

```
        <legend></legend>
```

```
        <div className="form-check">
```

```
          <input className="form-check-input" type="checkbox" value="" id="Vegan" />
```

```
          <label className="form-check-label" htmlFor="Vegan" >
```

```
            vegan
```

```
          </label>
```

```
        </div>
```

```
        <div className="form-check">
```

```
          <input className="form-check-input" type="checkbox" value="" id="LowCarb" />
```

```
          <label className="form-check-label" htmlFor="LowCarb" >
```

```
            low carb
```

```
          </label>
```

```
        </div>
```

```
      <div className="form-check">
```

```

    <input className="form-check-input" type="checkbox" value="" id="glutenfrei" />

    <label className="form-check-label" htmlFor="LowCarb" >

        glutenfrei

    </label>

</div>

<div className="form-check">

    <input className="form-check-input" type="checkbox" value="" id="proteinreich" />

    <label className="form-check-label" htmlFor="LowCarb" >

        proteinreich

    </label>

</div>

</fieldset>

</div>

);

}

*/

const InputDish = ({onTyping}) => {

    /*

    const [inputText, SetInputText] = useState("");

    const changeInputText = (e) => {

        SetInputText(e.target.value);

    }

```

```

const submit = (e) => {

  e.preventDefault();

  onClick(inputText)

}

*/}

return (

  <div className="input-container">

    <input type="text" onChange={(e) => onTyping(e.target.value)} className="form-control" placeholder="Bitte
Zutat eingeben"></input>

    { /*<form >

      <input type="text" onChange={changeInputText} value={inputText} className="form-control"
placeholder="Bitte Zutaten eingeben"></input>

      <input type="submit" onClick={submit} className="form-button-control" value="Suchen"></input>

    </form> */}

  </div>

);

}

```

```

export default InputDish;] + OutputDish.jsx

```

```

[import { useState, useEffect } from "react";

```

```

function CardDish (props) {

```

```

  return (

```

```

    <div className="test"> </div>

```

```

  )

```

```
}
```

```
const OutputDish = ({ingredient, onClickRecipe}) => {
```

```
  const [dishes, setDishes] = useState([]);
```

```
  useEffect(() => {
```

```
    if (!ingredient) {
```

```
      return;
```

```
    }
```

```
    else {
```

```
      fetch(`https://www.themealdb.com/api/json/v1/1/filter.php?i=${ingredient}`)
```

```
        .then(response => response.json())
```

```
        .then(data => {
```

```
          setDishes(data.meals);
```

```
        })
```

```
    }
```

```
  }, [ingredient]);
```

```
  return (
```

```
    <div>
```

```
      <h1>Child-Element (Output): {ingredient}</h1>
```

```
      <div className="card-container">
```

```
        {dishes?.map((dish) => {
```

```
<div key={dish.idMeal} className="card">
```

```
<img alt="dish" width="200px" src={dish.strMealThumb}/>
```

```
<div className="card-body">
```

```
<h5 className="card-title">{dish.strMeal}</h5>
```

```
<p className="card-text">Hier steht eine Kurzbeschreibung</p>
```

```
<button onClick={{e} => onClickRecipe(dish.idMeal)} className="btn btn-primary">Zum Rezept</button>
```

```
</div>
```

```
</div>
```

```
)))
```

```
</div>
```

```
</div>
```

```
);
```

```
}
```

```
export default OutputDish;] + SearchDish.jsx[import { useState } from "react";
```

```
import InputDish from "./InputDish";
```

```
import OutputDish from "./OutputDish";
```

```
import ShowDish from "./OutputDish";
```

```
const SearchDish = () => {
```

```
const [ingredient, SetIngredient] = useState("");
```

```
const [dishSelected, setDishSelected] = useState([]);
```

```
const handleInputText = (value) => {
```



```

const valueTrim = value.trim();

SetIngredient(valueTrim);

}

const handleClickRecipe = (dishID) => {

  fetch(`https://www.themealdb.com/api/json/v1/1/filter.php?i=${dishID}`)

    .then(response => response.json())

    .then(data => {

      setDishSelected(data.meals);

    })

}

return (

  <div className="SearchDish">

    <h1>parent-Element: {ingredient}</h1>

    { /*<InputDish onClick={handleInputText} /> Handling erst nach Klick auf Button*/ }

    <InputDish onTyping={handleInputText} />

    <OutputDish ingredient={ingredient} onClickRecipe={handleClickRecipe} />

    <ShowDish dish={dishSelected} />

  </div>

);

}

export default SearchDish; ] + ShowDish.jsx [

const ShowDish = ({dish}) => {

```

```

return (

  <div key={dish.idMeal} className="card">

    <img alt="dish" width="200px" src={dish.strMealThumb}/>

    <div className="card-body">

      <h5 className="card-title">{dish.strMeal}</h5>

      <p className="card-text">Hier steht eine Kurzbeschreibung</p>

    </div>

  </div>

);
}

```

```

export default ShowDish;] + im ordner src die datei APP.jsx [import React, { Component } from "react";

```

```

import Header from "../components/Header";

```

```

import SearchDish from "../components/SearchDish";

```

```

import "bootstrap/dist/css/bootstrap.min.css";

```

```

import "../style.css";

```

```

function App() {

```

```

  return (

    <div className="App">

```

```

      <Header />

```

```

      <SearchDish />

```

```

    </div>

```

```

  );

```

```

}

```

```
export default App;] + index.js [import React, { StrictMode } from 'react';
```

```
import ReactDOM from 'react-dom/client';
```

```
import App from './App';
```

```
const root = ReactDOM.createRoot(document.getElementById('root'));
```

```
root.render(  
  <StrictMode>  
    <App />  
  </StrictMode>  
);  
];  
+style.css
```

```
{  
  body {
```

```
    background-color: rgb(255, 255, 255);
```

```
  }
```

```
.header-container {
```

```
  display: flex;
```

```
  justify-content: space-between;
```

```
  align-items: center;
```

```
  background-color: #14354b;
```

```
  padding: 10px;
```

```
  color: #fff;
```

```
}
```

```
.nav.nav-tabs {
```

```
margin-bottom: 0;
```

```
margin-right: 40px;
```

```
}
```

```
.nav.nav-tabs .nav-item {
```

```
margin-right: 0px;
```

```
}
```

```
/* HeadLine Component */
```

```
.headline-container {
```

```
color: #fdfdfd;
```

```
margin-left: 70px;
```

```
}
```

```
/* Suchleiste (Zutaten) */
```

```
.form-control {
```

```
width: 70%;
```

```
resize: none;
```

```
margin-right: auto;
```

```
margin-left: 90px;
```

```
margin-top: 15px;
```

```
}
```

```
/* Button Suchleiste (Zutaten) */
```

```
.form-button-control {
```

```
cursor: pointer;
```

```
}
```

```
/* Button Suchleiste (Zutaten) */
```

```
.form-control {
```

```
width: 70%;
```

```
resize: none;
```

```
margin-right: auto;
```

```
margin-left: 90px;
```

```
margin-top: 15px;
```

```
}
```

```
/* FilterKategorien */
```

```
.filter-container {
```

```
margin-top: 5px;
```

```
margin-right: 240px;
```

```
display: flex;
```

```
}
```

```
/* InputDish Komponente */
```

```
.input-container {
```

```
margin-top: 20px;
```

```
}
```

```
/* CardDish und OutputDish Komponenten */
```

```
.card-container {
```

```
display: flex;
```

```
flex-wrap: wrap;
```

```
justify-content: space-around;
```

```
margin-top: 150px;
```

```
}
```

```
.card {
```

```
width: 300px;
```

```
margin-bottom: 20px;
```

```
}
```

```
.card-img-top {
```

```
width: 100%;
```

```
height: auto;
```

```
}
```

```
.card-body {
```

```
background-color: #ffffff;
```

```
padding: 15px;
```

```
}
```

```
.btn-primary {
```

```
background-color: #14354b;
```

```
color: #ffffff;
```

```
border: 1px solid #ffffff
```

```
}
```

```
@media (max-width: 850px) {
```

```
.card-container {
```

```
flex-direction: column;
```

```
}
```

```
}
```

], kannst du bitte die einzelnen funktionen auf deutsch genau erklären

ChatGPT

Natürlich, ich werde die einzelnen Funktionen und Komponenten im Code auf Deutsch erklären:

```
### Index.html
```

Dies ist die Haupt-HTML-Datei. Hier werden grundlegende Metadaten für die Webanwendung festgelegt, wie Charset, Favicon, Viewport-Einstellungen und Autorinformationen. Es enthält auch Verweise auf externe Ressourcen wie das Manifest für die Progressive Web App.

```
### Header.jsx
```

```
#### `NavBar`:
```

Diese Funktion definiert die Navigationsleiste mit zwei Links: "Gerichte" und "Cocktails".

```
#### `HeadLine`:
```

Definiert den Titel der Anwendung als Überschrift.

```
#### `Header`:
```

Diese Funktion kombiniert die oben genannten Komponenten und stellt den Header der Anwendung dar.

InputDish.jsx

Diese Komponente stellt eine Eingabemöglichkeit für Zutaten dar. Aktuell ist der Code für die State-Verwaltung und das Einreichen des Formulars auskommentiert.

OutputDish.jsx

`CardDish`:

Eine einfache Platzhalterkomponente für ein Gericht, die derzeit keinen Inhalt hat.

`OutputDish`:

Die Hauptkomponente für die Anzeige von Gerichten basierend auf einer bestimmten Zutat. Sie verwendet den `useEffect`-Hook, um Daten von einer externen API abzurufen und dann die erhaltenen Gerichte anzuzeigen.

SearchDish.jsx

Die Hauptkomponente für die Suche nach Gerichten. Hier werden die Eingabe- (`InputDish`) und Ausgabe- (`OutputDish`) Komponenten kombiniert. Der Zustand für die eingegebene Zutat wird hier verwaltet.

ShowDish.jsx

Diese Komponente zeigt ein ausgewähltes Gericht anhand seiner Informationen wie Bild, Name und Kurzbeschreibung.

App.jsx

Die Hauptkomponente der Anwendung. Hier werden der Header und die Suchkomponente (`SearchDish`) kombiniert.

index.js

Der Einstiegspunkt der React-Anwendung, der die Hauptkomponente (`App`) in die Wurzel (`root`) des HTML-Dokuments rendert.

style.css

Die CSS-Datei für das Styling der Anwendung. Hier werden verschiedene Stile für den Header, die Suchleiste, die Kartenansicht der Gerichte und andere Elemente festgelegt.

Diese Dateien und Komponenten zusammen bilden eine React-Anwendung zur Suche und Anzeige von Gerichten basierend auf Zutaten. Wenn du spezifische Fragen zu Teilen des Codes hast, lass es mich wissen!

User

welche daten erhält man bei der json - date zurück

ChatGPT

In der React-Anwendung wird eine externe API verwendet, um Daten zu Gerichten basierend auf einer Zutat abzurufen. Die genaue Struktur der empfangenen JSON-Daten hängt von der API ab, die in diesem Fall die TheMealDB-API ist. Hier ist eine allgemeine Vorstellung davon, welche Informationen in den JSON-Daten enthalten sein könnten, basierend auf der üblichen Struktur von Mahlzeiten-Datenbanken:

```
```json
```

```
{
```

```
 "meals": [
```

```
 {
```

```
 "idMeal": "123",
```

```
 "strMeal": "Spaghetti Bolognese",
```

```
 "strMealThumb": "url-zum-bild.jpg",
```

```
 "strCategory": "Pasta",
```

```
 "strArea": "Italian",
```

```
 "strInstructions": "Schritte zur Zubereitung der Mahlzeit...",
```

```
 // Andere relevante Informationen
```

```
 },
```

```
 {
```

```
 "idMeal": "456",
```

```
 "strMeal": "Chicken Curry",
```

```
 "strMealThumb": "url-zum-bild.jpg",
```

```
 "strCategory": "Curry",
```

```
"strArea": "Indian",
```

```
"strInstructions": "Schritte zur Zubereitung der Mahlzeit...",
```

```
// Andere relevante Informationen
```

```
},
```

```
// Weitere Gerichte
```

```
]
```

```
}
```

```
...
```

Die wichtigsten Informationen, die wahrscheinlich in den JSON-Daten enthalten sind:

- ``idMeal``: Eindeutige Identifikation für jedes Gericht.

- ``strMeal``: Der Name des Gerichts.

- ``strMealThumb``: Die URL zum Bild des Gerichts.

- ``strCategory``: Die Kategorie des Gerichts (z.B. Pasta, Curry).

- ``strArea``: Die Region oder Herkunft des Gerichts (z.B. Italian, Indian).

- ``strInstructions``: Eine ausführliche Anleitung zur Zubereitung des Gerichts.

Diese Informationen können dann in der React-Anwendung verwendet werden, um die Gerichte anzuzeigen und weitere Details über jedes Gericht darzustellen. Der genaue Aufbau kann jedoch je nach API und deren Datenstruktur variieren.

## User

kannst du für mich ordner component eine neue jsx. nach main ingredient mit der API [https://www.themealdb.com/api/json/v1/1/filter.php?i=chicken\\_breast](https://www.themealdb.com/api/json/v1/1/filter.php?i=chicken_breast), chicken\_breast ist die veränderbare variabel in der API. Von der übergeben Json-Datei idMeal gespeichert

## ChatGPT

Natürlich, ich verstehe, dass du eine neue Komponente in dem Ordner "components" erstellen möchtest, die die API `https://www.themealdb.com/api/json/v1/1/filter.php?i=chicken_breast` verwendet und die `idMeal` aus den empfangenen Daten speichert. Hier ist eine Beispielkomponente, die dies umsetzen könnte:

```
``jsx

// components/MainIngredientDishes.jsx

import React, { useState, useEffect } from "react";

const MainIngredientDishes = () => {

 const [dishes, setDishes] = useState([]);

 useEffect(() => {

 const fetchDishes = async () => {

 try {

 const response = await fetch(

 "https://www.themealdb.com/api/json/v1/1/filter.php?i=chicken_breast"

);

 if (!response.ok) {

 throw new Error("Fehler beim Abrufen der Daten");

 }

 const data = await response.json();

 setDishes(data.meals);

 } catch (error) {

 console.error("Fehler:", error);

 }

 }

 });

}
```

```
};
```

```
 fetchDishes();
```

```
 }, []); // Leere Abhängigkeitsliste sorgt dafür, dass der Effekt nur einmal ausgeführt wird.
```

```
 return (
```

```
 <div>
```

```
 <h2>Gerichte mit Hühnerbrust</h2>
```

```

```

```
 {dishes.map((dish) => (
```

```
 <li key={dish.idMeal}>
```

```
 {dish.strMeal} (ID: {dish.idMeal})
```

```

```

```
)))
```

```

```

```
 </div>
```

```
);
```

```
};
```

```
export default MainIngredientDishes;
```

```
'''
```

In dieser Komponente wird die `useEffect`-Hook verwendet, um die Daten von der API zu holen, sobald die Komponente montiert wird. Die erhaltenen Daten werden im Zustand `dishes` gespeichert und dann in einer Liste angezeigt. Die `idMeal` wird in dieser Liste ebenfalls angezeigt.

Du kannst diese Komponente dann in deiner Anwendung verwenden, indem du sie importierst und in deine Hauptkomponente (`SearchDish`, `App`, etc.) integrierst. Zum Beispiel:

```
``jsx
```

```
// Beispiel in SearchDish.jsx
```

```
import React from "react";
```

```
import InputDish from "../InputDish";
```

```
import OutputDish from "../OutputDish";
```

```
import ShowDish from "../ShowDish";
```

```
import MainIngredientDishes from "../MainIngredientDishes";
```

```
const SearchDish = () => {
```

```
 // ... bisheriger Code
```

```
 return (
```

```
 <div className="SearchDish">
```

```
 {/* ... bisheriger Code */}
```

```
 <MainIngredientDishes />
```

```
 </div>
```

```
);
```

```
}
```

```
export default SearchDish;
```

```
``
```

Hier wird die neue Komponente `MainIngredientDishes` in die `SearchDish`-Komponente integriert. Du kannst dies nach Bedarf in anderen Teilen deiner Anwendung anpassen.

## User

wie kann ich ein weiteres eingabe feld für eine Abfrage InputDish hinzufügen

## ChatGPT

Um ein weiteres Eingabefeld für eine Abfrage in der `InputDish`-Komponente hinzuzufügen, kannst du den State erweitern und ein weiteres Eingabefeld in der Render-Methode hinzufügen. Hier ist eine modifizierte Version der `InputDish`-Komponente mit zwei Eingabefeldern:

```
``jsx

// components/InputDish.jsx

import React, { useState } from "react";

const InputDish = ({ onTyping }) => {

 const [ingredient1, setIngredient1] = useState("");
 const [ingredient2, setIngredient2] = useState("");

 const handleInputChange = (e, setIngredient) => {

 setIngredient(e.target.value);

 };

 const handleSubmit = () => {

 // Hier kannst du die eingegebenen Zutaten weiterverarbeiten

 // Zum Beispiel: onTyping(`${ingredient1} ${ingredient2}`);

 };

 return (

 <div className="input-container">

 <input
```

```

 type="text"

 onChange={(e) => handleInputChange(e, setIngredient1)}

 value={ingredient1}

 className="form-control"

 placeholder="Bitte Zutat 1 eingeben"

 />

 <input

 type="text"

 onChange={(e) => handleInputChange(e, setIngredient2)}

 value={ingredient2}

 className="form-control"

 placeholder="Bitte Zutat 2 eingeben"

 />

 <button onClick={handleSubmit} className="form-button-control">

 Suchen

 </button>

</div>

);

};

export default InputDish;

``

```

In dieser Version der `InputDish`-Komponente habe ich zwei Zustandsvariablen `ingredient1` und `ingredient2` hinzugefügt, um die beiden eingegebenen Zutaten zu verfolgen. Die `handleInputChange`-Funktion wurde angepasst, um das entsprechende `setIngredient` für das jeweilige Eingabefeld zu verwenden. Der `handleSubmit`-Button ruft die `onTyping`-Funktion auf und kann die eingegebenen Zutaten kombinieren oder auf andere Weise verwenden.

Stelle sicher, dass du diese Komponente in deiner Anwendung entsprechend integrierst und den Zustand oder die eingegebenen Daten entsprechend verarbeitest.

User

die abfrage soll unabhängig von einander geschehen

ChatGPT

Verstanden, wenn die Abfragen unabhängig voneinander erfolgen sollen, kannst du einfach zwei separate Eingabefelder und zwei separate Zustände verwenden. Hier ist die angepasste `InputDish`-Komponente:

```
```jsx
```

```
// components/InputDish.jsx
```

```
import React, { useState } from "react";
```

```
const InputDish = ({ onTyping }) => {
```

```
  const [ingredient1, setIngredient1] = useState("");
```

```
  const [ingredient2, setIngredient2] = useState("");
```

```
  const handleInputChange = (e, setIngredient) => {
```

```
    setIngredient(e.target.value);
```

```
  };
```

```
  const handleSubmit1 = () => {
```

```
    onTyping(ingredient1);
```

```
    // Hier kannst du weitere Aktionen für die erste Zutat durchführen
```

```
  };
```

```
  const handleSubmit2 = () => {
```

```
    onTyping(ingredient2);
```

```
    // Hier kannst du weitere Aktionen für die zweite Zutat durchführen
```

```
  };
```



```
return (  
  
  <div className="input-container">  
  
    <input  
  
      type="text"  
  
      onChange={(e) => handleInputChange(e, setIngredient1)}  
  
      value={ingredient1}  
  
      className="form-control"  
  
      placeholder="Bitte Zutat 1 eingeben"  
  
    />  
  
    <button onClick={handleSubmit1} className="form-button-control">  
  
      Suchen Zutat 1  
  
    </button>  
  
    <input  
  
      type="text"  
  
      onChange={(e) => handleInputChange(e, setIngredient2)}  
  
      value={ingredient2}  
  
      className="form-control"  
  
      placeholder="Bitte Zutat 2 eingeben"  
  
    />  
  
    <button onClick={handleSubmit2} className="form-button-control">  
  
      Suchen Zutat 2  
  
    </button>  
  
  </div>  
  
);  
};
```

```
export default InputDish;
```

```
...
```

In dieser Version gibt es separate Zustände (`ingredient1` und `ingredient2`) und separate Handler (`handleSubmit1` und `handleSubmit2`). Jedes Eingabefeld und der zugehörige Button sind unabhängig voneinander, sodass du zwei separate Abfragen für verschiedene Zutaten durchführen kannst. Beachte, dass dies nur eine grundlegende Implementierung ist, und je nach deinen Anforderungen können weitere Anpassungen erforderlich sein.

User

wenn ich einen weiteren inputDish hinzufügen

ChatGPT

Wenn du einen weiteren `InputDish` hinzufügen möchtest, kannst du die Komponente so gestalten, dass sie für jede Zutat einen eigenen Zustand und Handler hat. Hier ist eine modifizierte Version der `InputDish`-Komponente, die mehrere Eingabefelder unterstützt:

```
```jsx
```

```
// components/InputDish.jsx
```

```
import React, { useState } from "react";
```

```
const InputDish = ({ onTyping }) => {
```

```
 const [ingredients, setIngredients] = useState(["", ""]);
```

```
 const handleInputChange = (e, index) => {
```

```
 const newIngredients = [...ingredients];
```

```
 newIngredients[index] = e.target.value;
```

```
 setIngredients(newIngredients);
```

```
 };
```

```
 const handleSubmit = (index) => {
```

```
 onTyping(ingredients[index]);
```

```

 // Hier kannst du weitere Aktionen für die entsprechende Zutat durchführen

};

return (
 <div className="input-container">
 {ingredients.map((ingredient, index) => (
 <div key={index}>
 <input
 type="text"
 onChange={(e) => handleInputChange(e, index)}
 value={ingredient}
 className="form-control"
 placeholder={`Bitte Zutat ${index + 1} eingeben`}
 />
 <button onClick={() => handleSubmit(index)} className="form-button-control">
 Suchen Zutat {index + 1}
 </button>
 </div>
))}
 </div>
);
};

export default InputDish;

```

In dieser Version verwendet die `InputDish`-Komponente ein Array (`ingredients`), um die Zustände für mehrere Zutaten zu speichern. Die Funktion `handleInputChange` wurde angepasst, um den Index des Eingabefelds zu berücksichtigen. Die Komponente rendert nun dynamisch für jede Zutat ein Eingabefeld und einen Button.

Du kannst dann beliebig viele `InputDish`-Komponenten in deiner Anwendung verwenden, und jedes wird seine eigenen Zutaten unabhängig voneinander verwalten. Zum Beispiel:

```
``jsx

// Beispiel in SearchDish.jsx

import React from "react";

import InputDish from "../InputDish";

import OutputDish from "../OutputDish";

import ShowDish from "../ShowDish";

const SearchDish = () => {

 // ... bisheriger Code

 return (

 <div className="SearchDish">

 <InputDish onTyping={handleTypingForIngredient1} />

 <InputDish onTyping={handleTypingForIngredient2} />

 {/* ... weitere InputDish-Komponenten nach Bedarf */}

 <OutputDish ingredient={ingredient} onClickRecipe={handleClickRecipe} />

 <ShowDish dish={dishSelected} />

 </div>

);

};
```

```
export default SearchDish;
```

```
'''
```

Hier wird `InputDish` mehrmals mit unterschiedlichen `onTyping`-Funktionen aufgerufen, um die Eingaben für verschiedene Zutaten zu verwalten.

## User

```
import { useState, useEffect } from "react";
```

```
function CardDish (props) {
```

```
 return (
```

```
 <div className="test"> </div>
```

```
)
```

```
}
```

```
const OutputDish = ({ingredient, onClickRecipe}) => {
```

```
 const [dishes, setDishes] = useState([]);
```

```
 useEffect(() => {
```

```
 if (!ingredient) {
```

```
 return;
```

```
 }
```

```
 else {
```

```
 fetch(`https://www.themealdb.com/api/json/v1/1/filter.php?i=${ingredient}`)
```

```
 .then(response => response.json())
```

```

 .then(data => {

 setDishes(data.meals);

 })

 }

 }, [ingredient]));

 return (

 <div>

 <h1>Child-Element (Output): {ingredient}</h1>

 <div className="card-container">

 {dishes?.map((dish) => (

 <div key={dish.idMeal} className="card">

 <div className="card-body">

 <h5 className="card-title">{dish.strMeal}</h5>

 <p className="card-text">Hier steht eine Kurzbeschreibung</p>

 <button onClick={(e) => onClickRecipe(dish.idMeal)} className="btn btn-primary">Zum Rezept</button>

 </div>

 </div>

))}

 </div>

 </div>

)
}

```

```
);
```

```
}
```

```
export default OutputDish;
```

kannst du mir den code genau erklären

ChatGPT

Natürlich, gerne erkläre ich den Code für die `OutputDish`-Komponente:

### 1. **\*\*Import von React Hooks:\*\***

```
```jsx
```

```
import { useState, useEffect } from "react";
```

```
```
```

Hier werden die React Hooks `useState` und `useEffect` importiert. Diese werden verwendet, um Zustandsvariablen in der Komponente zu verwalten und Seiteneffekte (wie das Laden von Daten von einer API) zu steuern.

### 2. **\*\*CardDish-Komponente (Nicht verwendete Platzhalterkomponente):\*\***

```
```jsx
```

```
function CardDish (props) {
```

```
  return (
```

```
    <div className="test"> </div>
```

```
  )
```

```
}
```

```
```
```

Diese Funktion definiert eine einfache Platzhalterkomponente namens `CardDish`, die in der aktuellen Implementierung nicht verwendet wird. Du kannst sie entfernen, wenn sie nicht benötigt wird.

### 3. **\*\*OutputDish-Komponente:\*\***

```
``jsx
```

```
const OutputDish = ({ ingredient, onClickRecipe }) => {
```

```
 // Zustandsvariable dishes und die zugehörige Funktion setDishes werden deklariert
```

```
 const [dishes, setDishes] = useState([]);
```

```
 // Der useEffect-Hook wird verwendet, um auf Änderungen der Zutat zu reagieren und Daten von der API abzurufen
```

```
 useEffect(() => {
```

```
 // Wenn keine Zutat vorhanden ist, wird die Funktion vorzeitig verlassen
```

```
 if (!ingredient) {
```

```
 return;
```

```
 } else {
```

```
 // Ansonsten wird die API aufgerufen, um Gerichte basierend auf der Zutat zu erhalten
```

```
 fetch(`https://www.themealdb.com/api/json/v1/1/filter.php?i=${ingredient}`)
```

```
 .then(response => response.json())
```

```
 .then(data => {
```

```
 // Die erhaltenen Gerichte werden im Zustand gespeichert
```

```
 setDishes(data.meals);
```

```
 })
```

```
 }
```

```
 }, [ingredient]);
```

```
 // Die JSX-Struktur wird zurückgegeben
```

```
 return (
```

```
 <div>
```

```
 <h1>Child-Element (Output): {ingredient}</h1>
```

```
 <div className="card-container">
```



```

 { /* Mapping über die Gerichte und Darstellung in Kartenformat */ }

 {dishes?.map((dish) => (

 <div key={dish.idMeal} className="card">

 <div className="card-body">

 <h5 className="card-title">{dish.strMeal}</h5>

 <p className="card-text">Hier steht eine Kurzbeschreibung</p>

 { /* onClickRecipe-Funktion wird aufgerufen, wenn der Button geklickt wird */ }

 <button onClick={(e) => onClickRecipe(dish.idMeal)} className="btn btn-primary">Zum
Rezept</button>

 </div>

 </div>

)
)}

</div>

</div>

);

}

```

```

// Die OutputDish-Komponente wird exportiert

```

```

export default OutputDish;

```

```

'''

```

Hier wird der Zustand `dishes` verwendet, um die von der API erhaltenen Gerichte zu speichern. Der `useEffect`-Hook ruft die API auf, wenn sich die Zutat ändert, und aktualisiert dann den Zustand `dishes` mit den erhaltenen Daten. Die JSX-Struktur der Komponente zeigt dann die Gerichte in einem Kartenformat an.

Ich hoffe, das hilft dir, den Code zu verstehen. Wenn du weitere Fragen hast, stehe ich zur Verfügung!

## User

```
import { useState } from "react";
```

```
import InputDish from "../InputDish";
```

```
import OutputDish from "../OutputDish";
```

```
import ShowDish from "../OutputDish";
```

```
import InputDish2 from "../InputDish2";
```

```
const SearchDish = () => {
```

```
 const [ingredient, SetIngredient] = useState("");
```

```
 const [dishSelected, setDishSelected] = useState([]);
```

```
 const handleInputText = (value) => {
```

```
 const valueTrim = value.trim();
```

```
 SetIngredient(valueTrim);
```

```
 }
```

```
 const handleClickRecipe = (dishID) => {
```

```
 fetch(`https://www.themealdb.com/api/json/v1/1/filter.php?i=${dishID}`)
```

```
 .then(response => response.json())
```

```
 .then(data => {
```

```
 setDishSelected(data.meals);
```

```
 })
```

```
 }
```

```

return (

 <div className="SearchDish">

 <h1>parent-Element: {ingredient}</h1>

 { /*<InputDish onClick={handleInputText} /> Handling erst nach Klick auf Button*/ }

 <InputDish onTyping={handleInputText} />

 <OutputDish ingredient={ingredient} onClickRecipe={handleClickRecipe} />

 <ShowDish dish={dishSelected} />

 <h1>parent-Element2 {strMeal}</h1>

 <InputDish2 onTyping={handleInputText} />

 <OutputDish strMeal={strMeal} onClickRecipe={handleClickRecipe} />

 </div>

);
}

```

```
export default SearchDish;
```

**erkläre mir bitte genau den code**

**ChatGPT**

**Natürlich, ich werde den Code für die `SearchDish`-Komponente im Detail erklären:**

```

``jsx

import { useState } from "react";

import InputDish from "../InputDish";

import OutputDish from "../OutputDish";

import ShowDish from "../OutputDish";

import InputDish2 from "../InputDish2";

```

```

const SearchDish = () => {

 // Zustandsvariablen für die Zutat, das ausgewählte Gericht und den Gerichtsnamen (strMeal)

 const [ingredient, SetIngredient] = useState("");

 const [dishSelected, setDishSelected] = useState([]);

 const [strMeal, setStrMeal] = useState(""); // Neuer Zustand für strMeal

 // Handler-Funktion für die Eingabe der Zutat

 const handleInputText = (value) => {

 const valueTrim = value.trim();

 SetIngredient(valueTrim);

 }

 // Handler-Funktion für das Klicken auf ein Rezept

 const handleClickRecipe = (dishID) => {

 fetch(`https://www.themealdb.com/api/json/v1/1/filter.php?i=${dishID}`)

 .then(response => response.json())

 .then(data => {

 setDishSelected(data.meals);

 })

 }

 // Handler-Funktion für die Eingabe des Gerichtsnamens (strMeal) im zweiten InputDish

 const handleInputText2 = (value) => {

 setStrMeal(value);

 }

```

```

return (

 <div className="SearchDish">

 /* Erstes InputDish */

 <h1>parent-Element: {ingredient}</h1>

 <InputDish onTyping={handleInputText} />

 <OutputDish ingredient={ingredient} onClickRecipe={handleClickRecipe} />

 <ShowDish dish={dishSelected} />

 /* Zweites InputDish */

 <h1>parent-Element2: {strMeal}</h1>

 <InputDish2 onTyping={handleInputText2} />

 <OutputDish strMeal={strMeal} onClickRecipe={handleClickRecipe} />

 </div>

);

}

export default SearchDish;

'''

```

Hier ist eine Aufschlüsselung des Codes:

#### 1. **\*\*Zustandsvariablen:\*\***

- ``ingredient``: Speichert die Zutat für das erste InputDish.
- ``dishSelected``: Speichert die ausgewählten Gerichte basierend auf der Zutat.
- ``strMeal``: Neuer Zustand, der den Gerichtsnamen speichert, der im zweiten InputDish eingegeben wird.

## 2. **\*\*Handler-Funktionen:\*\***

- `handleInputText`: Aktualisiert den Zustand `ingredient` mit der eingegebenen Zutat.

- `handleClickRecipe`: Ruft die API auf, um Gerichte basierend auf der Zutat oder dem Gerichtsnamen zu erhalten und aktualisiert den Zustand `dishSelected`.

- `handleInputText2`: Aktualisiert den Zustand `strMeal` mit dem eingegebenen Gerichtsnamen im zweiten `InputDish`.

## 3. **\*\*Rendering:\*\***

- Rendert die erste Gruppe von `InputDish`, `OutputDish` und `ShowDish` für die Zutat.

- Rendert die zweite Gruppe von `InputDish2` und `OutputDish` für den Gerichtsnamen (`strMeal`).

Insgesamt ermöglicht dieser Code zwei separate Eingabefelder und zugehörige Ausgabe für Zutaten und Gerichtsnamen. Jedes Eingabefeld hat seine eigene Logik und beeinflusst nur den entsprechenden Teil der Anwendung.

## User

```
import { useState } from "react";
```

```
import InputDish from "../InputDish";
```

```
import OutputDish from "../OutputDish";
```

```
import ShowDish from "../OutputDish";
```

```
import InputDish2 from "../InputDish2";
```

```
import OutputDish2 from "../OutputDish2";
```

```
const SearchDish = () => {
```

```
 const [ingredient, SetIngredient] = useState("");
```

```
 const [dishSelected, setDishSelected] = useState([]);
```

```
 const [strMeal, SetStrMeal] = useState("");
```

```
const [dishSelected2, setDishSelected2] = useState([]);
```

```
const handleInputText = (value) => {
```

```
 const valueTrim = value.trim();
```

```
 SetIngredient(valueTrim);
```

```
}
```

```
const handleClickRecipe = (dishID) => {
```

```
 fetch(`https://www.themealdb.com/api/json/v1/1/filter.php?i=${dishID}`)
```

```
 .then(response => response.json())
```

```
 .then(data => {
```

```
 setDishSelected(data.meals);
```

```
 })
```

```
}
```

```
return (
```

```
 <div className="SearchDish">
```

```
 <h1>parent-Element: {ingredient}</h1>
```

```
 { /*<InputDish onClick={handleInputText} /> Handling erst nach Klick auf Button*/ }
```

```
 <InputDish onTyping={handleInputText} />
```

```
 <OutputDish ingredient={ingredient} onClickRecipe={handleClickRecipe} />
```

```
 <ShowDish dish={dishSelected} />
```

```
 <h1>parent-Element2 {strMeal}</h1>
```

```
 <InputDish2 onTyping={handleInputText} />
```

```
<OutputDish2 strMeal={strMeal} onClickRecipe={handleClickRecipe} />
```

```
</div>
```

```
);
```

```
}
```

`export default SearchDish;` wie muss ich den code richtig vervollständigen

#### ChatGPT

Es sieht so aus, als ob du eine zweite Gruppe von Input- und Output-Dish-Komponenten hinzufügen möchtest, die mit einer anderen Zutat (`strMeal`) arbeitet. Hier ist eine mögliche Vervollständigung des Codes:

```
```jsx
```

```
import { useState } from "react";
```

```
import InputDish from "./InputDish";
```

```
import OutputDish from "./OutputDish";
```

```
import ShowDish from "./ShowDish"; // Beachte: Hier wird die Komponente ShowDish anstatt OutputDish verwendet
```

```
import InputDish2 from "./InputDish2";
```

```
import OutputDish2 from "./OutputDish2";
```

```
const SearchDish = () => {
```

```
  const [ingredient, SetIngredient] = useState("");
```

```
  const [dishSelected, setDishSelected] = useState([]);
```

```
  const [strMeal, SetStrMeal] = useState("");
```

```
  const [dishSelected2, setDishSelected2] = useState([]);
```



```
const handleInputText = (value) => {  
  
  const valueTrim = value.trim();  
  
  SetIngredient(valueTrim);  
  
}
```

```
const handleInputText2 = (value) => {  
  
  const valueTrim = value.trim();  
  
  SetStrMeal(valueTrim);  
  
}
```

```
const handleClickRecipe = (dishID) => {  
  
  fetch(`https://www.themealdb.com/api/json/v1/1/filter.php?i=${dishID}`)  
  
    .then(response => response.json())  
  
    .then(data => {  
  
      setDishSelected(data.meals);  
  
    })  
  
}
```

```
const handleClickRecipe2 = (dishID) => {  
  
  fetch(`https://www.themealdb.com/api/json/v1/1/filter.php?i=${dishID}`)  
  
    .then(response => response.json())  
  
    .then(data => {  
  
      setDishSelected2(data.meals);  
  
    })  
  
}
```

```

return (

  <div className="SearchDish">

    {/* Erste Gruppe */}

    <h1>parent-Element: {ingredient}</h1>

    <InputDish onTyping={handleInputText} />

    <OutputDish ingredient={ingredient} onClickRecipe={handleClickRecipe} />

    <ShowDish dish={dishSelected} />


    {/* Zweite Gruppe */}

    <h1>parent-Element2: {strMeal}</h1>

    <InputDish2 onTyping={handleInputText2} />

    <OutputDish2 strMeal={strMeal} onClickRecipe={handleClickRecipe2} />

  </div>

);

}

export default SearchDish;

'''

```

In diesem Beispiel habe ich `InputDish2`, `OutputDish2` und `handleClickRecipe2` für die zweite Gruppe hinzugefügt. Beachte, dass die zweite Gruppe mit `strMeal` arbeitet und separate Zustände (`strMeal` und `dishSelected2`) verwendet. Falls du bereits eine `ShowDish`-Komponente für die zweite Gruppe erstellt hast, **kannst du sie entsprechend verwenden.**

User

```
import { useState, useEffect } from "react";
```

```
function CardDish (props) {
```

```
  return (
```

```
<div className="test"></div>
```

```
)
```

```
}
```

```
const OutputDish2 = ({strMeal, onClickRecipe}) => {
```

```
  const [dishes, setDishes] = useState([]);
```

```
  useEffect(() => {
```

```
    if (!strMeal) {
```

```
      return;
```

```
    }
```

```
    else {
```

```
      fetch(`www.themealdb.com/api/json/v1/1/filter.php?i=${strMeal}`)
```

```
        .then(response => response.json())
```

```
        .then(data => {
```

```
          setDishes(data.meals);
```

```
        })
```

```
      }
```

```
    }, [strMeal]);
```

```

return (
  <div>
    <h1>Child-Element (Output): {strMeal}</h1>
    <div className="card-container">
      {dishes?.map((dish2) => (
        <div key={dish2.idMeal} className="card">
          <img alt="dish" width="200px" src={dish2.strMealThumb}/>
          <div className="card-body">
            <h5 className="card-title">{dish2.strMeal}</h5>
            <p className="card-text">Hier steht eine Kurzbeschreibung</p>
            <button onClick={(e) => onClickRecipe(dish2.idMeal)} className="btn btn-primary">Zum
Rezept</button>
          </div>
        </div>
      )
    </div>
  )
)}
</div>
</div>
);
}

```

export default OutputDish2; --> ist die Abfrage der API so richtig wenn man nach der Hauptzutat schaut bei der JSON format "meals": [

```

{
  "strMeal": "Chick-Fil-A Sandwich",
  "strMealThumb": "https://www.themealdb.com/images/media/meals/sbx7n71587673021.jpg",
  "idMeal": "53016"
},

```

ChatGPT

Es scheint, als ob die Abfrage der API in `OutputDish2` korrekt ist, aber es gibt einen kleinen Fehler im URL-Pfad. Der Pfad zur API sollte mit `https://` beginnen. Hier ist die korrigierte Zeile:

```
```jsx
```

```
fetch(`https://www.themealdb.com/api/json/v1/1/filter.php?i=${strMeal}`)
```

```
```
```

Der vollständige Code für `OutputDish2` mit der Korrektur sieht so aus:

```
```jsx
```

```
import { useState, useEffect } from "react";
```

```
const OutputDish2 = ({ strMeal, onClickRecipe }) => {
```

```
 const [dishes, setDishes] = useState([]);
```

```
 useEffect(() => {
```

```
 if (!strMeal) {
```

```
 return;
```

```
 } else {
```

```
 fetch(`https://www.themealdb.com/api/json/v1/1/filter.php?i=${strMeal}`)
```

```
 .then(response => response.json())
```

```
 .then(data => {
```

```
 setDishes(data.meals);
```

```
 })
```

```
 .catch(error => {
```

```
 console.error("Error fetching data:", error);
```

```

 });

 }

 }, [strMeal]);

 return (
 <div>

 <h1>Child-Element (Output): {strMeal}</h1>

 <div className="card-container">

 {dishes?.map((dish2) => (

 <div key={dish2.idMeal} className="card">

 <div className="card-body">

 <h5 className="card-title">{dish2.strMeal}</h5>

 <p className="card-text">Hier steht eine Kurzbeschreibung</p>

 <button onClick={(e) => onClickRecipe(dish2.idMeal)} className="btn btn-primary">Zum
Rezept</button>

 </div>

 </div>

)</div>

)</div>

)</div>

);
 }

 export default OutputDish2;

 ...

```

Dieser Code verwendet jetzt den korrekten API-URL-Pfad, um die Daten für die Hauptzutat abzurufen.

User

so sieht aktuell die InputDish2.jsx

```
[import { useState } from "react";
```

```
const InputDish2 = ({onTyping}) => {
```

```
 return (
```

```
 <div className="input-container">
```

```
 <input type="text" onChange={(e) => onTyping(e.target.value)} className="form-control" placeholder="Bitte Hauptzutat eingeben"></input>
```

```
 </div>
```

```
);
```

```
}
```

```
export default InputDish2;] + OutputDish2.jsx [import { useState, useEffect } from "react";
```

```
function CardDish (props) {
```

```
 return (
```

```
 <div className="test"> </div>
```

```
)
```

```
}
```

```
const OutputDish2 = ({strMeal, onClickRecipe}) => {
```

```
 const [dishes, setDishes] = useState([]);
```

```
useEffect(() => {

 if (!strMeal) {

 return;

 }

 else {

 fetch(`https://www.themealdb.com/api/json/v1/1/filter.php?i=${strMeal}`)

 .then(response => response.json())

 .then(data => {

 setDishes(data.meals);

 })

 }

 }, [strMeal]);

return (

 <div>

 <h1>Child-Element (Output): {strMeal}</h1>

 <div className="card-container">

 {dishes?.map((dish2) => (

 <div key={dish2.idMeal} className="card">

 <div className="card-body">

 <h5 className="card-title">{dish2.strMeal}</h5>
```



```
<p className="card-text">Hier steht eine Kurzbeschreibung</p>
```

```
<button onClick={(e) => onClickRecipe(dish2.idMeal)} className="btn btn-primary">Zum
Rezept</button>
```

```
</div>
```

```
</div>
```

```
)}
```

```
</div>
```

```
</div>
```

```
);
```

```
}
```

```
export default OutputDish2;] + SearchDish.jsx [import { useState } from "react";
```

```
import InputDish from "../InputDish";
```

```
import OutputDish from "../OutputDish";
```

```
import ShowDish from "../OutputDish";
```

```
import InputDish2 from "../InputDish2";
```

```
import OutputDish2 from "../OutputDish2";
```

```
import ShowDish2 from "../OutputDish2";
```

```
const SearchDish = () => {
```

```
 const [ingredient, SetIngredient] = useState("");
```

```
 const [dishSelected, setDishSelected] = useState([]);
```

```
 const [strMeal, SetStrMeal] = useState("");
```

```
 const [dishSelected2, setDishSelected2] = useState([]);
```

```
const handleInputText = (value) => {
```

```
 const valueTrim = value.trim();
```

```
 SetIngredient(valueTrim);
```

```
}
```

```
const handleInputText2 = (value) => {
```

```
 const valueTrim = value.trim();
```

```
 SetStrMeal(valueTrim);
```

```
}
```

```
const handleClickRecipe = (dishID) => {
```

```
 fetch(`https://www.themealdb.com/api/json/v1/1/filter.php?i=${dishID}`)
```

```
 .then(response => response.json())
```

```
 .then(data => {
```

```
 setDishSelected(data.meals);
```

```
 })
```

```
}
```

```
const handleClickRecipe2 = (dishID) => {
```

```
 fetch(`https://www.themealdb.com/api/json/v1/1/filter.php?i=${dishID}`)
```

```
 .then(response => response.json())
```

```
 .then(data => {
```

```
 setDishSelected2(data.meals);
```

```
 })
```

```
}
```

```
return (
```

```
 <div className="SearchDish">
```

```
 <h1>parent-Element: {ingredient}</h1>
```

```
 { /*<InputDish onClick={handleInputText} /> Handling erst nach Klick auf Button */ }
```

```
 <InputDish onTyping={handleInputText} />
```

```
 <OutputDish ingredient={ingredient} onClickRecipe={handleClickRecipe} />
```

```
 <ShowDish dish={dishSelected} />
```

```
 <h1>parent-Element2 {strMeal}</h1>
```

```
 <InputDish2 onTyping={handleInputText2} />
```

```
 <OutputDish2 strMeal={strMeal} onClickRecipe={handleClickRecipe2} />
```

```
 <ShowDish2 dish2={dishSelected2} />
```

```
 </div>
```

```
)
```

```
}
```

```
export default SearchDish;] + ShowDish2.jsx [
```

```
const ShowDish2 = ({dish2}) => {
```

```
 return (
```

```
 <div key={dish2.idMeal} className="card">
```

```

```

```
 <div className="card-body">
```

```
 <h5 className="card-title">{dish2.strMeal}</h5>
```

```

 <p className="card-text">Hier steht eine Kurzbeschreibung</p>

 </div>

</div>

);

}

export default ShowDish2;

```

] aktueller Zustand des Codes, wie kann ich am Besten bei den übermittelten code, diesen so anpassen, dass wenn nach der Eingabe der Hauptzaten und die einzelnen gerichte angezeigt werden. Die Button "Zum Rezept drücken", dass er die jeweilige "idMeal": "52850" (Beispielnummer von der API `https://www.themealdb.com/api/json/v1/1/filter.php?i=${strMeal}`) aus der JSON [ "meals": [`

```

{
 "strMeal": "Chick-Fil-A Sandwich",
 "strMealThumb": "https://www.themealdb.com/images/media/meals/sbx7n71587673021.jpg",
 "idMeal": "53016"
}} herauszieht und eine neue Abfrage startet mit der API Lookup full meal details by id

```

`www.themealdb.com/api/json/v1/1/lookup.php?i=.idMeal.`

## ChatGPT

Um die Funktionalität zu implementieren, bei der nach dem Klick auf den "Zum Rezept"-Button die `idMeal` des ausgewählten Gerichts verwendet wird, um weitere Details abzurufen, kannst du folgende Schritte unternehmen:

1. **Ändere die `onClickRecipe`-Funktion in `SearchDish.jsx`**, um sowohl die `idMeal` als auch andere Details abzurufen:

```

``jsx

const handleClickRecipe = (dishID) => {

 fetch(`https://www.themealdb.com/api/json/v1/1/lookup.php?i=${dishID}`)

 .then(response => response.json())

```

```

 .then(data => {

 // Hier erhältst du weitere Details des ausgewählten Gerichts

 console.log("Details des ausgewählten Gerichts:", data.meals[0]);

 // Setze den Zustand oder führe andere Aktionen mit den erhaltenen Details durch

 });
}

'''

```

2. **\*\*Passe den `onClickRecipe`-Handler für die zweite Gruppe (`handleClickRecipe2`) entsprechend an\*\*.**

```

'''jsx

const handleClickRecipe2 = (dishID) => {

 fetch(`https://www.themealdb.com/api/json/v1/1/lookup.php?i=${dishID}`)

 .then(response => response.json())

 .then(data => {

 // Hier erhältst du weitere Details des ausgewählten Gerichts

 console.log("Details des ausgewählten Gerichts (Gruppe 2):", data.meals[0]);

 // Setze den Zustand oder führe andere Aktionen mit den erhaltenen Details durch

 });

}

'''

```

3. **\*\*Ändere die `OutputDish` und `OutputDish2` Komponenten, um den korrekten `idMeal`-Wert für den Button zu verwenden\*\*.**

In `OutputDish.jsx`:

```
```jsx
```

```
<button onClick={(e) => onClickRecipe(dish.idMeal)} className="btn btn-primary">Zum Rezept</button>
```

```
```
```

In `OutputDish2.jsx`:

```
```jsx
```

```
<button onClick={(e) => onClickRecipe(dish2.idMeal)} className="btn btn-primary">Zum Rezept</button>
```

```
```
```

Durch diese Änderungen sollte nach dem Klick auf den "Zum Rezept"-Button die `idMeal` des ausgewählten Gerichts an den entsprechenden Handler übergeben werden, und du kannst dann eine neue Abfrage mit dieser `idMeal` für weitere Details starten. Beachte, dass du die erhaltenen Details in den entsprechenden Zuständen setzen oder andere Aktionen durchführen kannst, je nachdem, was du mit den Daten vorhast.

## User

kannst du hierzu mir eine weitere componente schreiben, die sich nach drücken des Button "Zum Rezept" [{

```
"meals": [
```

```
{
```

```
"idMeal": "52772",
```

```
"strMeal": "Teriyaki Chicken Casserole",
```

```
"strDrinkAlternate": null,
```

```
"strCategory": "Chicken",
```

```
"strArea": "Japanese",
```

```
"strInstructions": "Preheat oven to 350° F. Spray a 9x13-inch baking pan with non-stick spray.\r\nCombine soy sauce, ½ cup water, brown sugar, ginger and garlic in a small saucepan and cover. Bring to a boil over medium heat. Remove lid and cook for one minute once boiling.\r\nMeanwhile, stir together the corn starch and 2 tablespoons of water in a separate dish until smooth. Once sauce is boiling, add mixture to the saucepan and stir to combine. Cook until the sauce starts to thicken then remove from heat.\r\nPlace the chicken breasts in the prepared pan. Pour one cup of the sauce over top of chicken. Place chicken in oven and bake 35 minutes or until cooked through. Remove from oven and shred chicken in the dish using two forks.\r\n*Meanwhile, steam or cook the vegetables according to package directions.\r\nAdd the cooked vegetables and rice to the casserole dish with the chicken. Add most of the remaining sauce, reserving a bit to drizzle over the top when serving. Gently toss everything together in the casserole dish until combined. Return
```

to oven and cook 15 minutes. Remove from oven and let stand 5 minutes before serving. Drizzle each serving with remaining sauce. Enjoy!",

"strMealThumb": "https://www.themealdb.com/images/media/meals/wvpsxx1468256321.jpg",

"strTags": "Meat,Casserole",

"strYoutube": "https://www.youtube.com/watch?v=4aZr5hZXP\_s",

"strIngredient1": "soy sauce",

"strIngredient2": "water",

"strIngredient3": "brown sugar",

"strIngredient4": "ground ginger",

"strIngredient5": "minced garlic",

"strIngredient6": "cornstarch",

"strIngredient7": "chicken breasts",

"strIngredient8": "stir-fry vegetables",

"strIngredient9": "brown rice",

"strIngredient10": "",

"strIngredient11": "",

"strIngredient12": "",

"strIngredient13": "",

"strIngredient14": "",

"strIngredient15": "",

"strIngredient16": null,

"strIngredient17": null,

"strIngredient18": null,

"strIngredient19": null,

"strIngredient20": null,

"strMeasure1": "3/4 cup",

"strMeasure2": "1/2 cup",

```
 "strMeasure3": "1/4 cup",

 "strMeasure4": "1/2 teaspoon",

 "strMeasure5": "1/2 teaspoon",

 "strMeasure6": "4 Tablespoons",

 "strMeasure7": "2",

 "strMeasure8": "1 (12 oz.)",

 "strMeasure9": "3 cups",

 "strMeasure10": "",

 "strMeasure11": "",

 "strMeasure12": "",

 "strMeasure13": "",

 "strMeasure14": "",

 "strMeasure15": "",

 "strMeasure16": null,

 "strMeasure17": null,

 "strMeasure18": null,

 "strMeasure19": null,

 "strMeasure20": null,

 "strSource": null,

 "strImageSource": null,

 "strCreativeCommonsConfirmed": null,

 "dateModified": null
 }
]
}} öffnet
```



Um eine neue Komponente zu erstellen, die die Details eines Gerichts nach dem Klick auf den "Zum Rezept"-Button anzeigt, kannst du die folgende Komponente verwenden. Diese Komponente zeigt die Details des ausgewählten Gerichts anhand der übergebenen Daten an:

```
``jsx

import React from "react";

const RecipeDetails = ({ recipeDetails, onClose }) => {

 // Stelle sicher, dass recipeDetails vorhanden ist, bevor du darauf zugreifst

 if (!recipeDetails || !recipeDetails.length) {

 return null;

 }

 const {

 idMeal,

 strMeal,

 strCategory,

 strArea,

 strInstructions,

 strMealThumb,

 strTags,

 strYoutube,

 strIngredient1,

 strIngredient2,

 strIngredient3,

 // ... füge alle benötigten Zutaten hinzu

 } = recipeDetails[0];
```

```
return (

 <div className="recipe-details-container">

 <div className="recipe-details-header">

 <h2>{strMeal}</h2>

 <button onClick={onClose}>Schließen</button>

 </div>

 <div className="recipe-details-body">

 <p>Kategorie: {strCategory}</p>

 <p>Herkunft: {strArea}</p>

 <p>Zutaten:</p>

 {strIngredient1}

 {strIngredient2}

 {strIngredient3}

 {/* Füge alle benötigten Zutaten hinzu */}

 <p>Zubereitung:</p>

 <p>{strInstructions}</p>

 <p>Tags: {strTags}</p>

 {strYoutube && Zum Video}

 </div>

 </div>

);
};
```

```
export default RecipeDetails;
```

```
``
```

Diese Komponente erwartet `recipeDetails` als Prop, das die Details des ausgewählten Gerichts enthält. Der `onClose`-Handler wird verwendet, um die Komponente zu schließen. Stelle sicher, dass du die Styling-Anpassungen entsprechend deinen Bedürfnissen vornimmst.

In deiner `SearchDish.jsx`-Datei kannst du diese Komponente wie folgt verwenden:

```
``jsx
```

```
import React, { useState } from "react";
```

```
import InputDish from "../InputDish";
```

```
import OutputDish from "../OutputDish";
```

```
import RecipeDetails from "../RecipeDetails";
```

```
const SearchDish = () => {
```

```
 const [ingredient, setIngredient] = useState("");
```

```
 const [dishSelected, setDishSelected] = useState([]);
```

```
 const [showRecipeDetails, setShowRecipeDetails] = useState(false);
```

```
 const handleInputText = (value) => {
```

```
 const valueTrim = value.trim();
```

```
 setIngredient(valueTrim);
```

```
 }
```

```
 const handleClickRecipe = (dishID) => {
```

```
 fetch(`https://www.themealdb.com/api/json/v1/1/lookup.php?i=${dishID}`)
```

```

 .then(response => response.json())

 .then(data => {

 setDishSelected(data.meals);

 setShowRecipeDetails(true);

 });
 }

 const handleCloseRecipeDetails = () => {

 setShowRecipeDetails(false);

 }

 return (

 <div className="SearchDish">

 <h1>parent-Element: {ingredient}</h1>

 <InputDish onTyping={handleInputText} />

 <OutputDish ingredient={ingredient} onClickRecipe={handleClickRecipe} />

 {showRecipeDetails && (

 <RecipeDetails recipeDetails={dishSelected} onClose={handleCloseRecipeDetails} />

)}

 </div>

);
}

export default SearchDish;

```

Diese Änderungen fügen die `RecipeDetails`-Komponente hinzu und zeigen die Details des ausgewählten Gerichts an. Der "Schließen"-Button wird verwendet, um die Details-Komponente zu schließen. Beachte, dass

du das Styling für `recipe-details-container` und andere Klassen entsprechend deinen Anforderungen anpassen musst.

User

**handleClickRecipe2 hinzufügen**

ChatGPT

Natürlich, du kannst `handleClickRecipe2` auf ähnliche Weise hinzufügen wie `handleClickRecipe`. Hier ist eine aktualisierte Version deines `SearchDish.jsx`-Codes:

```
```jsx
```

```
import React, { useState } from "react";
```

```
import InputDish from "../InputDish";
```

```
import OutputDish from "../OutputDish";
```

```
import RecipeDetails from "../RecipeDetails";
```

```
const SearchDish = () => {
```

```
  const [ingredient, setIngredient] = useState("");
```

```
  const [dishSelected, setDishSelected] = useState([]);
```

```
  const [showRecipeDetails, setShowRecipeDetails] = useState(false);
```

```
  const handleInputText = (value) => {
```

```
    const valueTrim = value.trim();
```

```
    setIngredient(valueTrim);
```

```
  }
```

```
  const handleClickRecipe = (dishID) => {
```

```
    fetch(`https://www.themealdb.com/api/json/v1/1/lookup.php?i=${dishID}`)
```

```
      .then(response => response.json())
```

```

        .then(data => {
            setDishSelected(data.meals);
            setShowRecipeDetails(true);
        });
    }

```

```

const handleCloseRecipeDetails = () => {
    setShowRecipeDetails(false);
}

```

```

const handleClickRecipe2 = (dishID) => {
    // Führe eine ähnliche API-Anfrage durch wie in handleClickRecipe
    // und behandle die Daten entsprechend
}

```

```

return (
    <div className="SearchDish">
        <h1>parent-Element: {ingredient}</h1>
        <InputDish onTyping={handleInputText} />
        <OutputDish ingredient={ingredient} onClickRecipe={handleClickRecipe} />
        {showRecipeDetails && (
            <RecipeDetails recipeDetails={dishSelected} onClose={handleCloseRecipeDetails} />
        )}
        {/* Hier kannst du eine weitere OutputDish und RecipeDetails hinzufügen, wenn nötig */}
    </div>
);

```

```
}
```

```
export default SearchDish;
```

```
'''
```

Du kannst die Logik in `handleClickRecipe2` so gestalten, dass sie eine ähnliche API-Anfrage durchführt wie `handleClickRecipe`, jedoch mit dem entsprechenden `dishID` für das zweite Gericht. Beachte, dass du, wenn du separate Zustände (`dishSelected2`, etc.) für jedes Gericht haben möchtest, dies ebenfalls berücksichtigen solltest.

User

anhand der JSON [{

"meals": [

{

"idMeal": "52772",

"strMeal": "Teriyaki Chicken Casserole",

"strDrinkAlternate": null,

"strCategory": "Chicken",

"strArea": "Japanese",

"strInstructions": "Preheat oven to 350° F. Spray a 9x13-inch baking pan with non-stick spray.\r\nCombine soy sauce, ½ cup water, brown sugar, ginger and garlic in a small saucepan and cover. Bring to a boil over medium heat. Remove lid and cook for one minute once boiling.\r\nMeanwhile, stir together the corn starch and 2 tablespoons of water in a separate dish until smooth. Once sauce is boiling, add mixture to the saucepan and stir to combine. Cook until the sauce starts to thicken then remove from heat.\r\nPlace the chicken breasts in the prepared pan. Pour one cup of the sauce over top of chicken. Place chicken in oven and bake 35 minutes or until cooked through. Remove from oven and shred chicken in the dish using two forks.\r\n*Meanwhile, steam or cook the vegetables according to package directions.\r\nAdd the cooked vegetables and rice to the casserole dish with the chicken. Add most of the remaining sauce, reserving a bit to drizzle over the top when serving. Gently toss everything together in the casserole dish until combined. Return to oven and cook 15 minutes. Remove from oven and let stand 5 minutes before serving. Drizzle each serving with remaining sauce. Enjoy!",

"strMealThumb": "https://www.themealdb.com/images/media/meals/wvpsxx1468256321.jpg",

"strTags": "Meat,Casserole",

"strYoutube": "https://www.youtube.com/watch?v=4aZr5hZXP_s",

"strIngredient1": "soy sauce",

"strIngredient2": "water",

"strIngredient3": "brown sugar",

"strIngredient4": "ground ginger",

"strIngredient5": "minced garlic",

"strIngredient6": "cornstarch",

"strIngredient7": "chicken breasts",

"strIngredient8": "stir-fry vegetables",

"strIngredient9": "brown rice",

"strIngredient10": "",

"strIngredient11": "",

"strIngredient12": "",

"strIngredient13": "",

"strIngredient14": "",

"strIngredient15": "",

"strIngredient16": null,

"strIngredient17": null,

"strIngredient18": null,

"strIngredient19": null,

"strIngredient20": null,

"strMeasure1": "3/4 cup",

"strMeasure2": "1/2 cup",

"strMeasure3": "1/4 cup",

"strMeasure4": "1/2 teaspoon",

"strMeasure5": "1/2 teaspoon",

"strMeasure6": "4 Tablespoons",

"strMeasure7": "2",


```

    "strMeasure8": "1 (12 oz.)",
    "strMeasure9": "3 cups",
    "strMeasure10": "",
    "strMeasure11": "",
    "strMeasure12": "",
    "strMeasure13": "",
    "strMeasure14": "",
    "strMeasure15": "",
    "strMeasure16": null,
    "strMeasure17": null,
    "strMeasure18": null,
    "strMeasure19": null,
    "strMeasure20": null,
    "strSource": null,
    "strImageSource": null,
    "strCreativeCommonsConfirmed": null,
    "dateModified": null
  }
]

```

`}}` kannst du bitte variablen darstellen im code `RecipeDetails.jsx`

ChatGPT

Natürlich, hier ist eine Beispielimplementierung für deine `RecipeDetails.jsx`-Komponente unter Verwendung der bereitgestellten JSON-Daten:

```

```jsx
import React from "react";

```

```
const RecipeDetails = ({ recipeDetails, onClose }) => {

 if (!recipeDetails || recipeDetails.length === 0) {

 return null;

 }
}
```

```
const meal = recipeDetails[0];
```

```
return (
```

```
<div className="recipe-details">
```

```
<h2>{meal.strMeal}</h2>
```

```

```

```
<h3>Ingredients:</h3>
```

```

```

```
 {Array.from({ length: 20 }, (_, index) => index + 1)}
```

```
 .filter((index) => meal[`strIngredient${index}`])
```

```
 .map((index) => (
```

```
 <li key={index}>
```

```
 {meal[`strMeasure${index}`]} {meal[`strIngredient${index}`]}
```

```

```

```
)))
```

```

```

```
<h3>Instructions:</h3>
```

```
<p>{meal.strInstructions}</p>
```

```
<div className="video-link">
```

```
{meal.strYoutube && (
```

```

```

```
 Watch Video
```

```

```

```
))
```

```
</div>
```

```
<button onClick={onClose}>Close</button>
```

```
</div>
```

```
);
```

```
};
```

```
export default RecipeDetails;
```

```
...
```

Diese Komponente verwendet die Daten aus dem `recipeDetails`-Prop, um die Details des Rezepts anzuzeigen. Die Zutaten und Maße werden in einer Liste dargestellt, und die Anweisungen werden als Absatz angezeigt. Wenn ein YouTube-Link vorhanden ist, wird auch ein Link zum Anschauen des Videos angezeigt. Der "Schließen"-Button ruft die `onClose`-Funktion auf, um die Komponente zu schließen. Beachte, dass dies nur ein einfaches Beispiel ist, und du die Darstellung je nach Bedarf anpassen kannst.

## User

wie muss jetzt diese datei import `{ useState }` from "react";

```
import InputDish from "../InputDish";
```

```
import OutputDish from "../OutputDish";
```

```
import ShowDish from "../OutputDish";
```

```
import InputDish2 from "../InputDish2";
```

```
import OutputDish2 from "../OutputDish2";
```

```
import ShowDish2 from "../OutputDish2";
```

```
const SearchDish = () => {
```

```
 const [ingredient, SetIngredient] = useState("");
```

```
 const [dishSelected, setDishSelected] = useState([]);
```

```
 const [strMeal, SetStrMeal] = useState("");
```

```
 const [dishSelected2, setDishSelected2] = useState([]);
```

```
 const handleInputText = (value) => {
```

```
 const valueTrim = value.trim();
```

```
 SetIngredient(valueTrim);
```

```
 }
```

```
 const handleInputText2 = (value) => {
```

```
 const valueTrim = value.trim();
```

```
 SetStrMeal(valueTrim);
```

```
 }
```

```
 const handleClickRecipe = (dishID) => {
```

```
 fetch(`https://www.themealdb.com/api/json/v1/1/filter.php?i=${dishID}`)
```

```
 .then(response => response.json())
```

```
 .then(data => {
```

```
 setDishSelected(data.meals);
```

```
 })
```

```
}
```

```
const handleClickRecipe2 = (dishID) => {
```

```
 fetch(`https://www.themealdb.com/api/json/v1/1/filter.php?i=${dishID}`)
```

```
 .then(response => response.json())
```

```
 .then(data => {
```

```
 setDishSelected2(data.meals);
```

```
 })
```

```
}
```

```
return (
```

```
<div className="SearchDish">
```

```
<h1>parent-Element: {ingredient}</h1>
```

```
{/*<InputDish onClick={handleInputText} /> Handling erst nach Klick auf Button*/}
```

```
<InputDish onTyping={handleInputText} />
```

```
<OutputDish ingredient={ingredient} onClickRecipe={handleClickRecipe} />
```

```
<ShowDish dish={dishSelected} />
```

```
<h1>parent-Element2 {strMeal}</h1>
```

```
<InputDish2 onTyping={handleInputText2} />
```

```
<OutputDish2 strMeal={strMeal} onClickRecipe={handleClickRecipe2} />
```

```
<ShowDish2 dish2={dishSelected2} />
```

```
</div>
```

```
)
}
```

export default SearchDish; anpassen, dass er mit der "Meal id" arbeitet ,welche in Output.dish gesucht wird in der API (`https://www.themealdb.com/api/json/v1/1/filter.php?i=\${strMeal}`) und dann übergebe zum onClickRecipe(dish2.idMeal), so dass RecipeDetails.jsx aufgerufen werden kann in Kombination durch die Abfrage der API [www.themealdb.com/api/json/v1/1/lookup.php?i=52772](https://www.themealdb.com/api/json/v1/1/lookup.php?i=52772).

## ChatGPT

Um die Anpassungen vorzunehmen, damit die "Meal id" in der `OutputDish2`-Komponente verwendet wird und `RecipeDetails.jsx` aufgerufen wird, kannst du folgende Änderungen vornehmen:

1. **\*\*Ändere den Aufruf von `handleClickRecipe2` in `SearchDish`:\*\***

```
``jsx

const handleClickRecipe2 = (dishID) => {

 fetch(`https://www.themealdb.com/api/json/v1/1/lookup.php?i=${dishID}`)

 .then(response => response.json())

 .then(data => {

 setDishSelected2(data.meals);

 })

 }

``
```

Hier rufen wir die `lookup.php` API auf, um Details für eine bestimmte Mahlzeit abzurufen.

2. **\*\*Ändere die `onClickRecipe`-Funktion in `OutputDish2`:\*\***

```
``jsx

// OutputDish2.jsx

const OutputDish2 = ({ strMeal, onClickRecipe }) => {
```

```
// ...

return (

<div>

 {/* ... */}

 {dishes?.map((dish2) => (

 <div key={dish2.idMeal} className="card">

 <div className="card-body">

 <h5 className="card-title">{dish2.strMeal}</h5>

 <p className="card-text">Hier steht eine Kurzbeschreibung</p>

 <button onClick={() => onClickRecipe(dish2.idMeal)} className="btn btn-primary">Zum
Rezept</button>

 </div>

 </div>

)

)}

 {/* ... */}

</div>

);

};

'''
```

Hier haben wir `onClickRecipe(dish2.idMeal)` geändert, um die Mahlzeit-ID direkt zu übergeben.

3. **\*\*Ändere `SearchDish` für den Aufruf von `RecipeDetails`\*\***

```
```jsx
```

```
// SearchDish.jsx
```

```
const SearchDish = () => {
```

```
// ...
```

```
return (
```

```
<div className="SearchDish">
```

```
  { /* ... */ }
```

```
<OutputDish2 strMeal={strMeal} onClickRecipe={handleClickRecipe2} />
```

```
<ShowDish2 dish2={dishSelected2} />
```

```
{dishSelected2.length > 0 && (
```

```
<RecipeDetails recipeDetails={dishSelected2} onClose={() => setDishSelected2([])} />
```

```
))
```

```
</div>
```

```
);
```

```
};
```

```
...
```

Hier wird `RecipeDetails` nur gerendert, wenn `dishSelected2` Daten enthält. Der `onClose`-Handler wird verwendet, um `dishSelected2` zu leeren und das `RecipeDetails`-Fenster zu schließen.

Mit diesen Änderungen sollte `RecipeDetails` aufgerufen werden, wenn der "Zum Rezept"-Button in `OutputDish2` gedrückt wird.

User

```
[ <h3>Ingredients:</h3>
```

```
<ul>
```

```
  {Array.from({ length: 20 }, (_, index) => index + 1)}
```

```
    .filter((index) => meal[`strIngredient${index}`])
```



```
.map((index) => (
```

```
<li key={index}>
```

```
{meal[`strMeasure${index}`]} {meal[`strIngredient${index}`]}
```

```
</li>
```

```
)))
```

```
</ul>] wie meinst du genau diesen teil
```

ChatGPT

Der oben gezeigte Code-Abschnitt erstellt eine Liste der Zutaten und Mengen für ein Rezept basierend auf einer JSON-Struktur, die wahrscheinlich von einer API-Antwort stammt. Lassen Sie uns den Code im Detail durchgehen:

```
```jsx
```

```
<h3>Ingredients:</h3>
```

```

```

```
{Array.from({ length: 20 }, (_, index) => index + 1)}
```

```
.filter((index) => meal[`strIngredient${index}`])
```

```
.map((index) => (
```

```
<li key={index}>
```

```
{meal[`strMeasure${index}`]} {meal[`strIngredient${index}`]}
```

```

```

```
)))
```

```

```

```
```
```

1. `<h3>Ingredients:</h3>`: Dies erstellt eine Überschrift "Ingredients" für die Zutatenliste.

2. ``: Dies erzeugt eine ungeordnete Liste, in der die Zutaten und Mengen aufgelistet werden.

3. `{Array.from({ length: 20 }, (_, index) => index + 1) ...}`: Hier wird ein Array von Zahlen von 1 bis 20 erstellt. Dies repräsentiert wahrscheinlich die maximal mögliche Anzahl von Zutaten in einem Rezept.

4. `.filter((index) => meal[`strIngredient${index}`])`: Hier wird das Array gefiltert, um nur diejenigen Indizes zu behalten, für die `meal[`strIngredient${index}`]` einen Wert hat. Dies ermöglicht es, leere Einträge zu überspringen.

5. `.map((index) => ...)`: Hier wird über die gefilterten Indizes iteriert, und für jeden Index wird ein ``-Element erstellt, das die Menge und den Namen der Zutat enthält.

6. `{meal[`strMeasure${index}`]} {meal[`strIngredient${index}`]}`: Hier werden die Menge (`meal[`strMeasure${index}`]`) und der Name der Zutat (`meal[`strIngredient${index}`]`) für jeden Index in der Liste platziert.

Insgesamt wird dies dazu verwendet, eine Zutatenliste mit Mengen für ein Rezept darzustellen. Beachten Sie, dass dies auf der Annahme basiert, dass die Zutaten und Mengen in der API-Antwort unter den Eigenschaften `strIngredient1`, `strMeasure1`, `strIngredient2`, `strMeasure2`, usw. gespeichert sind.