Projektbezeichnung	yourChoice
Projektleiter	Josua Weber
Autor	Jonas Hauß
Erstellt am	28.03.2018
Zuletzt geändert am	28.03.2018
Bearbeitungszustand	Erste Version fertiggestellt
Dokumentbezeichnung	Frontend-Modulbeschreibung.docx
Seitenanzahl	8

Historie der Dokumentversionen:

Version	Datum	Autor	Änderungsgrund / Bemerkung
1.0	28.03.2018	Jonas Hauß	Ersterstellung

Frontend-Modulbeschreibung

Der Einsatz von React erlaubt es die Frontend-Anwendung als System von Komponenten zu entwerfen. Die Webseite wird nicht als Sammlung von einzelnen Seiten erstellt, sondern mit Hilfe von Komponenten zusammengesetzt. Diese Bausteine der Benutzeroberfläche werden in separaten Gruppen organisiert. Der Hintergrundgedanke ist eine klare Trennung einzelner Zuständigkeiten.

Des Weiteren werden funktionale Bestandteile der Benutzeroberfläche (UI) ebenfalls in Module untergebracht. Somit können z.B. Dienste oder auch Hilfsfunktionen klar von der restlichen Anwendung getrennt werden. Dies erlaubt einen wiederverwendbaren Einsatz der Geschäftslogik ohne sie direkt in der Benutzeroberfläche integrieren zu müssen.

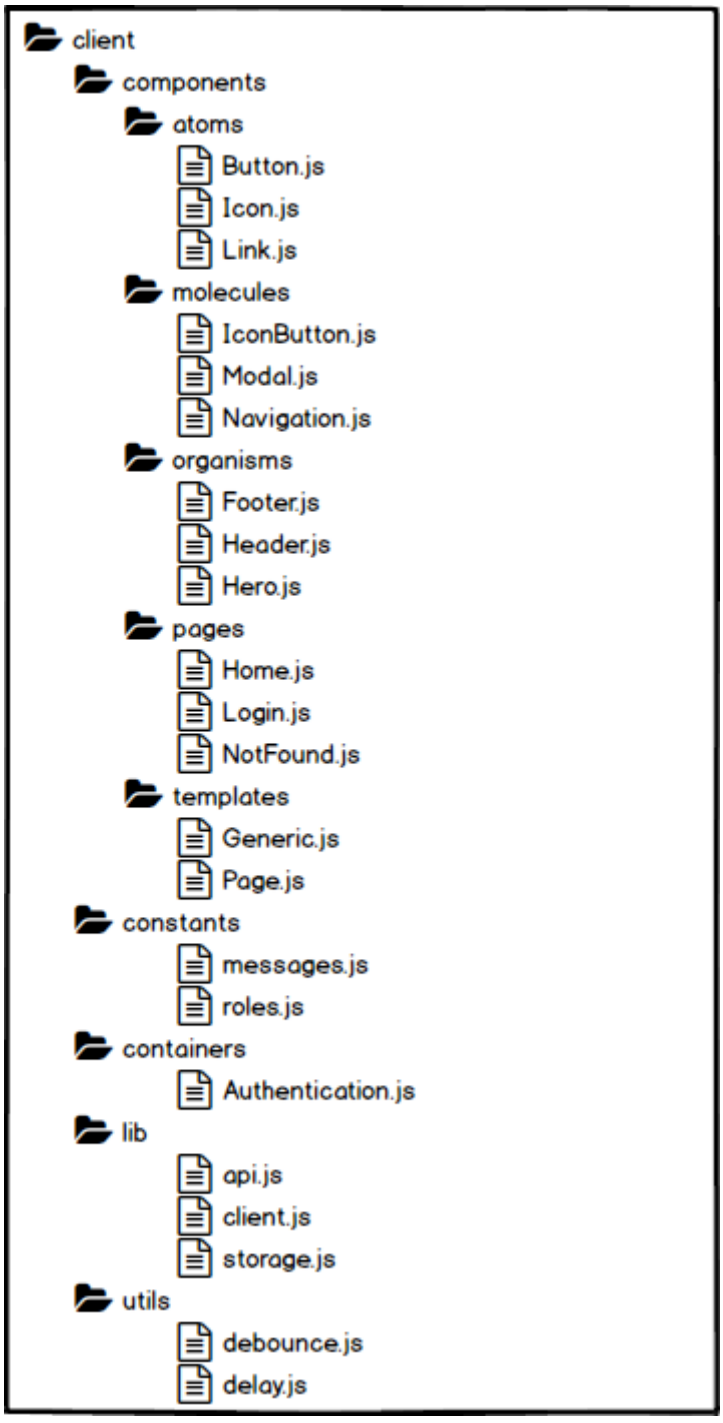

Abbildung 1: Exemplarische Datei-/Verzeichnisstruktur

1. “components”

React ist eine Bibliothek zur Erstellung von Benutzeroberflächen. Eine klare Struktur bzw. ein Rahmen wird dabei nicht vorgegeben. Komponenten können beliebig aufgeteilt und wiederverwendet werden. Die Modularisierung gibt dabei vor wie sich schlussendlich die gesamte Anwendung aus atomaren Elementen zusammensetzt.

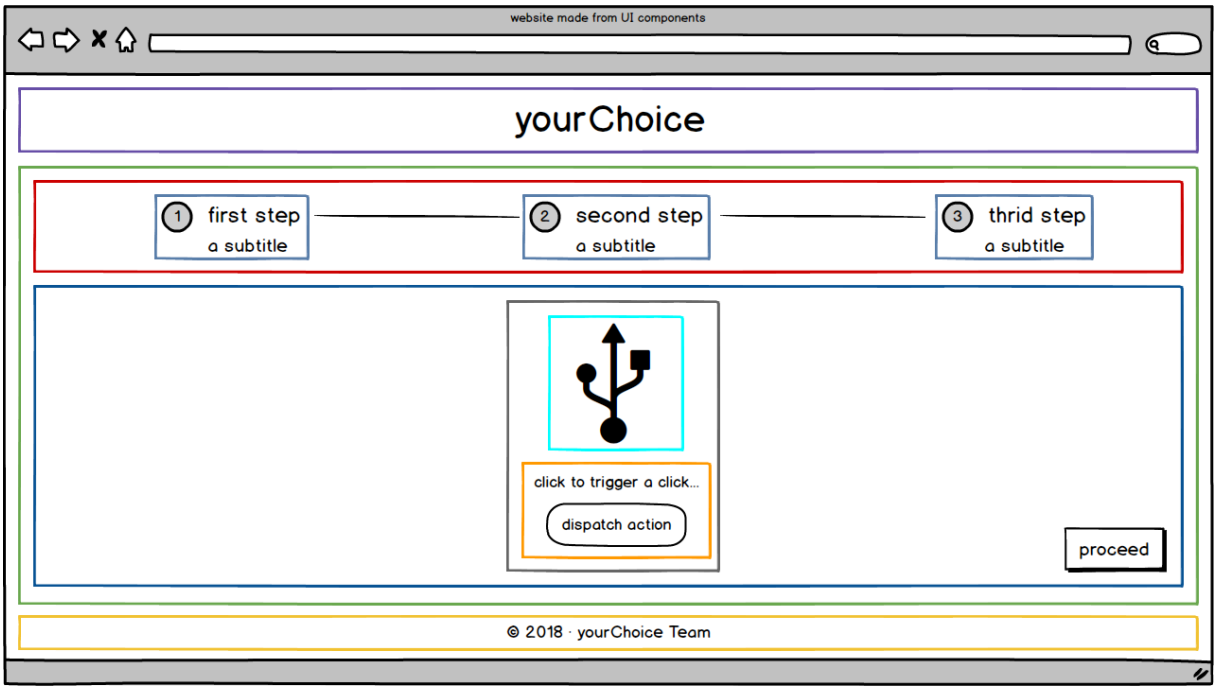

Abbildung 2: Beispiel einer komponentenbasierten Benutzeroberfläche

1.1. Übersicht

Bei den hier aufgeführten Komponenten-Typen handelt es sich hauptsächlich um funktionslose Bestandteile der Benutzeroberfläche. Sprich diese sind zustandslos und somit rein zur Darstellung von Inhalten gedacht. Das Verhalten bzw. die eingehenden Benutzerinteraktionen werden von Containern separat bereitgestellt. Somit können Daten von ihrer Darstellung separiert werden.

Die nachfolgenden Bestandteile bzw. Module der Benutzeroberfläche werden in folgende Gruppen eingeteilt, wobei von oben nach unten gesehen sich die Komponenten immer weiter aus darüber liegenden Gruppen zusammensetzen. Grundsätzlich gibt es keine strikte Trennung da schlussendlich jeder Komponenten-Typ von React gleich behandelt wird. Die Strukturierung dient zur Steigerung der Modularität und der daraus folgenden Wart- und Testbarkeit. Klar getrennte und isolierte Bestandteile des UIs lassen sich geschickter verwalten.

1.2. “atoms”

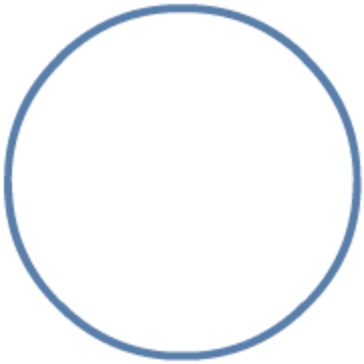

Abbildung 3: Atomare Komponente

Atome sind die kleinstmögliche Form von UI-Komponenten. Unter anderem werden einzelne native HTML-Tags, einfache React-Komponenten oder Komponenten aus Bibliotheken zu diesen atomaren Bestandteilen einer Benutzeroberfläche dazugezählt. Atome dienen einem einzelnen Zweck. Sie bilden den Grundbestandteil des UIs.

z.B. ein Text-Eingabefeld

```
const Input = props => <input {...props} />
```

1.3. “molecules”

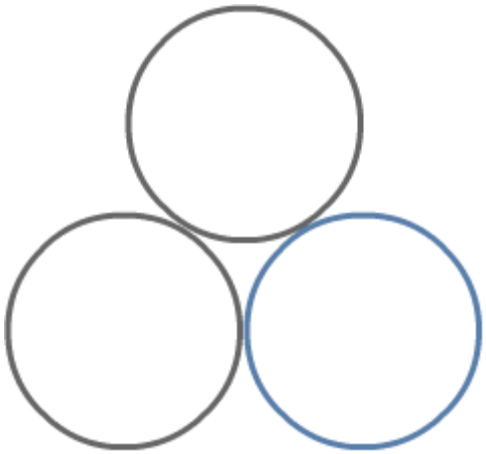

Abbildung 4: Molekül zusammengesetzt aus Atomen

Ein Molekül setzt sich aus einer Gruppe von Atomen zusammen. Die Gruppe zusammen repräsentiert ein weiterer Baustein des UIs. Diese Gruppierung ist für Elemente gedacht die in Kombination als eine Einheit arbeiten.

z.B. ein Formular-Feld mit Label

```
const Field = ({ label, ...props }) => (
  <Label>
    {label}
    <Input {...props} />
  <Label>
)
```

1.4. “organisms”

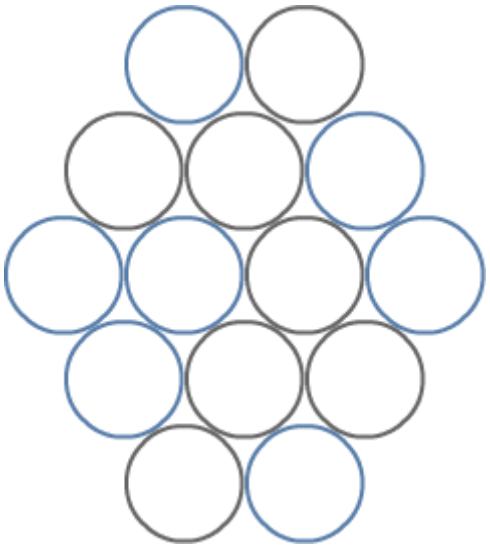

Abbildung 5: Organismus zusammengesetzt aus Molekülen und/oder Atomen

Organismen stellen die nächstgrößere Form da. Sie setzen sich aus einer Gruppe von Atomen, Molekülen und/oder anderen Organismen zusammen. Diese UI-Elemente fallen in der Regel komplex aus und stellen einen deutlichen Abschnitt der Oberfläche dar.

z.B. ein Formular für Benutzereingaben

```
const Form = (props) => (
  <form {...props}>
    <Field label="Name" type="text" />
    <Field label="Email" type="email" />
  <form>
)
```

1.5. “templates”

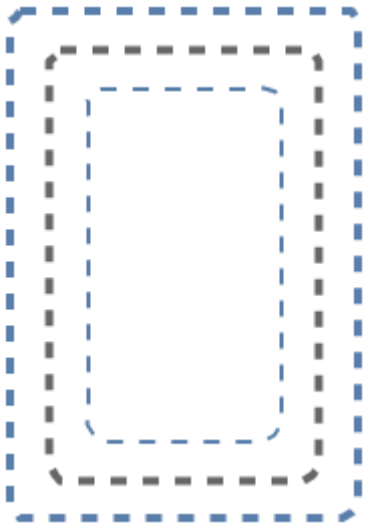

Abbildung 6: Template zur Wiederverwendung eines Seitenlayouts

Templates werden zur Erstellung von Layouts verwendet. Sie dienen als Rahmen für einzelne Seiten. So können zum Beispiel Elemente, die auf allen Seiten benötigt werden, stets gleich platziert werden. Insgesamt sind Templates der Verbund von recht abstrakten Molekülen/Organismen. Sie stellen diese Einheiten zusammen dar und repräsentieren somit das Grundgerüst einer jeden Szene.

z.B. die Standard-Vorlage einer Seite

```
const PageTemplate = ({ header, children }) => (
  <main>
    {header && <div>{header}</div>}
    {children}
  </main>
)
```

1.6. “pages”

Abbildung 7: Einzelne Seite bzw. Szene basierend auf Komponenten

Seiten oder auch Szenen setzen sich hauptsächlich, aber nicht ausschließlich, aus Organismen zusammen. Sie sind die reale Repräsentation bzw. Darstellung eines Templates.

z.B. eine Login-Seite mit Formular-Feld und Seiten-Kopfleiste

```
const LoginPage = () => (
  <PageTemplate header={<Header />}>
    <Form />
  </PageTemplate>
)
```

2. “containers”

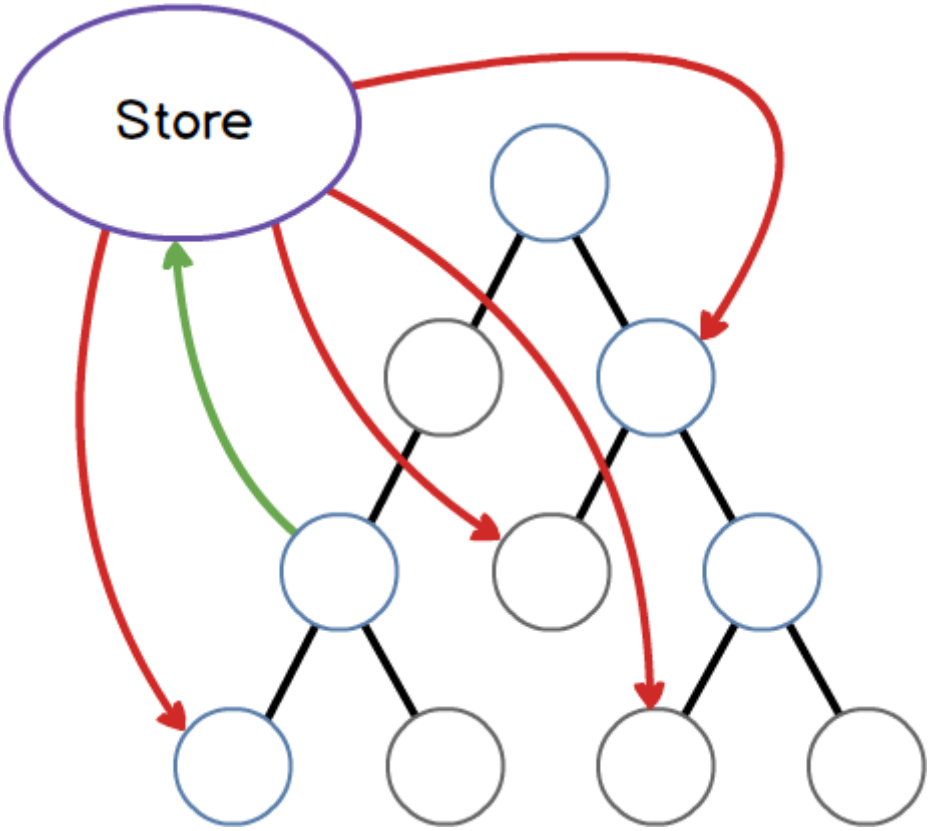

Abbildung 8: Store Komponenten-Anbindung über Container/Subscriber

Ein modularer Ansatz erfordert ebenfalls eine Auslagerung des globalen Anwendungszustands. Immerhin zeichnet React sich besonders im Neu-Rendern von Änderungen aus. Es muss also über eine zentrale Schnittstelle erfasst werden können welche Daten sich wirklich ändern. Damit kann der React-Render-Algorithmus die Änderungen im DOM feststellen und nur diese austauschen.

Ebenfalls gehört das Verhalten von Komponenten nicht zu ihrer eigentlichen Darstellung. Mit Hilfe von Containern können Daten getrennt verwaltet werden. Diese verknüpfen den globalen Zustand mit den gewählten Komponenten. So kann das Verhalten und die darzustellenden Informationen von einer zentralen Stelle an alle Unterelemente weitergereicht werden.

Der globale Zustand wird als Store bezeichnet und durch einen sogenannten Provider allen Unterkomponenten zur Verfügung gestellt. Damit der Zustand nicht an jede Komponente weitergereicht wird, muss eine Subkomponente, die auf den Store zugreifen möchte, unter Einsatz einer Subscriber-Komponente mit diesem verknüpft werden. Die Subkomponente wird in diesem Fall als Container bezeichnet und stellt der darunterliegenden Komponente Funktionen und Daten bereit.

3. Weitere Module

3.1. “constants”

Konstante Werte die anwendungsübergreifend verwendet werden, können über dieses Modul bereitgestellt werden. Allgemein sollte Code nicht unnötig wiederholt sondern wiederverwendet werden.

3.2. “lib”

Nicht alles an Modulen kann in Komponenten oder Containern eingeteilt werden. Untermodule in diesem Modul werden für sich gekapselt behandelt und definieren den Kern der applikationsweiten Geschäftslogik. Die einzelnen Dienste können zwischen Komponenten, Seiten und Szenen ausgetauscht werden. Sie stellen eine Brücke zwischen der Serveranwendung und dem hier beschriebenen Frontend dar. Grundsätzlich werden Seiteneffekte z.B. Netzwerkanfragen oder das Abspeichern von Cookies über solche Module ausgeführt.

3.3. “utils”

Kleine Helfer-Funktionen die weder in einen Dienst noch eine Bibliothek passen, werden in diesem Modul definiert. Zum Beispiel Funktionen zum Verzögern von Benutzereingaben oder dem Formatieren von Daten. Meistens können diese Einheiten nicht klar zugeordnet werden, da sie die unterschiedlichsten Aufgaben erfüllen.