JavaScript und JavaScript Frameworks

React – Erste Schritte

Übersicht und Zielstellung

- In diesem Abschnitt erfahren Sie:
  - wie ein React-Projekt organisiert ist,
  - wie Medien eingebunden werden,
  - wie man Komponenten variabel einsetzen kann,
  - wie man Inhalte aus einem Array erstellen kann,
  - wie man mit Komponenten verschachteln kann,
  - Wie man eine einfache Navigation umsetzen kann

Start in das Thema

CodeSandbox.io zur Entwicklung nutzen

Die vorliegenden Dateien

index.js

```
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";

import App from "./App";

const rootElement = document.getElementById("root");
const root = createRoot(rootElement);

root.render(
   <StrictMode>
      <App />
   </StrictMode>
);
```

Die erste Zeile importiert die React-StrictMode-Komponente aus der React-Bibliothek. StrictMode ist ein Tool, das Entwicklern hilft, potenzielle Probleme in ihren React-Code zu finden und zu beheben, indem es zusätzliche Warnungen und Fehlermeldungen im Console-Output generiert.

Die zweite Zeile importiert die "createRoot" Funktion aus der "react-dom/client" Bibliothek. "createRoot" ist eine neue Methode, die seit React 18 verfügbar ist und die Möglichkeit bietet, eine App im Concurrent Mode zu rendern.

Die dritte Zeile importiert die "App" Komponente aus einer separaten Datei namens "App.js".

In der vierten Zeile wird das DOM-Element "root" aus dem HTML-Dokument ausgewählt und der Konstante "rootElement" zugewiesen.

Die fünfte Zeile ruft die "createRoot" Funktion auf und gibt das DOM-Element "rootElement" als Argument an. Das Ergebnis wird in der Konstante "root" gespeichert.

In der sechsten Zeile wird die "render" Methode auf "root" aufgerufen, um die "App" Komponente in das "root" DOM-Element zu rendern. Die "App" Komponente wird in ein <StrictMode> Element eingewickelt, um Entwickler zu warnen, falls sie veraltete Funktionen oder Technologien verwenden.

CodeSandbox.io zur Entwicklung nutzen

Die vorliegenden Dateien

App.js

```
import "./styles.css";

export default function App() {
   return (
      <div className="App">
         <h1>Hello CodeSandbox</h1>
         <h2>Start editing to see some magic happen!</h2>
      </div> );
}
```

Dies ist ein einfaches React-Komponenten-Codebeispiel, der eine HTML-DOM-Struktur (h1 und h2-Elemente) zurückgibt. Die CSS-Styles für diese HTML-Elemente werden in einer separaten Datei namens "styles.css" definiert und in die Komponente importiert..

Die Zeile `import "./styles.css";` importiert die Styles aus der "./styles.css" Datei und fügt sie in die Komponente ein. Dadurch können die definierten Styles auf die HTML-Elemente angewendet werden.

Die Funktion "App" ist die Hauptkomponente und sie gibt die HTML-Struktur zurück. Die "className" -Eigenschaft wird verwendet, um der Wurzel-Div einen Klassennamen "App" zuzuweisen, auf den in der CSS-Datei verwiesen werden kann.

CodeSandbox.io zur Entwicklung nutzen

Die index.js

Das Grundprinzip

```
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";

import App from "./App";

const rootElement = document.getElementById("root");
const root = createRoot(rootElement);

root.render(
   <StrictMode>
      <App />
   </StrictMode>
);
```

Im HTML-Dokument wird ein Element gesucht – die ID root.

Für das React-Rendern wird das Element übergeben.

Dann wird der Inhalt für das Element gerändert.

Der eigentliche Inhalt ist in der vorher importierten Komponente App zusammengesetzt

CodeSandbox.io zur Entwicklung nutzen

Die Dateistruktur

Nun geht es darum, weitere Komponenten zu erstellen und einzubinden

```
src/
├── App.js
├── index.js
├── components/
│   ├── Header.js
│   ├── Footer.js
│   ├── Home.js
│   └── ...
└── ...
```

Alle Quell-Dateien liegen im Ordner src

Es bietet sich an, mit Unterordnern eine gewisse Ordner-Struktur zu definieren

Im Ordner components sind nun ein paar Komponenten definiert

Die Komponente App sollte im src-Ordner liegen

In der Komponente App werden die Komponenten zusammengefügt.

CodeSandbox.io zur Entwicklung nutzen

Der einfache Aufbau einer Komponente

Schauen wir uns den Aufbau einer einfachen Komponente an:

```
// components/Header.js

import React from 'react';

function Header() {
  return (
    <header>
      <h1>Header</h1>
    </header>
  );
}

export Header;
```

Als erster wird der React Bestandteil geladen

Dann wird eine export Funktion für die Ausgabe definiert
Der Name sollte dem Componenten-Namen entsprechen.

Für die Rückgabe wird nun der Code definiert (hier einfacher HTML Code)

So können individuelle Komponenten definiert werden.

CodeSandbox.io zur Entwicklung nutzen

Die Haupt-Komponente

Nun muss die Komponenten noch in die App eingebunden werden:

```
// App.js

import React from 'react';
import { Header } from './components/Header';

function App() {
  return (
    <div>
      <Header />
    </div>
  );
}

export default App;
```

Als erster wird der React Bestandteil geladen

Dann wird die Komponenten importiert.

Die Komponente wird in die App eingebunden

Durch export wird die Gesamtkomponente ausgeliefert und gerändert.

Aufgabe: Erstellen Sie eine Komponente für dem Inhalt und eine Komponente für den Fuß.

CodeSandbox.io zur Entwicklung nutzen

Bilder in einer Komponente

Alle üblichen Medien sind für den Einsatz in einer Komponente denkbar.

Die Dateien müssen im Ordner public und dort in eventuellen Unterordnern gespeichert werden.

Das Einbinden der Datei in einer Komponente erfolgt wie im „normalen“ HTML.

```
import React from 'react';

function MyComponent() {
   return (
      <div>
          <img src="/pfad/zum/bild.jpg" alt="Bildbeschreibung" />
     </div> );
}

export default MyComponent;
```

Achten Sie darauf, der pfad/zum Bezug auf den öffentlichen Root-Ordner **public** vergeben werden muss.

CodeSandbox.io zur Entwicklung nutzen

Komponenten mit Properties

Eine Komponenten kann auch für einen mehrfachen Einsatz vorbereitet werden.

Durch die Übergabe von Parameter(props) können Komponenten mit Platzhaltern ausgestattet werden.

```
// components/Visitenkarte.js

import React from 'react';

export function Visitenkarte(props) {
  return (
    <div>
      <h1>Angaben zur Person</h1>
      <p>{props.vorname} {props.famname}</p>
    </div>
  );
}
```

Die „Platzhalter“ werden im Code mit angegeben.

Sie werden in {} eingebunden.

Die Bezeichnung setzt sich zusammen aus dem Parameternamen in der Funktion und dem Namen, der in der App verwenden wird.

CodeSandbox.io zur Entwicklung nutzen

Die Properties auffüllen

Nun muss die Komponenten noch in die App eingebunden werden:

```
// App.js

import React from 'react';
import { Header } from './components/Header';
import { Visitenkarte } from './components/Visitenkarte';


function App() {
  return (
    <div>
      <Header />
      <Visitenkarte vorname="Gerald" famname="Reinhardt" />
    </div>
  );
}

export default App;
```

Beim Einbinden der Komponente können die Werte definiert werden.

Achten Sie darauf, dass die „Attributnamen“ den entsprechen, die in der Komponente verwendet werden.

CodeSandbox.io zur Entwicklung nutzen

Komponenten auf der Basis von Arrays einbinden

Angenommen, Sie haben eine Liste von Benutzern, die Sie als Array darstellen möchten.
Jeder Benutzer in der Liste hat einen Namen und ein Alter:

```
const users = [
  { name: 'Alice', age: 25 },
  { name: 'Bob', age: 30 },
  { name: 'Charlie', age: 35 },
];
```

Sie können nun eine User-Komponente erstellen, die den Namen und das Alter eines Benutzers als Props erhält und diese Informationen in JSX darstellt:

```
function User(props) {
  return (
    <div>
      <h2>{props.name}</h2>
      <p>Age: {props.age}</p>
    </div>
  );
}
```

CodeSandbox.io zur Entwicklung nutzen

Komponenten auf der Basis von Arrays einbinden

Um nun die User-Komponente für jeden Benutzer in der Liste anzuzeigen, können Sie eine UserList-Komponente erstellen, die das users-Array als Prop erhält und eine User-Komponente für jeden Benutzer in der Liste rendert:

```
function UserList(props) {
  return (
    <div>
      {props.users.map(user => (
        <User key={user.name} name={user.name} age={user.age} />
      ))}
    </div>
  );
}
```

CodeSandbox.io zur Entwicklung nutzen

Komponenten auf der Basis von Arrays einbinden

Die UserList-Komponente rendert also eine Liste von User-Komponenten, wobei jeder Benutzer in der users-Liste als Prop an die User-Komponente übergeben wird.
Sie können dann die UserList-Komponente in Ihrer Haupt-App-Komponente wie folgt einbinden:

```
function App() {
  const users = [
    { name: 'Alice', age: 25 },
    { name: 'Bob', age: 30 },
    { name: 'Charlie', age: 35 },
  ];

  return (
    <div>
      <h1>User List</h1>
      <UserList users={users} />
    </div>
  );
}
```

CodeSandbox.io zur Entwicklung nutzen

Komponenten in einander verschachteln

```
function OuterComponent() {
  return (
    <div>
      <h2>Outer Component</h2>
      <InnerComponent />
    </div>
  );
}

function InnerComponent() {
  return (
    <div>
      <h3>Inner Component</h3>
      <p>Das ist der Inhalt der inneren Komponente.</p>
    </div>
  );
}
```

```
function App() {
  return (
    <div>
      <h1>My App</h1>
      <OuterComponent />
    </div>
  );
}
```

In diesem Beispiel wird die OuterComponent-Komponente erstellt, die eine Überschrift enthält und die InnerComponent-Komponente rendert. Die InnerComponent-Komponente selbst enthält eine Überschrift und einen Absatz.

Wenn Sie nun die OuterComponent-Komponente in Ihrer Haupt-App-Komponente rendern, wird die InnerComponent-Komponente ebenfalls gerendert und in der OuterComponent-Komponente verschachtelt.

CodeSandbox.io zur Entwicklung nutzen

Navigation innerhalb von React

Man kann eine einfache Navigation innerhalb der Anwendung erstellen, indem man den *Zustand* der Anwendung verwendet und die Anzeige der Komponenten basierend auf diesem *Zustand* ändert.

Schauen wir uns das für drei Komponenten

Nutzen wir für das Beispiel drei Komponten

```
import React from "react";

function Contact() {
   return (
      <h1>Kontact</h1>
   );
}

export default Contact;
```

```
import React from "react";

function About() {
   return (
      <h1>Über uns</h1>
   );
}

export default About;
```

```
import React from "react";

function Home() {
   return (
      <h1>Herzlich Willkommen</h1>
   );
}

export default Home;
```

CodeSandbox.io zur Entwicklung nutzen

Navigation innerhalb von React

In der App.js wird nun die Navigation definiert

```
import React, { useState } from 'react';
import Home from ,./components/Home';
import About from './components/ About';
import Contact from './components/ Contact';
```

Beim Importieren vom React wird der **Status** für die spätere Verarbeitung (Klicken) mit angegeben.

```
function App() {
 const [page, setPage] = useState('home');

 const handleNavigation = (newPage) => {
    setPage(newPage);
};
```

Dann wird die Funktion der App definiert

Für die Auswertung der Navigation wird eine Konstante definiert

CodeSandbox.io zur Entwicklung nutzen

Navigation innerhalb von React

Die Wert für die spätere Einbindung der Komponente wird definiert

```
let pageContent;
if (page === 'about') {
    pageContent = <About />;
  } else if (page === 'contact') {
    pageContent = <Contact />;
  } else {
    pageContent = <Home />;
}
```

Der Container für die spätere Zuweisung wird definiert.

In einer if-Abfrage wird der übergebene Wert geprüft und im Ergebnis wird der Code für die Komponente festgelegt.

CodeSandbox.io zur Entwicklung nutzen

Navigation innerhalb von React

Dann wird der Anzeige-Inhalt definiert

```
return (
   <div>
      <nav>
         <ul>
            <li onClick={() => handleNavigation('home')}>Home</li>
            <li onClick={() => handleNavigation('about')}>About</li>
            <li onClick={() => handleNavigation('contact')}>Contact</li>
        </ul>
      </nav>

      {pageContent}

   </div>
  );
}

export default App;
```

Die Link-Liste für Navigation

Die Komponente, die in der IF-Bedingung gesetzt wurde

VIELEN DANK!