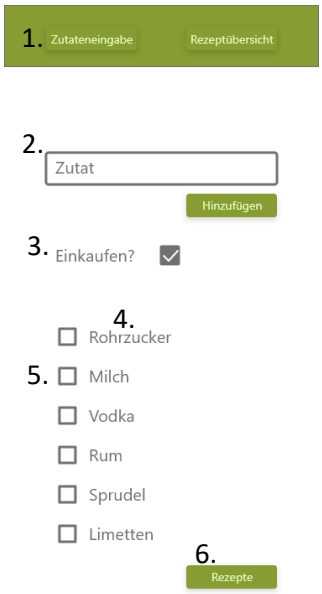
Dokumentation Cocktail App

Mockup/Requirements

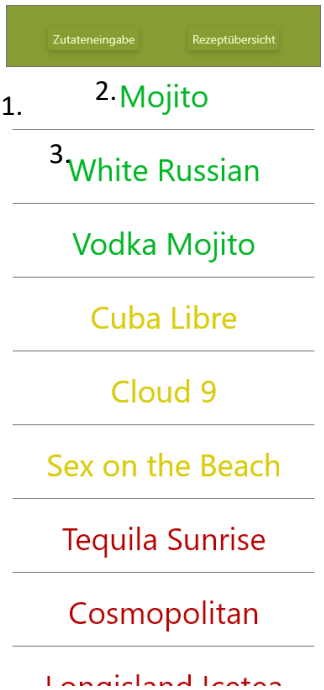
Bei Adobe XD wurde vorab ein Prototyp mit Iphone Bildschirmmaßen erstellt. Dieser dient als Orientierung für alle folgenden Entscheidungen hinsichtlich React Componenten, Klassendiagrammen etc., außerdem werden hier die Anforderungen an die Funktionen der App definiert.

Zutateneingabe:



1. Navigationsleite mit zwei Buttons zum Wechsel zwischen der Zutateneingabe und der Rezeptübersicht
2. Eingabefeld mit Hinzufügen Button – die Zutaten werden der verwendeten Zutatenliste hinzugefügt
3. Checkbox Einkaufen? Durch checken der Checkbox werden auch Rezepte angezeigt, bei denen nicht alle Zutaten vorhanden sind
4. Liste mit hinzugefügten Zutaten
5. Durch checken der Checkbox, wird die entsprechende Zutat aus der Liste entfernt
6. Rezept-Button zur Start der Suche

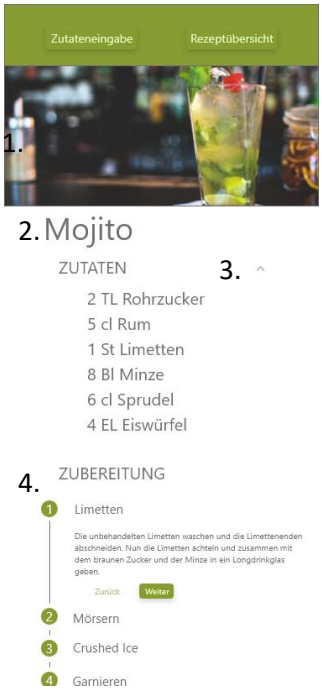
Rezeptübersicht:



1. Liste, sortiert nach höchstem Match-Faktor, mit möglichen Rezepten, basierend auf:
   - Zutaten
   - Einkaufen? (Ja/Nein)
2. Matching des Rezepts basierend auf den angegebenen Zutaten.
   - Grün (Match = 100%)
   - Gelb (Match < 100% & >=50%)
   - Rot (Match < 50%)
3. Jedes Element der Liste ist ein Link zur jeweiligen Rezeptseite

Rezept:

1. Bild des Cocktails
2. Name des Cocktails
3. Ausklappbare Liste mit Zutaten des Cocktails
4. Stepper mit Zubereitungsschritten – durch die Buttons Weiter und Zurück kann zwischen den Schritten gesrpungen werden



Notizen zum Unterschied zwischen Mockup und Endprodukt:

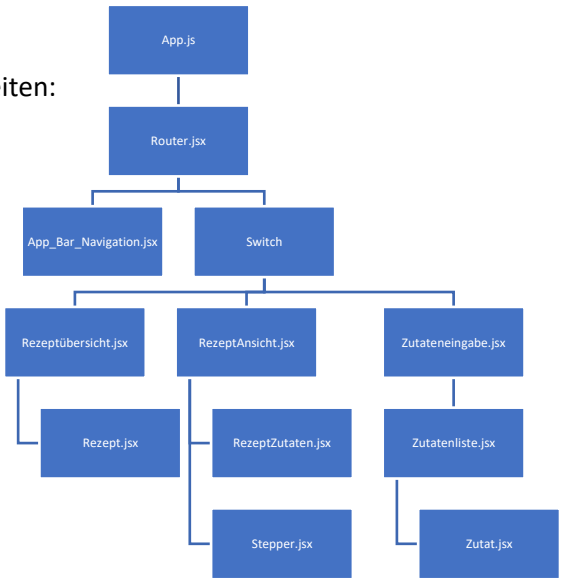
- Der Button „Rezepte“ auf der Seite Zutateneingabe existiert nicht. Die Navigation zu der Rezeptübersicht und zurück läuft über die Navigationsleiste.
- Die Buttons in der Navigationsleiste sind nicht beschrieben, sondern beeinhalten Icons, welche deutlich die Funktion der jeweiligen Seite widerspiegeln.

Klassendiagramm/Konzept

Der Aufbau der React.js App ist rechts zu sehen. Die **App.js** rendert die **Router.jsx**. Innerhalb der Router.jsx wird der React Router definiert, welcher es durch den **Switch** ermöglicht, zwischen den verschiedenen Seiten:

- Rezeptübersicht.jsx (URL: /Rezeptübersicht)
- RezeptAnsicht.jsx (URL: /Rezept:id)
- Zutateneingabe.jsx (URL: /)

zu wechseln. Innerhalb dieses Routers wird auch die **App_Bar_Navigation.jsx** gerendert. Diese wird über alle Routen hinweg gerendert.



In der Router.jsx werden state variablen die eingegebenen Zutaten und ein boolean Wert für „Einkaufen“ gespeichert. Dieser State wird auch über localstorage in den Cache bei jeder Änderung geschrieben und bei einem Component Mount wieder abgerufen.

Innerhalb der Router.jsx werden die Methoden:

- updateIngredients
- deleteIngredients
- handleChangeShopping

definiert.

Diese Methoden und die state Variablen werden an Zutateneingabe.jsx und Rezeptübersicht.jsx über props weitergegeben.

Die **Zutateneingabe.jsx** nutzt CallBacks um Werte des State in Router.jsx zu ändern bspw. Zutaten zu der Liste hinzufügen/löschen oder den Einkaufen boolean Wert ändern. Außerdem iterriert die **Zutatenliste.jsx** über die Zutaten Liste und gibt die einzelnen Zutaten als **Zutat.jsx** zurück. Die Zutat.jsx gibt den Namen und eine Checkbox, die bei Klick die deleteIngredients Funktion Call Back ausführt, zurück.

Die **Rezeptübersicht.jsx** lädt im ersten Schritt die Firebase Datenbank „Rezepte“ herunter und fügt deren Objekte ({Name: String, Ingredients: Array}) einem Array im State hinzu. Im nächsten Schritt werden die Matchingraten der einzelnen Objekte berechnet. Hierfür wird für jedes Rezept Objekt der Ingredients Array mit dem über die props gereichten Zutaten Array verglichen. Es wird die Anzahl an übereinstimmenden Strings festgehalten und durch die Länge des Ingredients Arrays des Rezept Objekts geteilt. Dieser Float Wert wird dann dem Objekt hinzugefügt, somit sieht ein Objekt am Ende wie folgt aus: {Name: String, Ingredients: Array, MatchingRate: Float}. Diese Objekte werden innerhalb des Arrays absteigend nach MatchingRate sortiert. Nun wird über den Rezepte Array iterriert, hierbei wird jedes Objekt an **Rezept.jsx** gereicht und von diesem als Button mit der an der Matchingrate orientierten Hintergrundfarbe zurückgegeben (grün = 1, gelb >=0.5, rot < 0.5). Falls „Einkaufen“ nicht gesetzt wurde, werden nur die Rezepte zurückgegeben, deren Matchingrate bei 1 liegt. Der Inhalt der Buttons ist jeweils ein Link zu URL: /Rezept/+Name des geklickten Rezepts.

Die **RezeptAnsicht.jsx** wird gerendert, wenn die Route URL: /Rezept/:id geladen wird. Hierbei wird über id der Name des Rezeptes mitgegeben. Die RezeptAnsicht beinahltet drei Elemente:

1. den Namen des Rezepts
2. RezeptZutaten.jsx
3. Stepper.jsx

Nachdem das gesuchte Rezept aus der Firebase Datenbank Rezepte als {Name: String, Ingredients: Array, Amount: Array, Description: Array, Steps: Array} heruntergeladen wurde, wird über die **RezeptZutaten.jsx** in einer Table die Amounts den Ingredients gegenüber gestellt. Die **Stepper.jsx** ist eine Material UI Komponente und nutzt die Steps und Description Arrays des Rezept Objekts.

Generell lässt sich sagen, dass alle Elemente über Material UI Grid angeordnet wurden und auch in den meisten Fällen Material UI Komponenten genutzt wurden für bspw. Buttons.

Alle React externen genutzten Libaries sind:

- Axios (Datenbank pull)
- Material UI (GUI Elemente)

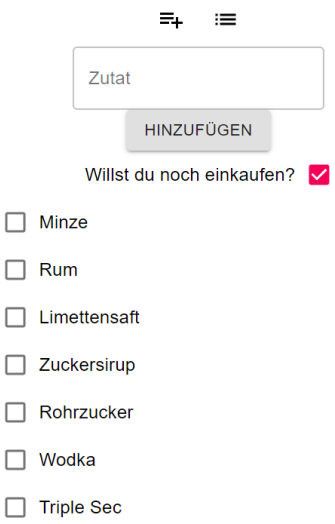
Abschluss

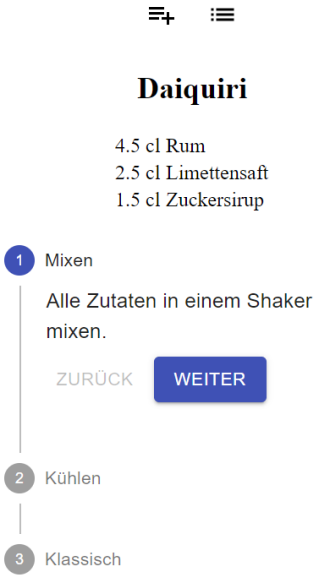
Die Anwendung erfüllt alle an sie gestellten Anforderungen und ist voll funktionsfähig unter https://dhbwws20.web.app/ erreichbar.

Mögliche Verbesserungen:

- Bilder auf den einzelnen Rezept Seiten darstellen
- Index mit Farbe und MatchinRate bereitstellen innerhalb der App
- Zutatenvorschläge bei der Zutateneingabe
- Umfangreichere Cocktail Datenbank
- Mengenangaben bei Zutateneingabe

Screenshots der fertigen Anwendung:





Vielen Dank fürs Lesen!