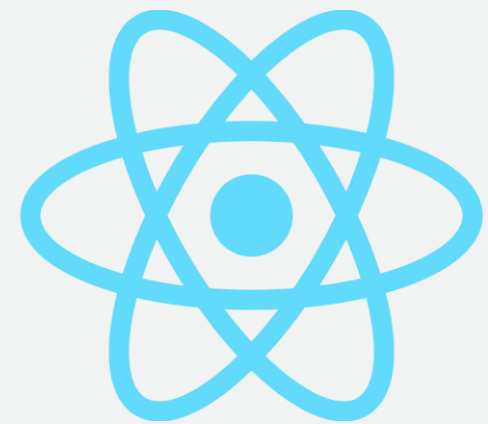
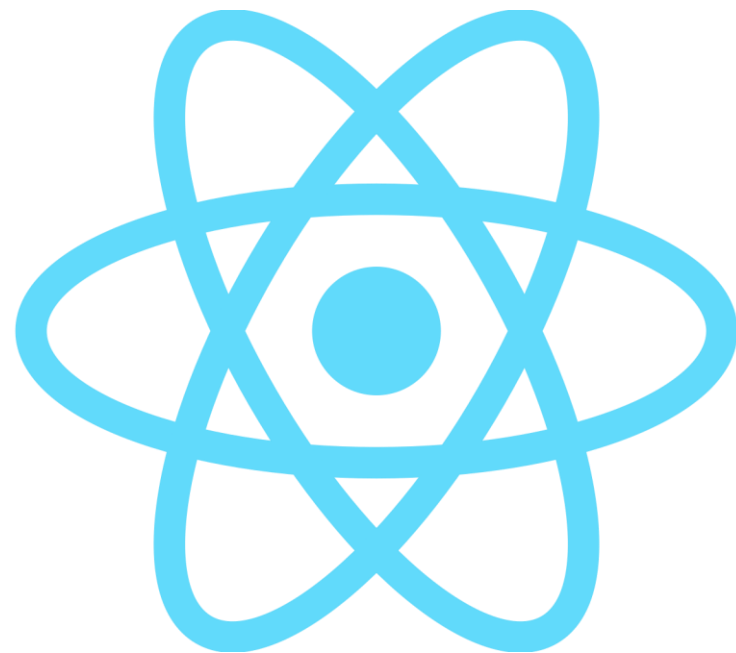


Grundlagen zu React.js

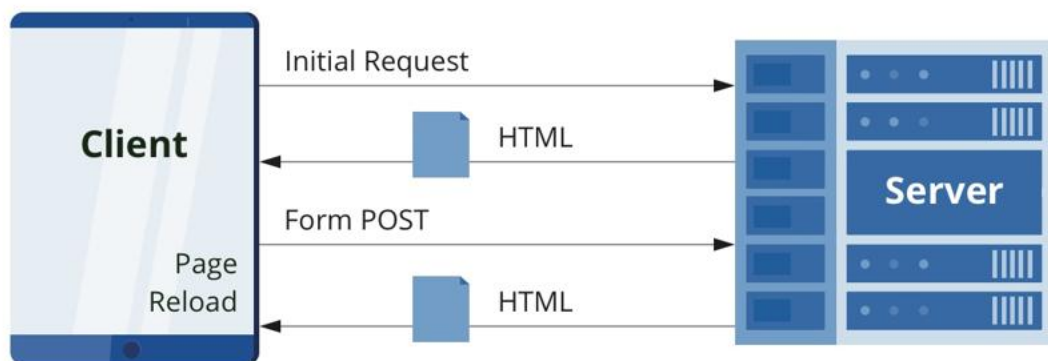


Übersicht

- Traditionelle Websites vs SPA
- Was ist React.js?
- Create-React-App
- React Elemente
 - React Komponenten
 - JavaScript XML (JSX)
 - React Props
 - Conditional & List Rendering



Traditional Page Lifecycle

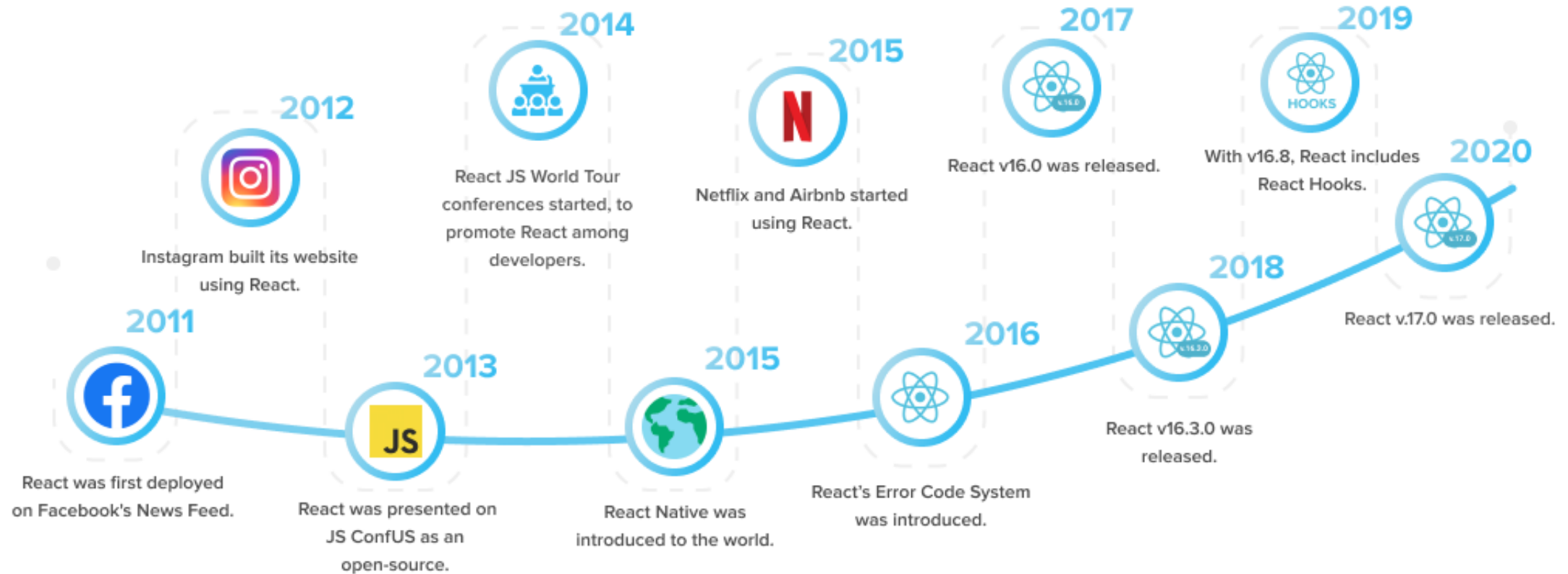


SPA Lifecycle



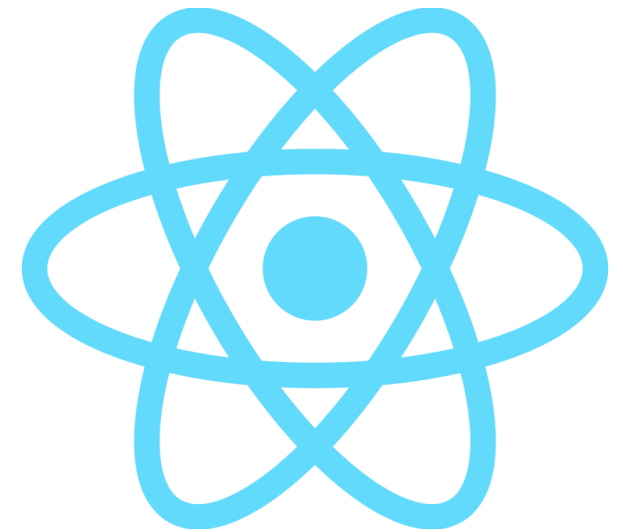
	SPA	MPA
Sleek UX	✓	✗
Easy SEO	✗	✓
Security	✗	✓
Less server load	✓	✗
Offline functionality	✓	✗
Mobile adaptability	✓	✗
Application scalability	✗	✓
UI/data separation	✓	✗
Speed	✓	✗
Fast launch	✗	✓
Works without JavaScript	✗	✓

HISTORY BEHIND REACT



Was ist React.js?

- React ist eine JavaScript-Bibliothek zur Entwicklung von Benutzeroberflächen.
- Es ermöglicht die Erstellung von interaktiven und dynamischen Webanwendungen.
- React basiert auf einem komponentenbasierten Ansatz, der die Entwicklung und Wartung erleichtert.
- React bietet eine verbesserte Leistung und Effizienz bei der Aktualisierung von Benutzeroberflächen, indem immer nur einzelne Komponenten der Seite aktualisiert werden



<https://react.dev/learn>

Create-React-App

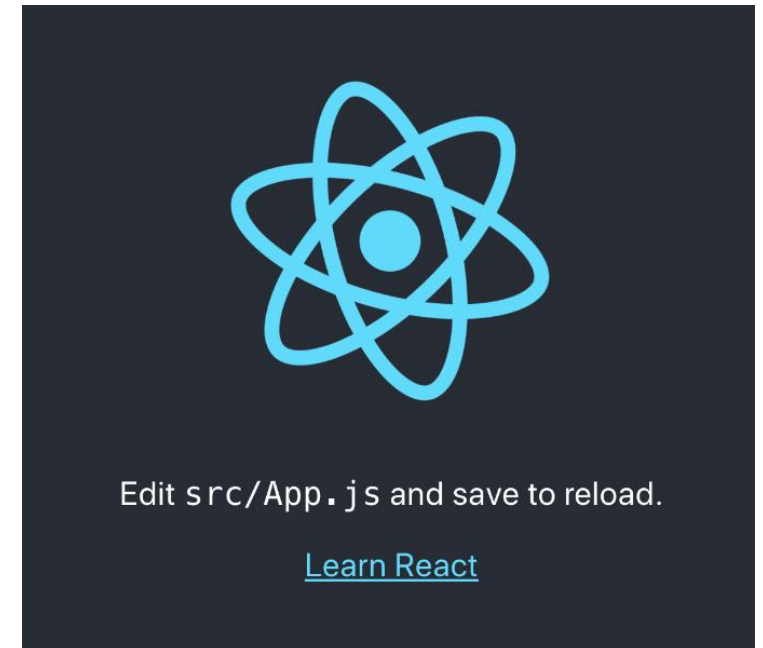
- Create React App ist eine offiziell unterstützte Methode zur Erstellung von Single-Page React-Anwendungen. Es bietet eine moderne Build-Umgebung ohne Konfiguration.
- Du musst keine Tools wie webpack oder Babel installieren oder konfigurieren. Sie sind bereits vorinstalliert und verborgen, sodass du dich auf den Code konzentrieren kannst.
- Erstelle ein Projekt mit dem Befehl „npx create-react-app [Name deines Projekts]“ und du kannst sofort loslegen.



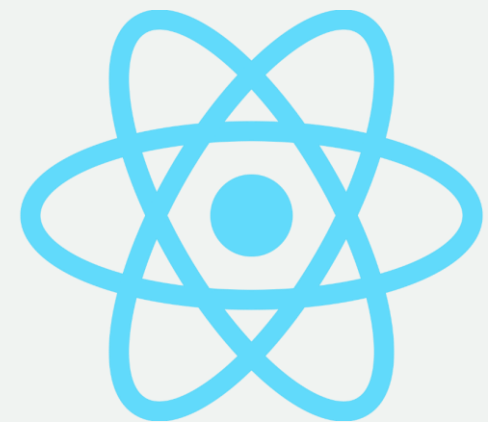
<https://create-react-app.dev/>

Übung Create-React-App

- Navigiere zum Ordner mit dem Befehl `cd „./0 Eure Codebeispiele/[Dein Name]/React“`
- Erstelle ein Projekt mit dem Befehl `„npx create-react-app [Name deines Projekts]“` innerhalb des Ordners `„./0 Eure Codebeispiele/[Dein Name]/React“`
- Das Erstellen der React App kann einige Minuten dauern
- Nach erfolgreicher Installation führe den Befehl `„npm start“` aus
- Ein Server sollte hochfahren und eine React Startseite anzeigen
- Gemeinsam werden wir versuchen zu verstehen, wie das Projekt aufgebaut ist



UI Elemente



React Komponenten

- React-Anwendungen bestehen aus Komponenten.
- Eine Komponente ist ein Teil der Benutzeroberfläche (UI), der über seine eigene Logik und Darstellung verfügt.
- Eine Komponente kann so klein wie eine Schaltfläche oder so groß wie eine ganze Seite sein.
- Komponenten können ineinander verschachtelt werden
 - Achtung: Komponenten können andere Komponenten rendern, aber ihre Definitionen dürfen niemals verschachtelt sein.



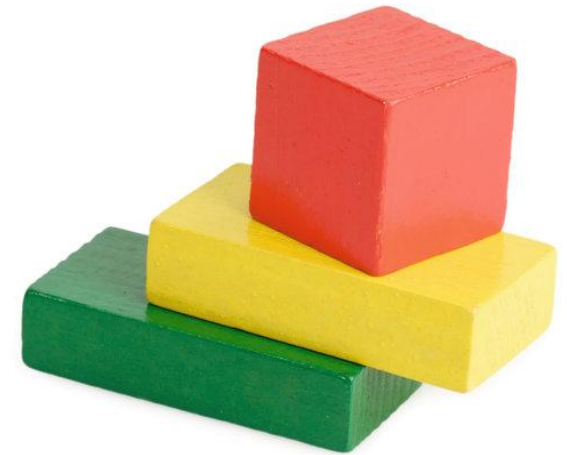
React Komponenten als Funktionen

- React Komponenten können durch Klassen oder durch Funktionen definiert werden, wobei Trend stark Richtung Funktionen geht
- React-Komponenten starten IMMER mit Großbuchstaben
- React-Komponenten sind JavaScript-Funktionen, die Markup (JSX siehe nächste Folie) zurückgeben.
- „export default“ ist eine Standard-JavaScript-Syntax (nicht spezifisch für React). Es ermöglicht, die Hauptfunktion in einer Datei zu markieren, damit sie später von anderen Dateien über „import“ importiert werden kann
- Bsp: import MyApp from „./MyApp.js“
- Tipp: Best practice ist eine Komponente je JS-Datei, wobei die Datei denselben Namen wie die Funktion der React Komponente hat

```
function MyButton() {  
  return (  
    <button>  
      I'm a button  
    </button>  
  );  
}  
  
export default function MyApp() {  
  return (  
    <div>  
      <h1>Welcome to my app</h1>  
      <MyButton />  
    </div>  
  );  
}
```

Übung zu React Komponenten

- Erstelle einen Ordner „components“ in deinem React Projekt
- Erstelle die Datei „MyParentComponent.js“ und „MyNestedComponent.js“ innerhalb des Ordners „components“
- MyNestedComponent.js beinhaltet eine gleichnamige Funktion die ein `<div>` Element zurückgibt welches den Text „Ich bin eine verschachtelte Komponente“ beinhaltet
- MyParentComponent.js beinhaltet eine gleichnamige Funktion die ein `<div>` Element mit dem Text „Ich bin die Parent Komponente“ und zusätzlich die Komponente „MyNestedComponent“ zurückgibt
- Bind die Komponente „MyParentComponent“ in App.js ein



JavaScript XML (JSX)

- JSX steht für JavaScript XML und ist eine Syntaxerweiterung für JavaScript.
- Es ermöglicht das Schreiben von HTML-ähnlichem Code innerhalb von JavaScript, was die Erstellung von Benutzeroberflächen in React erleichtert.
 - Um HTML in JSX zu verwandeln kann folgender Converter verwendet werden: <https://transform.tools/html-to-jsx>
- JSX wird von React verwendet, um virtuelle DOM-Elemente zu erstellen, die dann weiters in tatsächliches HTML umgewandelt und im Browser gerendert werden



Regeln für JSX

1. Gib ein einziges Parent Element zurück

- Um mehrere Elemente aus einer Komponente zurückzugeben, umschließe sie mit einem einzigen Elternelement, zum Beispiel mit einem `<div>`
- Wenn man kein zusätzliches `<div>` im HTML hinzufügen möchten, kann stattdessen ein leeres Fragment `<></>` als Parent Element verwendet werden

2. Schließe alle Tags

- JSX erfordert, dass Tags explizit geschlossen werden: Selbstschließende Tags wie `` oder `` müssen explizit mit `` oder `` geschlossen werden

3. Verwende meistens camelCase!

- JavaScript hat Einschränkungen bei Variablennamen. Zum Beispiel dürfen ihre Namen keine Bindestriche enthalten oder reservierte Wörter wie "class" sein.
- Deshalb werden in React viele HTML- und SVG-Attribute in camelCase geschrieben.
- Wichtiges Beispiel:
"class" wird in JSX zu "className",

```
export default function TodoList() {
  return (
    <>
      <h1>Hedy Lamarr's Todos</h1>
      
      <ul>
        <li>Invent new traffic lights</li>
        <li>Rehearse a movie scene</li>
        <li>Improve the spectrum technology</li>
      </ul>
    </>
  )
}
```

Quiz zu JSX: Wo ist der Fehler?

```
export default function Bio() {  
  return (  
    <div class="intro">  
      <h1>Welcome to my website!</h1>  
    </div>  
    <p class="summary">  
      You can find my thoughts here.  
      <br><br>  
      <b>And <i>pictures</b></i> of scientists!  
    </p>  
  );  
}
```

JavaScript in JSX verwenden

- Das Einbetten von JavaScript-Ausdrücken innerhalb von JSX wird ermöglicht, indem der Code in geschweifte Klammern {} eingeschlossen wird.
- So können auch zuvor definierte Variablen, Funktionen und Ausdrücke in JSX verwendet werden, um dynamische Inhalte zu rendern.
- Man kann geschweifte Klammern in JSX nur auf zwei Arten verwenden:
 - Als Text direkt innerhalb eines JSX-Tags:
`<h1>{name}'s To Do List</h1>` funktioniert, aber
`<{tag}>Gregorio Y. Zara's To Do List</{tag}>` funktioniert nicht.
 - Als Attribute, die unmittelbar auf das =-Zeichen folgen:
`src={avatar}` liest die Variable avatar, aber `src="{avatar}"` übergibt den String "{avatar}".

```
export function Greeting() {  
  const name = 'John';  
  const greetingMessage = `Hello, ${name}`;  
  
  return (  
    <div>  
      <h1>Greeting</h1>  
      <p>{greetingMessage}</p>  
    </div>  
  );  
}
```


Styling in JSX

- Um externe Stylesheets einzubinden wird das CSS File importiert:
Beispiel: `import "../pathToStyle/style.css"`
- Um einem JSX Element eine Klasse des CSS zu geben wird das Attribut `className` verwendet
- Inline Styling ist auch in JSX möglich. Dafür verwendet man doppelte geschwungene Klammern `{[..]}` innerhalb derer der Style angegeben wird
- Bei Inline Styling werden CSS-Attribute ohne Bindestrich, stattdessen mit CamelCase geschrieben
Bsp: `background-color` → `backgroundColor`

```
import "../style.css"

function MyButton() {
  return (
    <>
      <button className="button">
        I'm a button
      </button>
      <div style={
        {
          backgroundColor: 'black',
          color: 'pink'
        }
      }>Test
    </div>
    </>
  );
}
```

React props in Parentkomponente

- React-Komponenten verwenden Props, um miteinander zu kommunizieren. Jede Elternkomponente kann ihren Kindkomponenten Informationen über Props übergeben.
- Mittels Props können jegliche JavaScript-Werte übergeben werden, einschließlich Variablen, Objekten, Arrays und Funktionen.
- Objekte werden durch Doppelte geschwungene Klammern übergeben
- Props werden über JSX-Tags an Kindkomponenten übergeben

```
export default function Profile() {  
  return (  
    <Avatar  
      person={{ name: 'Lin Lanying', imageId: '1bX5QH6' }}  
      size={100}  
    />  
  );  
}
```

React props in Kindkomponente

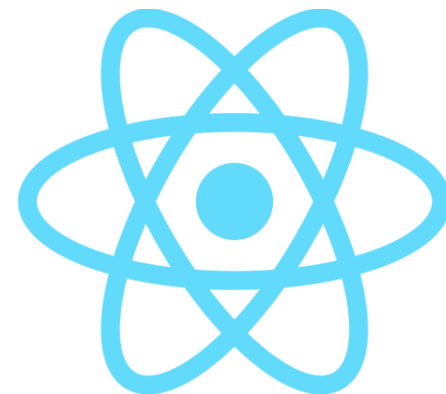
- Du kannst diese Props lesen, indem du ihre Namen (z.B. person, size) durch Kommas getrennt innerhalb von ({ und }) direkt nach der Funktion Avatar auflistest.
- Dies ermöglicht es dir, sie innerhalb der Komponente zu verwenden, ähnlich wie du es mit einer Variablen tun würdest.
- Es ist auch möglich alle übergebenen Werte in der Variable "props" zu übergeben ohne alle Variablen einzeln zu listen
- Um eine spezielle Variable der Props anzusprechen, wird props.[Variablenname] verwendet

```
function Avatar({ person, size }) {  
  return (  
    <img  
      className="avatar"  
      src={getImageUrl(person)}  
      alt={person.name}  
      width={size}  
      height={size}  
    />  
  );  
}
```

```
function Avatar(props) {  
  return (  
    <img  
      className="avatar"  
      src={getImageUrl(props.person)}  
      alt={props.person.name}  
      width={props.size}  
      height={props.size}  
    />  
  );  
}
```

Übung zu React & JSX

- Definiere eine Variable `let color = [beliebige Farbe]` innerhalb von `MyParentComponent`
- Übergib die Farbe als prop an die `MyNestedComponent`
- In der „`MyNestedComponent`“ wird die übergebene Farbe verwendet um die Textfarbe zu definieren. Verwende dafür `Inline Styling`
- Zusatzaufgabe (etwas schwerer):
 - Definiere eine zusätzliche Variable `let componentVisibility = true/false` innerhalb von `MyParentComponent` und übergib diesen Wert als prop an `MyNestedComponent`
 - Implementiere Logik in `MyNestedComponent` wodurch die Komponente nur gerendert wird wenn `componentVisibility === true`
Achtung: Dafür darf kein Styling verwendet werden bsp: `display` oder `visibility`



Bilder einfügen

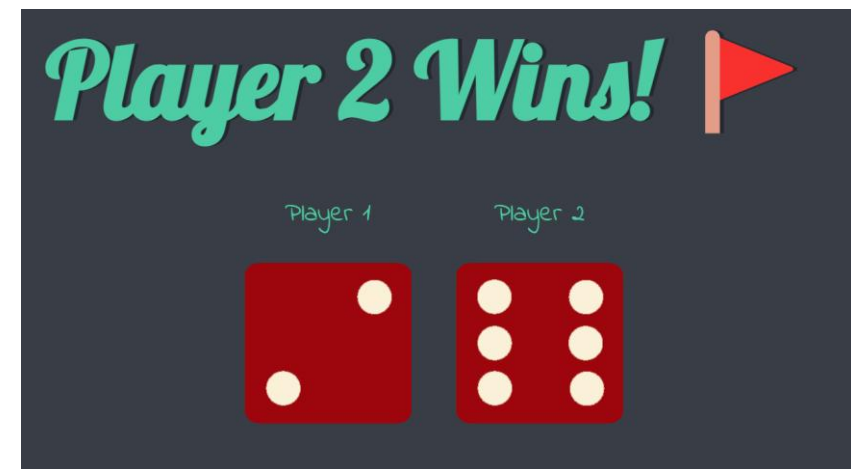
- Um Bilder in React einzubinden muss es zunächst importiert werden
 - Bsp: `import image from „./images/image.png“`
- Danach wird die importierte Variable als src eines img Tags eingebunden
 - Bsp: ``

```
import React from "react";
import "./styles.css";
import image from "./images/image.png"

export default function ImageExample() {
  return (
    <>
      <img className="img1" src={image} />
    </>
  );
}
```

Übung zur Wiederholung „The Dice Game“

- Wir haben bereits einmal das „Dice Game“ programmiert
- Das Programm soll je nachdem wer die höhere Zahl gewürfelt hat den Sieger ernennen.
- Wenn die Seite neu geladen wird, werden erneut 2 Würfel gewürfelt
- Innerhalb der DiceGame Komponente sollen die Zufallszahlen der Würfel bestimmt werden und über props an die Dice Komponente übergeben werden
- Innerhalb der DiceGame Komponente soll der Gewinner des Spiels bestimmt werden und über props an die Header Komponente übergeben werden
- Die Komponenten sollten folgender Baumstruktur entsprechen



```

└─ DiceGame/
    └─ Header
    └─ Dice
  
```

Komponenten ineinander verschachteln

- Es ist üblich, eingebaute Browser-Tags zu verschachteln.
- Häufig möchten wir unsere eigenen React Komponenten ineinander verschachteln.
- Wenn Inhalte innerhalb eines JSX-Tags verschachtelt werden, wird die übergeordnete Komponente diese Inhalte in einer Prop namens children erhalten.
- Die children-Prop ermöglicht es der übergeordneten Komponente, den Inhalt innerhalb des Tags zu rendern oder auf ihn zuzugreifen.

```
import Avatar from './Avatar.js';

function Card({ children }) {
  return (
    <div className="card">
      {children}
    </div>
  );
}

export default function Profile() {
  return (
    <Card>
      <Avatar
        size={100}
        person={{
          name: 'Katsuko Saruhashi',
          imageId: 'Yfe0qp2'
        }}
      />
    </Card>
  );
}
```

Conditional Rendering

- In React können wir JSX bedingt rendern, indem wir:
- Innerhalb von JavaScript (außerhalb vom return Statement):
 - JavaScript-Syntax wie if-Anweisungen
 - Ternäre Operatoren (&& und ? :) verwenden
- Innerhalb vom JSX Code:
 - Ternäre Operatoren (&& und ? :)
 - If-Anweisungen können **nicht** innerhalb von JSX Code verwendet werden

```
if (isPacked) {
  return <li className="item">{name} ✓</li>;
}
return <li className="item">{name}</li>;
```

If Anweisungen

```
return (
  <li className="item">
    {isPacked && name + ' ✓'}
  </li>
);
```

```
return (
  <li className="item">
    {isPacked ? name + ' ✓' : name}
  </li>
);
```

Ternäre Operatoren && / ?:

Ternäre Operatoren in Beispielen:

&&: Wenn Bedingung erfüllt ist, (&&) dann rendere name + „ ✓ “. Sonst rendere nichts (null)

? : Wenn Bedingung erfüllt ist, (?) dann rendere name + „ ✓ “. (:) sonst rendere name

Rendering von Listen

- Oft möchten wir die selbe Komponenten mit unterschiedlichen Daten, aus einem Array oder einem Objekt stammend, anzeigen.
- Dafür können wir die JavaScript-Methoden verwenden, um ein Array oder ein Object von Daten zu mappen
- `map()` wird verwendet, um ein Datenarray in ein Array von Komponenten zu transformieren.

```
const people = [  
  'Creola Katherine Johnson: mathematician',  
  'Mario José Molina-Pasquel Henríquez: chemist',  
  'Mohammad Abdus Salam: physicist',  
  'Percy Lavon Julian: chemist',  
  'Subrahmanyan Chandrasekhar: astrophysicist'  
];  
  
export default function List() {  
  const listItems = people.map(person =>  
    <li>{person}</li>  
  );  
  return <ul>{listItems}</ul>;  
}
```

Rendering von gefilterten Listen

- Häufig kommt es vor das zunächst ein Array, bestehend aus Objekten, gefiltert werden muss bevor es auf ein JSX Element gemapped wird
- Um Arrays zu filtern wird die filter Methode verwendet
- Danach wird das gefilterte Array mit Hilfe von map() erneut auf JSX Elemente gemapped
- Achtung: Wird ein Array aus Objekten gemapped, müssen natürlich die Properties des Objekts angesprochen werden und nicht das gesamte Objekt

```
export default function List() {
  const people = [{
    id: 0,
    name: 'Creola Katherine Johnson',
    profession: 'mathematician',
  }, {
    id: 1,
    name: 'Mario José Molina-Pasque',
    profession: 'chemist',
  }, {
    id: 2,
    name: 'Mohammad Abdus Salam',
    profession: 'physicist',
  }, {
    name: 'Percy Lavon Julian',
    profession: 'chemist',
  }];

  const chemists = people.filter(person =>
    person.profession === 'chemist'
  );

  const listItems = chemists.map(person =>
    <li>
      <img
        src={getImageUrl(person)}
        alt={person.name}
      />
      <p>
        <b>{person.name}</b>
        { ' ' + person.profession + ' ' }
        known for {person.accomplishment}
      </p>
    </li>
  );
  return <ul>{listItems}</ul>;
}
```

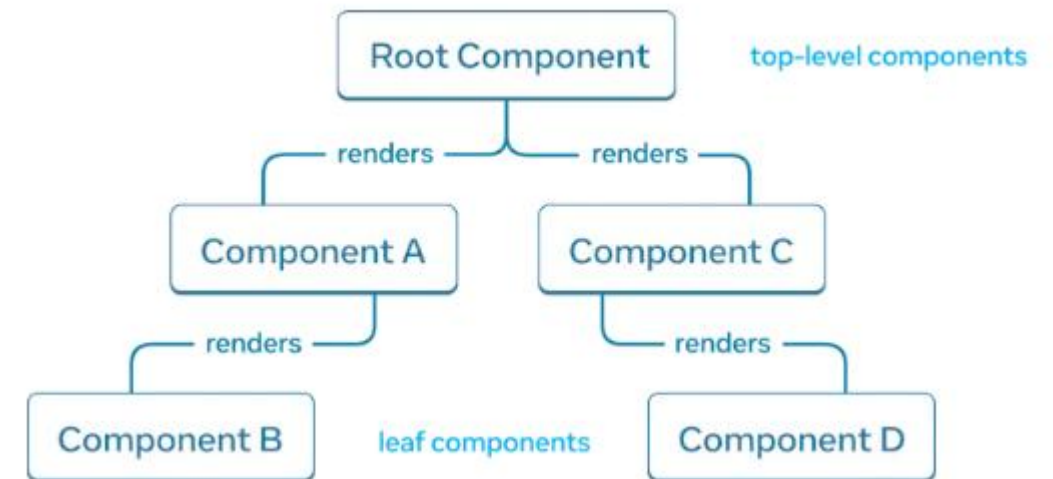
Keys für gemappte Elemente

- Jedes mit `map()` erstellte JSX Element benötigt einen eindeutigen Key
- Schlüssel sagen React, welchem Array-Element jede Komponente entspricht, damit es sie später zuordnen kann.
- Dies wird wichtig, wenn sich Array-Elemente bewegen können (z. B. durch Sortieren), eingefügt oder gelöscht werden.
- Schlüssel helfen React zu erkennen, was genau passiert ist, und die richtigen Aktualisierungen am DOM-Baum vorzunehmen.
- Anstatt Schlüssel dynamisch zu generieren, sollten Schlüssel aus den Objektdaten kommen

```
const listItems = chemists.map(person =>  
  <li key={person.id}>  
    <img  
      src={getImageUrl(person)}  
      alt={person.name}  
    />  
  </li>  
)
```

UI Baumstruktur

- React verwendet Bäume, um die Beziehungen zwischen Komponenten und Modulen zu modellieren.
- Ein React-Renderbaum ist eine Darstellung der Eltern- und Kindbeziehung zwischen Komponenten.
- Der React-Renderbaum ermöglicht es React, die Hierarchie der Komponenten zu verfolgen und zu organisieren.
- Durch die Baumstruktur kann React effizient Updates durchführen und die Komponenten entsprechend ihrer Beziehungen rendern.

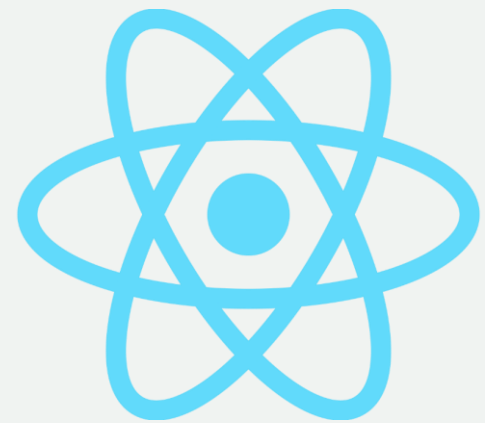


React Challenges

- Challenges zu JSX Syntax
<https://react.dev/learn/javascript-in-jsx-with-curly-braces#challenges>
- Challenges zu Properties (props)
<https://react.dev/learn/passing-props-to-a-component#challenges>
- Challenge zu Conditional Rendering (3 optional)
<https://react.dev/learn/conditional-rendering#challenges>
- Challenge zu List Rendering (3 & 4 optional)
<https://react.dev/learn/rendering-lists#challenges>



Events & States



React Events

- React ermöglicht es, Eventhandler in JSX hinzuzufügen.
- Eventhandler sind eigenen Funktionen, die in Reaktion auf Interaktionen wie Klicken, Hovern, Fokussieren usw. ausgelöst werden.
- Um einen Eventhandler hinzuzufügen, definieren wir zunächst eine Funktion und übergeben diese dann als onClick-Prop an das entsprechende JSX-Tag.
- Eventhandler :
 - Werden in der Regel innerhalb Ihrer Komponenten definiert.
 - Haben Namen, die mit „handle“ beginnen, gefolgt vom Namen des Ereignisses

```
export default function Button() {  
  function handleClick() {  
    alert('You clicked me!');  
  }  
  
  return (  
    <button onClick={handleClick}>  
      Click me  
    </button>  
  );  
}
```

stopPropagation & preventDefault

- Eventhandler erhalten ein Ereignisobjekt als ihr einziges Argument. Üblicherweise wird es als "e" oder "event" bezeichnet, was für "Event" steht.
- Mit diesem Ereignisobjekt können wir die Event-Propagation stoppen.
 - Wenn wir verhindern möchten, dass ein Ereignis Elternkomponenten erreicht, müssen wir **e.stopPropagation()** aufrufen
- Einige Browser-Ereignisse haben standardmäßiges Verhalten, das mit ihnen verbunden ist.
 - Zum Beispiel wird ein `<form>` Submit-Ereignis, standardmäßig die ganze Seite neu laden
 - Um zu verhindern dass dieses Standardverhalten auftritt wird der Befehl **e.preventDefault()** aufgerufen

```
function Button({ onClick, children }) {  
  return (  
    <button onClick={e => {  
      e.stopPropagation();  
      onClick();  
    }}>  
      {children}  
    </button>  
  );  
}
```

```
export default function Signup() {  
  return (  
    <form onSubmit={e => {  
      e.preventDefault();  
      alert('Submitting!');  
    }}>  
      <input />  
      <button>Send</button>  
    </form>  
  );  
}
```

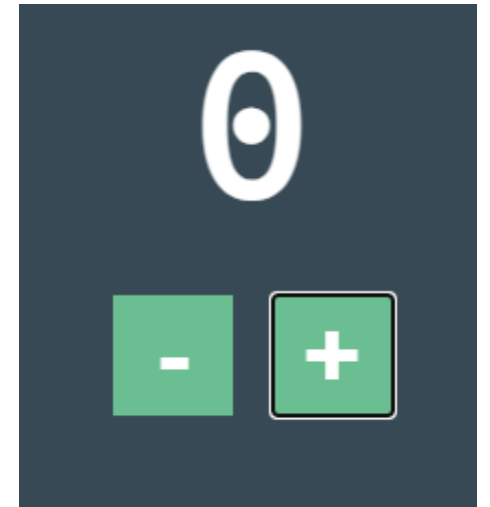

Kurze Übung: CounterApp

- Baue eine React Komponente mit der Anzeige einer Zahl und 2 Buttons (+, -)
- Definiere eine Variable counter = 0
- Schritt 1: Beim Klick auf + oder - wird in der Konsole der Wert von counter +1 / -1 ausgegeben.
- Schritt 2: Der Wert der angezeigten Zahl wird ebenfalls aktualisiert und angezeigt.
- Was fällt dir auf?



Problem: Änderungen von Variablen

- Lokale Variablen bestehen zwischen den Renders nicht fort. Wenn React diese Komponente ein zweites Mal rendert, rendert es sie von Grund auf neu - es berücksichtigt keine Änderungen an den lokalen Variablen.
- Änderungen an lokalen Variablen lösen keine Renders aus. React erkennt nicht, dass es die Komponente erneut mit den neuen Daten rendern muss.
- Lösung: React States



React States - Zustand Variablen (1)

- Vorteil von State-Variablen:
 - Eine State-Variable, speichert Daten zwischen erneuten Renders.
 - Liefert eine Setter-Funktion, um die Variable zu aktualisieren
 - Jedes Mal wenn sich die Variable ändert wird die Komponente neu gerendert und Änderungen werden sofort sichtbar
- Um eine State-Variable verwenden zu können muss das Package `useState` von React importiert werden
- Eine State-Variable wird gemeinsam mit ihrer Setter-Funktion in einer `const` Variable gespeichert und über die `useState([Defaultwert])` Methode initialisiert (initialer Wert wird vergeben)

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

export default function Counter() {
  const [count, setCount] = useState(0)

  function handleClick(){
    setCount(count+1)
  }
  return (
    <>
      <h1>{count}</h1>
      <button onClick={handleClick}>+</button>
    </>
  );
}
```

React States - Zustand Variablen (2)

- Eine State-Variable kann nur auf der obersten Ebene Ihrer Komponenten deklariert werden. Sie können nicht innerhalb von Bedingungen, Schleifen oder anderen verschachtelten Funktionen deklariert werden.
- Eine State-Variable wird genau wie jede andere Variable verwendet, allerdings werden Änderungen nur über die Settermethode veranlasst.
Bsp: ~~count = 2~~ sondern `setCount(2)`
- Nach Konvention wird die Settermethode nach dem Schema „set“+Variablenname (großer Anfangsbuchstabe) benannt

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

export default function Counter() {
  const [count, setCount] = useState(0)

  function handleClick(){
    setCount(count+1)
  }
  return (
    <>
      <h1>{count}</h1>
      <button onClick={handleClick}>+</button>
    </>
  );
}
```

Übung „The Dice Game“

- Erweitere deine „DiceGame“ Komponente um einen Button „NewGameButton“ der das Spiel erneut startet und die Würfel neu würfelt.
- Verwende eine 2 State-Variablen um die Augenzahlen der Würfel zu bestimmen
- Verwende eine State-Variable um den Gewinner zu definieren



```

└─ DiceGame/
    ├── Header
    ├── Dice
    └─ NewGameButton
  
```

Pro-Übung zu React „Lottozahlen Vorhersage“

- Schreibe eine React Komponente „LottoGenerator“ um die nächsten Lotto Zahlen vorherzusagen und damit reich zu werden
- Die darunterliegenden Komponenten sind in folgender Struktur aufgeteilt (rechts unten)
- Da es verschiedene Lottoformate gibt und wir für alle Formate die richtigen Zahlen wissen wollen müssen wir verschiedene Limits festlegen können (Min, Max) welche Werte die Zahlen annehmen können.
- Optional: bei manchen Lottoformaten werden mehr oder weniger Zahlen gezogen deshalb wäre eine zusätzliche Einstellung „Anzahl gezogener Zahlen“ auch gut.

Lotto Zahlen Vorhersage


Diese React App ist im Stande die kommenden Lottozahlen vorherzusagen. Generiere deine Zahlen und werde reich

Result

3 9 12 27 35 42

Unteres Limit

Oberes Limit

Generate 

Clear

```

└─ LottoGenerator/
   └─ Header
   └─ LottoNumbers
   └─ Settings/
      └─ MinValue
      └─ MaxValue
      └─ GenerateButton
      └─ ClearButton

```

React State - Zeitliches Verhalten

- Das Setzen des Zustands fordert einen neuen Render an.
- React speichert den Zustand außerhalb der Komponente
- Wenn Sie useState aufrufen, gibt Ihnen React eine Momentaufnahme des Zustands für diesen Render.
- Jeder Render (und Funktionen darin) wird immer die Momentaufnahme des Zustands "sehen", die React diesem Render gegeben hat.
- Die Auswirkungen der Änderung einer State-Variable sind erst beim nächsten Rendering der Komponente ersichtlich

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

export default function Counter() {
  const [number, setNumber] = useState(0);

  return (
    <>
      <h1>{number}</h1>
      <button onClick={() => {
        setNumber(number + 5);
        alert(number);
      }}>+5</button>
    </>
  )
}
```

React State - Zeitliches Verhalten (2)

- Was passiert wenn man auf den Button klickt?
- Da die Setterfunktion nur mit einer Momentaufnahme arbeitet, und den Zustand (Wert) aktualisiert, der bei der Ausführung der Funktion bestand, wird der Wert von number tatsächlich nur um 1 erhöht, unabhängig davon, wie oft der Button geklickt wird.

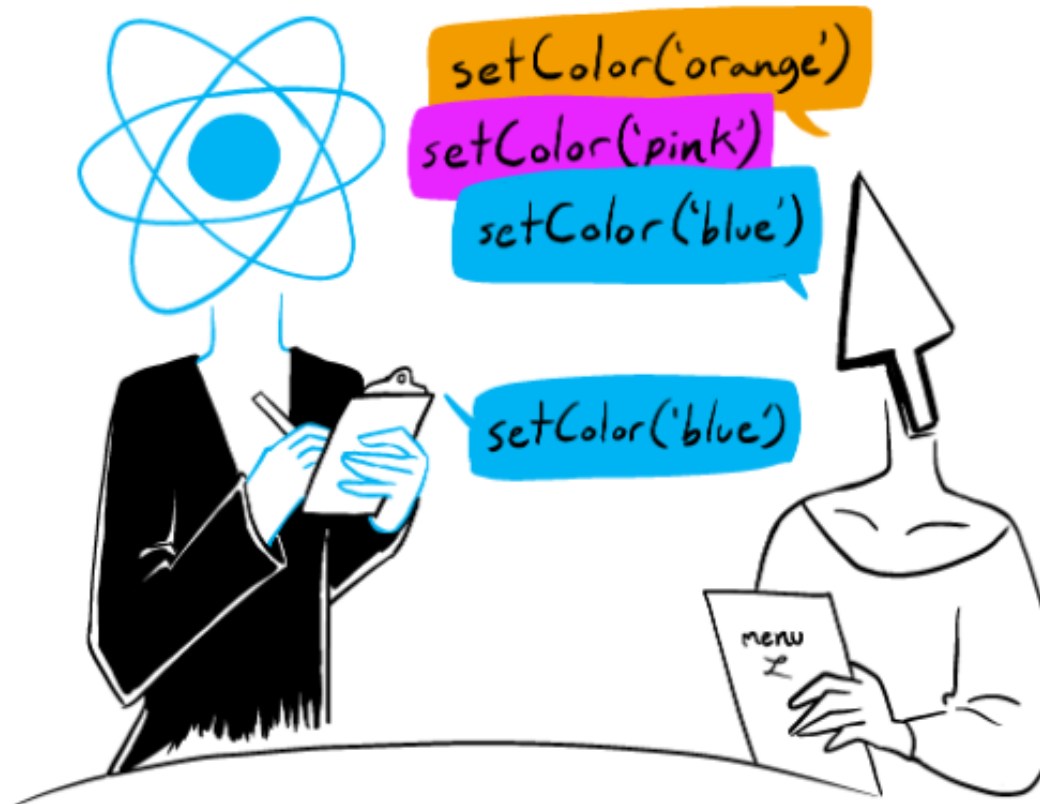
```
export default function Counter() {  
  const [number, setNumber] = useState(0);  
  
  return (  
    <>  
      <h1>{number}</h1>  
      <button onClick={() => {  
        setNumber(number + 1);  
        setNumber(number + 1);  
        setNumber(number + 1);  
      }}>+3</button>  
    </>  
  )  
}
```


React State - Zeitliches Verhalten (3)

- Was passiert wenn man auf den Button klickt?
- Trotz des Timeouts, wird keine zusätzliche Erhöhung um 5 gemacht weil die setTimeout Funktion mit der Momentaufnahme des Zustands arbeitet. Die setCount Methode wird also auch nach dem Timeout nur den alten Wert um 5 erhöhen

```
export default function Counter() {  
  const [number, setNumber] = useState(0);  
  
  return (  
    <>  
      <h1>{number}</h1>  
      <button  
        onClick={() => {  
          setCount(count + 5);  
          setTimeout(() => {  
            setCount(count + 5);  
          }, 3000);  
        }}  
      >  
        +5  
      </button>  
    </>  
  )  
}
```

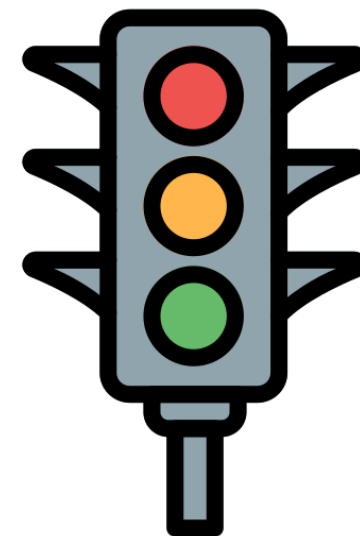
React State - Zeitliches Verhalten (4)



<https://react.dev/learn/queueing-a-series-of-state-updates>

Übung zu States & zeitliches Verhalten

- Hier ist ein Ampelkomponente, die beim Drücken des Buttons umschaltet
- Füge einen Alert zum Klick-Handler hinzu.
- Wenn die Ampel grün ist und "Gehen" anzeigt, sollte durch Klicken auf den Button "Stop kommt als nächstes" angezeigt werden.
- Wenn die Ampel rot ist und "Stop" anzeigt, sollte durch Klicken auf den Button "Gehen kommt als nächstes" angezeigt werden.
- <https://react.dev/learn/state-as-a-snapshot#challenges>



Mehrfache Änderung eines Zustands (updater functions)

- Das mehrfache Aktualisieren desselben Zustands vor dem nächsten Rendering kann in seltenen Fällen notwendig sein.
- Anstelle von `setNumber(number + 1)` kann eine Funktion übergeben werden, die den nächsten Zustand basierend auf dem vorherigen berechnet, wie z. B. `setNumber(n => n + 1)`.
- Hier wird eine sogenannte updater function verwendet um den zukünftigen Wert, basierend auf dem zuvor gesetzten Wert, zu berechnen

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

export default function Counter() {
  const [number, setNumber] = useState(0);

  return (
    <>
      <h1>{number}</h1>
      <button onClick={() => {
        setNumber(n => n + 1);
        setNumber(n => n + 1);
        setNumber(n => n + 1);
      }}>+3</button>
    </>
  )
}
```

Mehrfache Änderung eines Zustands

- Was passiert beim Klick auf den Button?
- Ausgabe: 6

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

export default function Counter() {
  const [number, setNumber] = useState(0);

  return (
    <>
      <h1>{number}</h1>
      <button onClick={() => {
        setNumber(number + 5);
        setNumber(n => n + 1);
      }}>Increase the number</button>
    </>
  )
}
```

Mehrfache Änderung eines Zustands

- Was passiert beim Klick auf den Button?
- Ausgabe: 5

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

export default function Counter() {
  const [number, setNumber] = useState(0);

  return (
    <>
      <h1>{number}</h1>
      <button onClick={() => {
        setNumber(n => n + 1);
        setNumber(number + 5);
      }}>Increase the number</button>
    </>
  )
}
```

Übung zu mehrfache Änderung eines Zustands

- Jedes Mal, wenn der Benutzer die Schaltfläche „Buy“ drückt, soll der „Pending“-Zähler um eins erhöht werden. Nach drei Sekunden soll der „Pending“-Zähler abnehmen und der „Completed“-Zähler erhöhen.
- Jedoch verhält sich der „Pending“-Zähler nicht wie beabsichtigt. Wenn du auf „Buy“ drückst, verringert er sich auf -1 (was nicht möglich sein sollte!). Und wenn du zweimal schnell hintereinander klickst, verhalten sich beide Zähler anscheinend unvorhersehbar.
- Warum passiert das? Behebe beide Zähler.
- <https://react.dev/learn/queueing-a-series-of-state-updates#challenges>

Pending: -1

Completed: 4

Buy

State-Objekte

- Der State kann jede Art von JavaScript-Wert halten, einschließlich Objekten.
- Es wird jedoch nicht empfohlen, Objekte, die sich im React-State befinden, direkt zu ändern. Dadurch wird kein neues Rendering ausgelöst und Änderungen bleiben für den Benutzer unsichtbar
- Stattdessen, wenn du ein Objekt aktualisieren möchtest, musst du ein neues Objekt erstellen (oder eine Kopie eines vorhandenen erstellen) und dem State das neue Objekt zuweisen
- Problem: Was machen wir wenn ein Objekt sehr viele Properties beinhaltet? Jedes Einzelne abtippen ist aufwändig ☹️

```
const [position, setPosition] = useState({
  x: 0,
  y: 0
});
```

State Definition mit Objekt als Wert

```
onPointerMove={e => {
  position.x = e.clientX;
  position.y = e.clientY;
}}
```

Direkte Änderung der Properties wird im UI nicht sichtbar

```
onPointerMove={e => {
  setPosition({
    x: e.clientX,
    y: e.clientY
  });
}}
```

Bessere Variante:
Über Setterfunktion neues Objekt zuweisen

Objekte duplizieren / Spread Operator

- Mit dem Spread-Operator können Objekte einfach geklont werden, ohne die ursprünglichen Objekte zu verändern.
- Durch Verwendung des Spread-Operators können neue Objekte erstellt werden, die Teile eines vorhandenen Objekts enthalten, sowie zusätzliche Eigenschaften hinzugefügt werden.

```
//ohne Spread Operator
setPerson({
  firstName: e.target.value,
  lastName: person.lastName,
  email: person.email
  //viele weitere Properties
});

//mit Spread Operator
setPerson({
  ...person,
  firstName: e.target.value
});
```

Formulare & Eingabefelder mit States

- Verwende React, um Formulare zu erzeugen und die Verwaltung von Zuständen zu ermöglichen
- Implementiere Steuerelemente für Eingabefelder, indem du das value-Attribut und das onChange-Ereignis verwendest.
- Validiere Benutzereingaben, indem du entsprechende Funktionen einfügst, um die Eingaben auf Korrektheit zu prüfen.
- Verwalte mehrere Eingabefeldwerte entweder durch separate Zustandsvariablen oder durch ein Objekt, das mehrere Werte enthält.
- Implementiere eine handleSubmit Funktion mit der das Absenden des Formulars bearbeitet wird. Beachte hier event.preventDefault() um ein Neuladen der Seite zu verhindern

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

export default function MyForm() {
  const [formData, setFormData] = useState({
    username: '',
    //hier können noch weitere Attribute stehen
  });

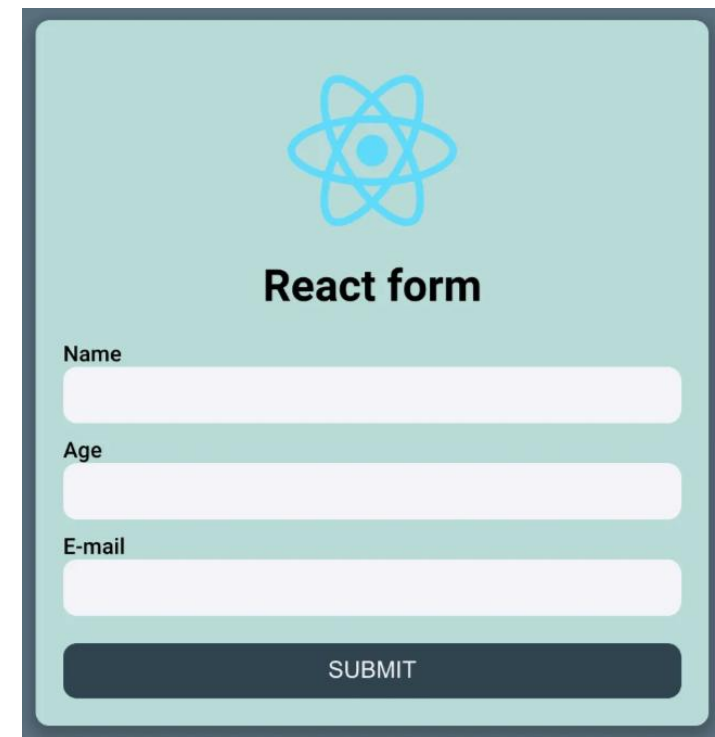
  const handleChange = (event) => {
    setFormData({ ...formData, name: event.target.value });
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    console.log(formData);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        name="username"
        value={formData.username}
        onChange={handleChange}
        placeholder="Benutzername"
      />
      <button type="submit">Absenden</button>
    </form>
  );
}
```

Übung zu Formulare & State-Objekt

- Erstelle eine React-Komponente namens UserInfoForm.
- Verwende den useState-Hook, um einen State formData zu initialisieren, der ein Objekt mit den Feldern name, email und age enthält.
- Erstelle eine Funktion handleChange, die die Benutzereingaben aktualisiert, wenn sich die Werte in den Eingabefeldern ändern.
- Erstelle eine Funktion handleSubmit, die aufgerufen wird, wenn der Benutzer das Formular absendet. In dieser Funktion wird der eingegebene Name mit einem alert-Fenster angezeigt.
- Baue das Formular mit den Eingabefeldern für Name, E-Mail und Alter sowie einem Submit-Button auf, der die handleSubmit-Funktion aufruft.



State Struktur Prinzipien

- Gruppiere zusammenhängende Zustände. Wenn sich immer zwei oder mehr Zustandsvariablen gleichzeitig aktualisieren, nutze Objekte um sie zu einer einzigen Zustandsvariablen zusammenzuführen.
- Vermeide stark verschachtelte Zustand. Stark hierarchischer Zustand ist nicht sehr praktisch zu aktualisieren. Wenn möglich, bevorzugen Sie eine flache Strukturierung des Zustands.

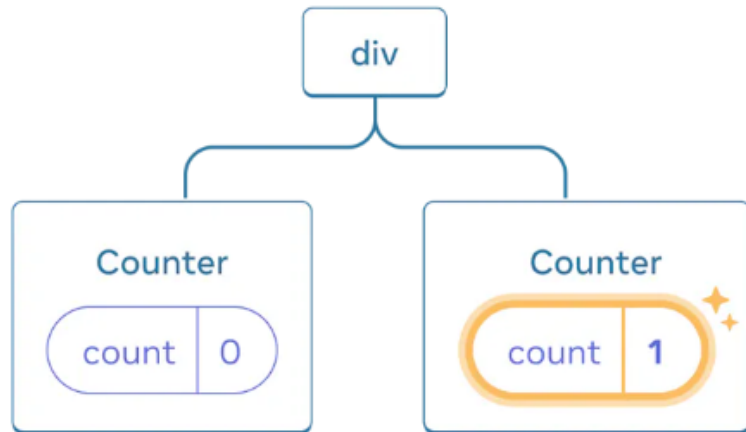
```
const [x, setX] = useState(0);  
const [y, setY] = useState(0);
```

```
const [position, setPosition] = useState({ x: 0, y: 0 });
```

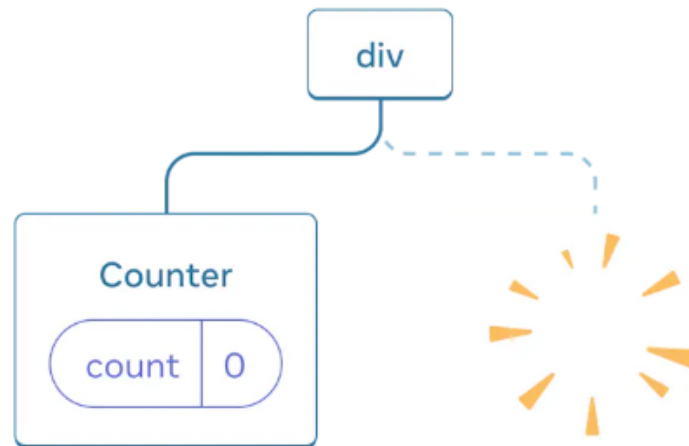
Zustand erhalten und zurücksetzen

- In React bleibt der Zustand erhalten, solange die gleiche Komponente an derselben Position gerendert wird.
- Die gleiche Komponente an derselben Position behält ihren Zustand bei.
- Unterschiedliche Komponenten an derselben Position setzen den Zustand zurück.
- Der Zustand wird nicht in JSX-Tags gespeichert. Er ist mit der Baumposition verbunden, in die du dieses JSX platzierst.
- Mehr Details dazu findest du hier:
<https://react.dev/learn/preserving-and-resetting-state#>

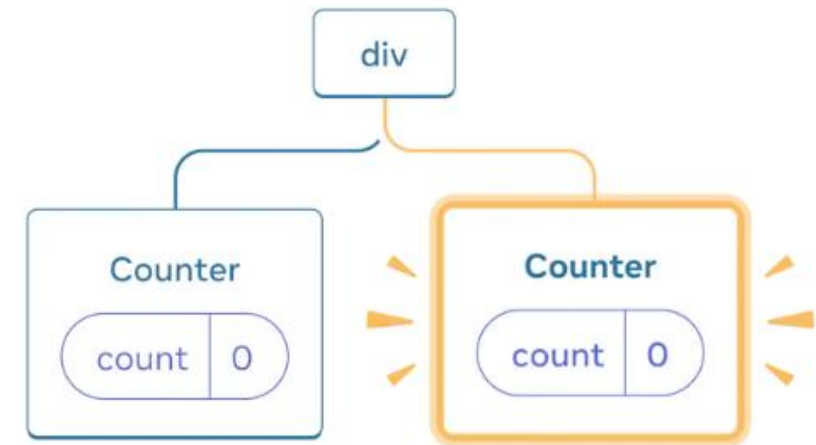
Zustand erhalten und zurücksetzen



Updating state



Deleting a component

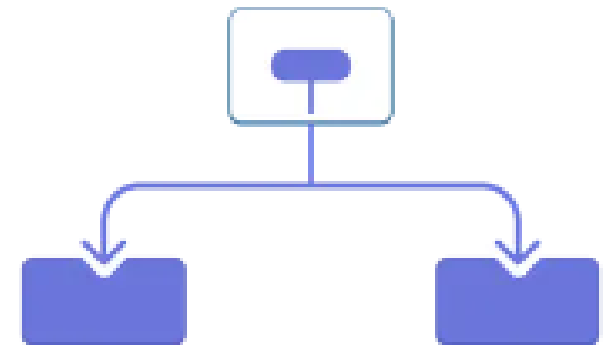


Adding a component

Lifting State Up

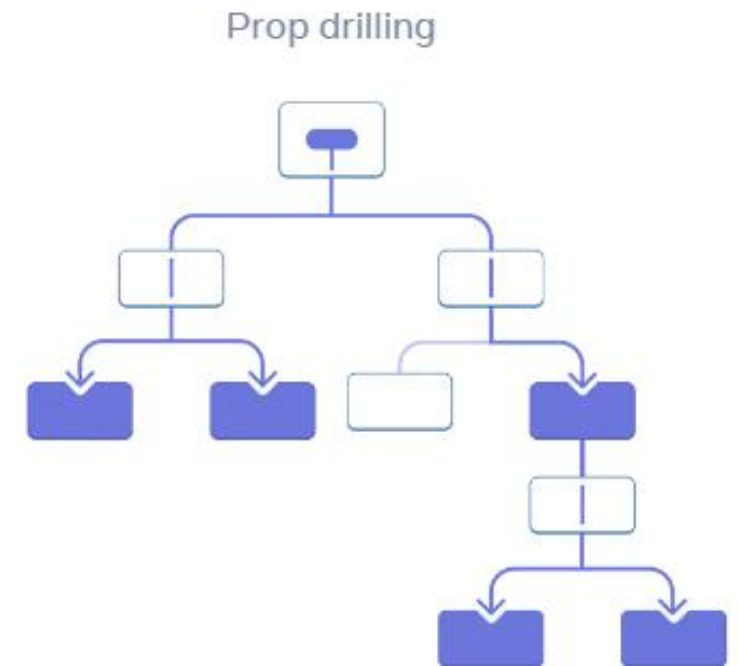
- Manchmal möchtest du, dass der Zustand von zwei Komponenten immer zusammen geändert wird.
- Um dies zu erreichen, entferne den Zustand aus beiden Komponenten.
- Verschiebe ihn zum nächsten gemeinsamen Elternelement.
- Übergebe ihn dann über Props an das Parent Element.
- Dies wird als Lift up des Zustands bezeichnet und ist eine häufig eingesetzte Methode

Lifting state up

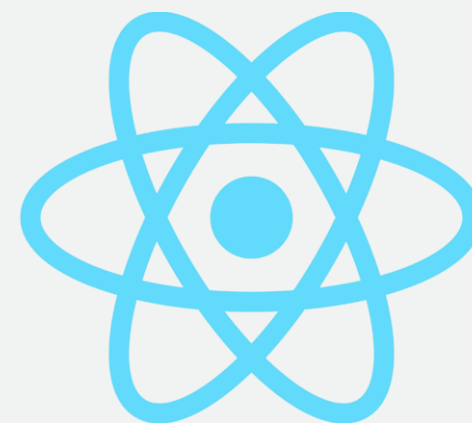


Problem mit Props

- Das Übergeben von Props ist eine großartige Möglichkeit, Daten explizit durch Ihren UI-Baum an die Komponenten weiterzuleiten, die sie verwenden.
- Das Übergeben von Props kann jedoch umständlich und unpraktisch werden, wenn ein Prop tief durch den Baum übergeben wird oder wenn viele Komponenten dieselbe Prop benötigen.
- Der nächstgelegene gemeinsame Vorfahre kann weit von den Komponenten entfernt sein, die Daten benötigen, und das Hochziehen des Zustands bis dorthin kann zu einer Situation führen, die als "Prop-Drilling" bezeichnet wird.
- Lösung: Context



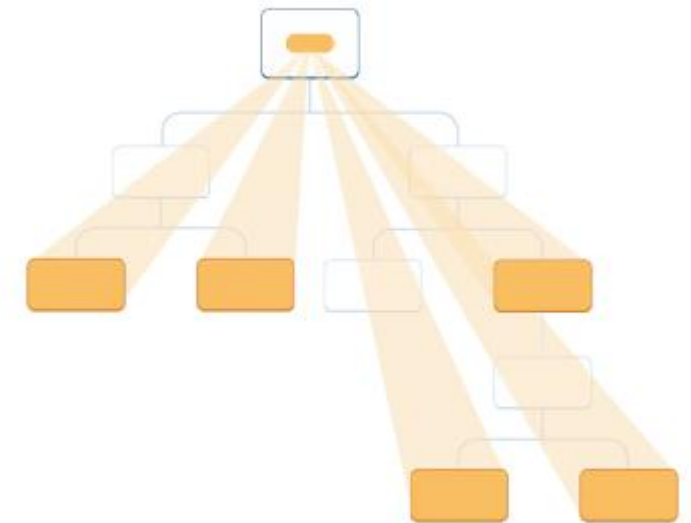
Context & Hooks



React Context

- Normalerweise werden Informationen von einer Elternkomponente an eine Kindkomponente über Props übergeben.
- Das Weitergeben von Props kann umständlich und unpraktisch werden, wenn sie durch viele Zwischenkomponenten hindurchgereicht werden müssen oder wenn viele Komponenten dieselben Informationen benötigen.
- Der Kontext ermöglicht es der Elternkomponente, bestimmte Informationen allen Komponenten im darunterliegenden Baum - unabhängig von der Tiefe - ohne explizites Weitergeben durch Props zur Verfügung zu stellen.

Using context in distant children



React Context

```
<Section>
  <Heading level={4}>Sub-sub-heading</Heading>
  <Heading level={4}>Sub-sub-heading</Heading>
  <Heading level={4}>Sub-sub-heading</Heading>
</Section>
```

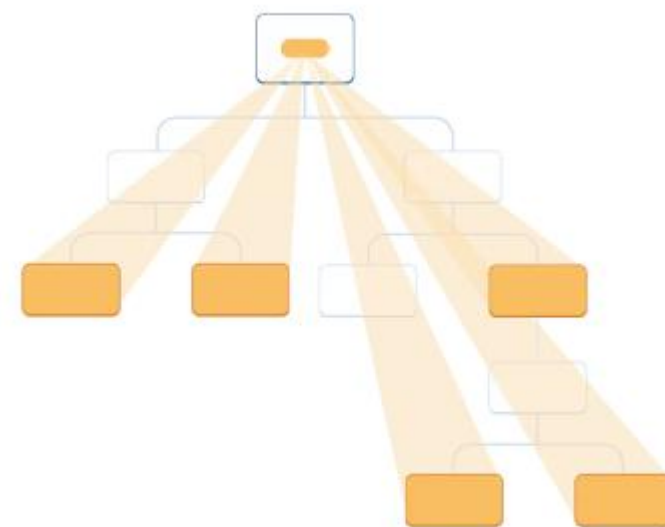


```
<Section level={4}>
  <Heading>Sub-sub-heading</Heading>
  <Heading>Sub-sub-heading</Heading>
  <Heading>Sub-sub-heading</Heading>
</Section>
```

React Context Vorgehen

1. Einen Kontext erstellen (LevelContext genannt), um eine Umgebung zu definieren.
 2. Den erstellten Kontext verwenden, wo die Komponente die Daten benötigt
 3. Den Kontext von der Komponente bereitstellen, die die Daten spezifiziert
- Ein Kontext ermöglicht es einem Elternelement - selbst einem entfernten! - Daten an den gesamten darin enthaltenen Baum bereitzustellen.

Using context in distant children



React Context Erstellen

- Zuerst muss der Kontext erstellt werden.
- Dieser muss aus einer Datei exportiert werden, damit die Komponenten ihn verwenden können.
- Das einzige Argument für createContext ist der Standardwert.
- Es könnte jeder beliebige Wert übergeben werden (auch Objekte).

```
import { createContext } from 'react';  
export const LevelContext = createContext(1);
```

React Context Bereitstellen (Provider)

- Bevor der Kontext verwendet wird muss dieser noch von einer Parent Komponente (Provider) bereitgestellt werden
- Dafür wird der Kontext vom zuvor erstellten File importiert
- Um den Kontext an Unterkomponenten (children) weiterzugeben verwende folgende Syntax:

```
import { LevelContext } from './LevelContext.js';

export default function Section({ level, children }) {
  return (
    <section className="section">
      <LevelContext.Provider value={level}>
        {children}
      </LevelContext.Provider>
    </section>
  );
}
```

```
<NameDesKontext.Provider value={wertDerAnChildrenKomponentenÜbergebenWird}>{children}</NameDesKontext.Provider>
```

React Context Verwenden

- Importiere den useContext-Hook aus React sowie deinen Kontext
- Lese den Wert aus dem importierten Kontext LevelContext
- Dadurch werden keine Props sondern ein bereitgestellter Kontext in der Komponente verwendet
- Die Komponente verwendet den Wert des nächstgelegenen <LevelContext.Provider> im UI-Baum über ihr.

```
import { useContext } from 'react';
import { LevelContext } from '../LevelContext.js';

export default function Heading({ children }) {
  const level = useContext(LevelContext);
  // ...
}
```

React Context Usecases

- Theming: Ein Kontext für das Erscheinungsbild ermöglicht es Komponenten, sich dem visuellen Look anzupassen.
- Aktueller Benutzeraccount: Ein Kontext für den angemeldeten Benutzer vereinfacht den Zugriff darauf in der gesamten App.
- Allgemein: wenn viele Komponenten in unterschiedlichen Schichten dieselben Daten benötigen, bietet sich ein Kontext an.
- Achte dennoch darauf Kontext nicht zu ausgiebig zu verwenden, führt häufig zu komplexerem und schwer verständlichem Code → Datenfluss wird verschleiert

React Context Challenge

- In diesem Beispiel ändert das Umschalten des Kontrollkästchens die imageSize-Prop, die an jedes `<PlacelImage>` übergeben wird. Der Zustand des Kontrollkästchens wird im Top-Level-App-Komponenten gehalten, aber jedes `<PlacelImage>` muss davon Kenntnis haben.
- Aktuell übergibt App imageSize eine List, die es an PlacelImage weitergibt. Entfernen Sie die imageSize-Prop und übergeben Sie sie stattdessen direkt von der App-Komponente an PlacelImage.
- Sie können den Kontext in Context.js deklarieren.
- <https://react.dev/learn/passing-data-deeply-with-context#challenges>

Aufbau bisher bekannter React Komponenten

- **Rendering code:** Dieser Code befindet sich auf der obersten Ebene Ihrer Komponente und ist dafür verantwortlich, die Props und den Zustand zu verarbeiten, zu transformieren und das JSX zurückzugeben, das auf dem Bildschirm angezeigt werden soll.
- **Event-Handler:** Event-Handler sind verschachtelte Funktionen innerhalb Ihrer Komponenten, die Aktionen ausführen, anstatt sie nur zu berechnen. Event-Handler enthalten "Seiteneffekte" (sie ändern den Zustand des Programms), die durch eine bestimmte Benutzeraktion verursacht werden (zum Beispiel ein Klick auf eine Schaltfläche oder das

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

export default function MyForm() {
  const [formData, setFormData] = useState({
    username: '',
    //hier können noch weitere Attribute stehen
  });

  const handleChange = (event) => {
    setFormData({ ...formData, name: event.target.value });
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    console.log(formData);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        name="username"
        value={formData.username}
        onChange={handleChange}
        placeholder="Benutzername"
      />
      <button type="submit">Absenden</button>
    </form>
  );
}
```

Wie löse ich Events aus wenn sich der Zustand der Komponente ändert?

- Beispiel: eine Anwendung, die eine Liste von Items anzeigt und es dem Benutzer ermöglicht, die Items anhand verschiedener Filterkriterien zu sortieren. Wenn der Benutzer bestimmte Filter auswählt, soll die Liste der Benutzer entsprechend aktualisiert werden.
- Beispiel: eine ChatRoom-Komponente, die sich mit dem Chat-Server verbinden muss, wenn sie auf dem Bildschirm sichtbar ist. Es gibt jedoch kein einzelnes bestimmtes Ereignis wie einen Klick, das dazu führt, dass der ChatRoom angezeigt wird.
- Beispiel: Wenn sich der Inhalt eines Warenkorbs ändert, möchten wir bestimmte Aktionen ausführen, wie zum Beispiel die Aktualisierung des Gesamtpreises oder das Speichern des Warenkorbs im lokalen Speicher.



React useEffect

- **useEffect** ermöglicht es, Nebeneffekte in Funktionskomponenten einzubauen.
- Es wird **nach** dem Rendern einer Komponente ausgeführt und bietet eine Möglichkeit, Code auszuführen, der auf Zustandsänderungen, Props oder Lebenszyklusevents reagiert.
- Mit useEffect können wir spezielle Operationen durchführen, wie Daten laden, Abonnements einrichten oder die DOM-Manipulation durchführen.
- Es akzeptiert eine Funktion als erstes Argument, die den Nebeneffekt beschreibt, sowie ein optionales Array von Abhängigkeiten als zweites Argument, um zu steuern, wann der Nebeneffekt ausgeführt wird.

```
import { useEffect } from 'react';

export default function MyComponent() {

  useEffect(() => {
    //Tue etwas nachdem die Komponente gerendert wurde
    console.log("die Komponente wurde erfolgreich gerendert");
  });

  return <div>
    |   Meine Komponente
  </div>;
}
```

React useEffect Vorgehen

1. Importiere useEffect aus dem React-Paket in deiner Datei
2. Deklariere useEffect in deiner Funktionskomponente
3. Innerhalb von useEffect gibst du eine Funktion an, die den Effekt definiert, den du ausführen möchtest
4. Optional kannst du ein zweites Argument in Form eines Arrays angeben, um anzugeben, auf welche Abhängigkeiten der Effekt reagieren soll

```
import { useEffect } from 'react';

export default function MyComponent() {

  useEffect(() => {
    //Tue etwas nachdem die Komponente gerendert wurde
    console.log("die Komponente wurde erfolgreich gerendert")
  });

  return <div>
    |   Meine Komponente
    </div>;
}
```

React useEffect Abhängigkeiten

- Abhängigkeiten eines useEffects werden in Form eines optionalen Arrays, als zweiten Parameter der useEffect Funktion übergeben
- Abhängigkeiten definieren noch genauer wann der Code innerhalb von useEffect ausgeführt werden soll
- Hier gibt es 3 Möglichkeiten:
 - Ohne Angabe von Abhängigkeiten (kein Array)
 - Mit Angabe eines leeren Arrays
 - Mit Angabe eines Arrays, welches meherer Zustände beinhaltet

```
import { useEffect } from 'react';

export default function MyComponent() {

  const [myState, setMyState] = useState("")

  useEffect(() => {
    //Tue etwas nachdem die Komponente gerendert wurde
    console.log("der Zustand wurde erfolgreich verändert")
  },[myState]);

  return <div>
    |   Meine Komponente
    </div>;
}
```

```
import { useEffect } from 'react';

export default function MyComponent() {

  useEffect(() => {
    //Tue etwas nachdem die Komponente gerendert wurde
    console.log("die Komponente wurde erfolgreich gerendert")
  });

  return <div>
    |   Meine Komponente
    </div>;
}
```

Ohne Angabe eines Arrays:
Code wird immer ausgeführt nachdem die Komponente neu gerendert wurde

```
import { useEffect } from 'react';

export default function MyComponent() {

  const [myState, setMyState] = useState("")

  useEffect(() => {
    //Tue etwas nachdem die Komponente gerendert wurde
    console.log("die Komponente wurde zum ersten mal gemountet"),[]);
  },[]);

  return <div>
    |   Meine Komponente
    </div>;
}
```

Angabe eines leeren Arrays:
Code wird nur ausgeführt, nachdem die Komponente zum ersten Mal gerendert wurde

```
import { useEffect } from 'react';

export default function MyComponent() {

  const [myState, setMyState] = useState("")

  useEffect(() => {
    //Tue etwas nachdem die Komponente gerendert wurde
    console.log("der Zustand wurde erfolgreich verändert"),[myState]);
  },[myState]);

  return <div>
    |   Meine Komponente
    </div>;
}
```

Angabe eines Arrays mit Zuständen:
Code wird immer ausgeführt, wenn mindestens einer der angegebenen Zustände sich geändert hat

Übung zu React useEffect

- Füge zusätzliche Nebeneffekte (useEffect) in deiner Counter App ein
- Jedes mal wenn die Komponente neu gerendert wird gib den Text „Komponente wurde neu gerendert“ in der Konsole aus
- Jedes Mal wenn die Komponente gemountet wird (zum ersten mal gerendert) gib den Text „Komponente wurde gemountet“ in der Konsole aus
- Jedes Mal wenn sich der Zustand des Counters ändert, gib den Text „Counter: " + Wert des Zustands in der Konsole aus



Best Practices useEffect

- Missing Dependencies vermeiden
 - Problem: Alle Zustände die innerhalb des useEffect - Codes verwendet werden, müssen als Abhängigkeiten im Dependency Array angegeben werden
 - Liefert zwar keinen Fehler sondern eine Warnung, sollte allerdings trotzdem vermieden werden.
- Event spezifische Logik innerhalb eines Effects vermeiden
 - Problem: Unnötige Zustandsänderung bevor eine Funktion ausgeführt wird
- Aktualisierungen von Zuständen basierend auf Props oder Zuständen vermeiden
 - Problem: Unnötiges Neurendern durch vermeidbare voneinander abhängige Zustände
- Weitere Best Practices
 - <https://react.dev/learn/you-might-not-need-an-effect>

Missing Dependencies vermeiden

```
function VideoPlayer({ src, isPlaying }) {  
  const ref = useRef(null);  
  
  useEffect(() => {  
    if (isPlaying) {  
      console.log('Calling video.play()');  
      ref.current.play();  
    } else {  
      console.log('Calling video.pause()');  
      ref.current.pause();  
    }  
  }, []); // This causes an error  
  
  return <video ref={ref} src={src} loop playsInline />;  
}
```

Lint Error

14:6 - React Hook useEffect has a missing dependency: 'isPlaying'. Either include it or remove the dependency array.

Event spezifische Logik innerhalb eines Effects

- Problem: Unnötige Zustandsänderung bevor eine Event basierte Funktion ausgeführt wird

```
// ● Avoid: Event-specific logic inside an Effect
const [jsonToSubmit, setJsonToSubmit] = useState(null);
useEffect(() => {
  if (jsonToSubmit !== null) {
    post('/api/register', jsonToSubmit);
  }
}, [jsonToSubmit]);
```



```
function handleSubmit(e) {
  e.preventDefault();
  // ✓ Good: Event-specific logic is in the event handler
  post('/api/register', { firstName, lastName });
}
// ...
}
```

Aktualisierungen von Zuständen basierend auf Props oder Zuständen

- Problem: Unnötiges Neurendern durch vermeidbare voneinander abhängige Zustände

```
function Form() {
  const [firstName, setFirstName] = useState('Taylor');
  const [lastName, setLastName] = useState('Swift');
```



```
// ● Avoid: redundant state and unnecessary Effect
const [fullName, setFullName] = useState('');
useEffect(() => {
  setFullName(firstName + ' ' + lastName);
}, [firstName, lastName]);
// ...
}
```

```
function Form() {
  const [firstName, setFirstName] = useState('Taylor');
  const [lastName, setLastName] = useState('Swift');
  // ✅ Good: calculated during rendering
  const fullName = firstName + ' ' + lastName;
  // ...
}
```

Aufwändige Berechnung innerhalb von useEffect

- Wie können wir diesen Code refactoren?

```
function TodoList({ todos, filter }) {  
  const [newTodo, setNewTodo] = useState('');  
  
  // ● Avoid: redundant state and unnecessary Effect  
  const [visibleTodos, setVisibleTodos] = useState([]);  
  useEffect(() => {  
    setVisibleTodos(getFilteredTodos(todos, filter));  
  }, [todos, filter]);  
  
  // ...  
}
```

```
function TodoList({ todos, filter }) {  
  const [newTodo, setNewTodo] = useState('');  
  
  // ✓ This is fine if getFilteredTodos() is not slow.  
  const visibleTodos = getFilteredTodos(todos, filter);  
  // ...  
}
```

Aufwändige Berechnung innerhalb von useEffect

- Wo siehst du ein Problem mit diesem Code?
 - Wann wird die `getFilteredTodos` Funktion aufgerufen?
 - Problem: nach jedem Rendering wird die zeitaufwändige Berechnung durchgeführt auch wenn sich keine der abhängigen Zustände geändert hat
- Wie können wir diesen Code noch weiter refactoren?

```
function TodoList({ todos, filter }) {  
  const [newTodo, setNewTodo] = useState('');  
  // ✅ This is fine if getFilteredTodos() is not slow.  
  const visibleTodos = getFilteredTodos(todos, filter);  
  // ...  
}
```

React useMemo

- useMemo ist ein React Hook der verwendet wird, um teure Berechnungen nur bei Bedarf neu zu berechnen, und basierend auf ihren Abhängigkeiten zu memoisieren und zurückzugeben.
- Es ermöglicht die Kontrolle über die Abhängigkeiten der Berechnung, wodurch diese nur neu ausgeführt wird, wenn sich die angegebenen Abhängigkeiten ändern.
- Durch Memoisierung können unnötige Neuberechnungen in Komponenten vermieden werden, was die Leistung verbessern kann, insbesondere bei Berechnungen mit hohen Kosten.

```
import { useMemo, useState } from 'react';

function TodoList({ todos, filter }) {
  const [newTodo, setNewTodo] = useState('');
  const visibleTodos = useMemo(() => {
    // ✅ Does not re-run unless todos or filter change
    return getFilteredTodos(todos, filter);
  }, [todos, filter]);
  // ...
}
```

React useMemo

- Refactored Code aus unserem vorangegangenen Beispiel
- Die getFilteredTodos Methode wird nur ausgeführt wenn sich einer der angegebenen Zustände sich ändert

```
function TodoList({ todos, filter }) {
  const [newTodo, setNewTodo] = useState('');
  // ✅ This is fine if getFilteredTodos() is not slow.
  const visibleTodos = getFilteredTodos(todos, filter);
  // ...
}
```



```
import { useMemo, useState } from 'react';

function TodoList({ todos, filter }) {
  const [newTodo, setNewTodo] = useState('');
  const visibleTodos = useMemo(() => {
    // ✅ Does not re-run unless todos or filter change
    return getFilteredTodos(todos, filter);
  }, [todos, filter]);
  // ...
}
```


Refactoring Code Challenge

1. Die untenstehende Todo-Liste zeigt eine Liste von Todos an. Wenn das Kontrollkästchen "Nur aktive Todos anzeigen" aktiviert ist, werden abgeschlossene Todos nicht in der Liste angezeigt. Unabhängig davon, welche Todos sichtbar sind, zeigt der Fußzeilenbereich die Anzahl der Todos an, die noch nicht abgeschlossen sind. Vereinfachen Sie diese Komponente, indem Sie alle unnötigen Zustände und Effekte entfernen.
2. Ihre Aufgabe besteht darin, den Effekt zu entfernen, der die Liste der sichtbaren Todos im TodoList-Komponente neu berechnet. Sie müssen jedoch sicherstellen, dass `getVisibleTodos()` nicht erneut ausgeführt wird (und daher keine Protokolle druckt), wenn Sie in das Eingabefeld tippen.

☐ Show only active todos

- ~~Get apples~~
- ~~Get oranges~~
- Get carrots

React useRef Hook

- Mit useRef können Sie Referenzen auf DOM-Elemente speichern, um später darauf zuzugreifen oder sie zu manipulieren.
- useRef ermöglicht das Speichern von Werten zwischen Rerenderungen einer Komponente, ohne dass dabei ein erneutes Rendern ausgelöst wird.
- Sie können auf den aktuellen Wert der Referenz über ref.current zugreifen, der auch zwischen den Rerenderungen erhalten bleibt.
- Ähnlich dem bereits bekannten `document.getElementById(„myelement“)` aber spezifisch für React. Performanter bei mehrfachen Renderings

```
import React, { useRef } from 'react';

export default function ExampleComponent() {
  const inputRef = useRef(null);

  const focusInput = () => {
    inputRef.current.focus();
  };

  return (
    <div>
      <input ref={inputRef} type="text" />
      <button onClick={focusInput}>Focus auf Eingabefeld</button>
    </div>
  );
}
```

React Custom Hooks

- Benutzerdefinierte Hooks ermöglichen die Wiederverwendung von Logik zwischen verschiedenen Komponenten, indem sie gemeinsame Funktionalitäten in unabhängige und wiederverwendbare Hooks extrahieren.
- Sie erlauben es, komplexe Logik in abstrakte und leicht verständliche Hooks zu kapseln, was die Lesbarkeit und Wartbarkeit des Codes verbessert.
- Bsp rechts: Ein Hook der eingesetzt wird um die Netzwerkverbindung zu prüfen und einen React Status dafür zu setzen

```
function useOnlineStatus() {  
  const [isOnline, setIsOnline] = useState(true);  
  useEffect(() => {  
    function handleOnline() {  
      setIsOnline(true);  
    }  
    function handleOffline() {  
      setIsOnline(false);  
    }  
    window.addEventListener('online', handleOnline);  
    window.addEventListener('offline', handleOffline);  
    return () => {  
      window.removeEventListener('online', handleOnline);  
      window.removeEventListener('offline', handleOffline);  
    };  
  }, []);  
  return isOnline;  
}
```

Custom Hooks Forms Beispiel

- Geschickter Einsatz von Custom Hooks bei Formularen nimmt uns viel Arbeit ab und lässt uns unseren Code viel wartbarer gestalten
- <https://react.dev/learn/reusing-logic-with-custom-hooks#custom-hooks-let-you-share-stateful-logic-not-state-itself>

First name:
Last name:

Good morning, Mary Poppins.


```
export function useFormInput(initialValue) {  
  const [value, setValue] = useState(initialValue);  
  
  function handleChange(e) {  
    setValue(e.target.value);  
  }  
  
  const inputProps = {  
    value: value,  
    onChange: handleChange  
  };  
  
  return inputProps;  
}
```

Custom Hooks Challenges




- Löse die Challenges 1 - 3 des Custom Hooks Kapitel
- <https://react.dev/learn/reusing-logic-with-custom-hooks#challenges>
- Diese Komponente verwendet eine Zustandsvariable und einen Effekt, um eine Zahl anzuzeigen, die sich alle Sekunde erhöht. Extrahiere diese Logik in einen benutzerdefinierten Hook namens useCounter. Dein Ziel ist es, die Implementierung der Counter-Komponente genau so aussehen zu lassen:

Seconds passed: 132




Glean VS-Code Extension



glean v5.2.2

Wix  wix.com |  198,975 |  (29)

The extension provides refactoring tools for your React codebase

[Disable](#)  [Uninstall](#)  

This extension is enabled globally.

Browser Plugin

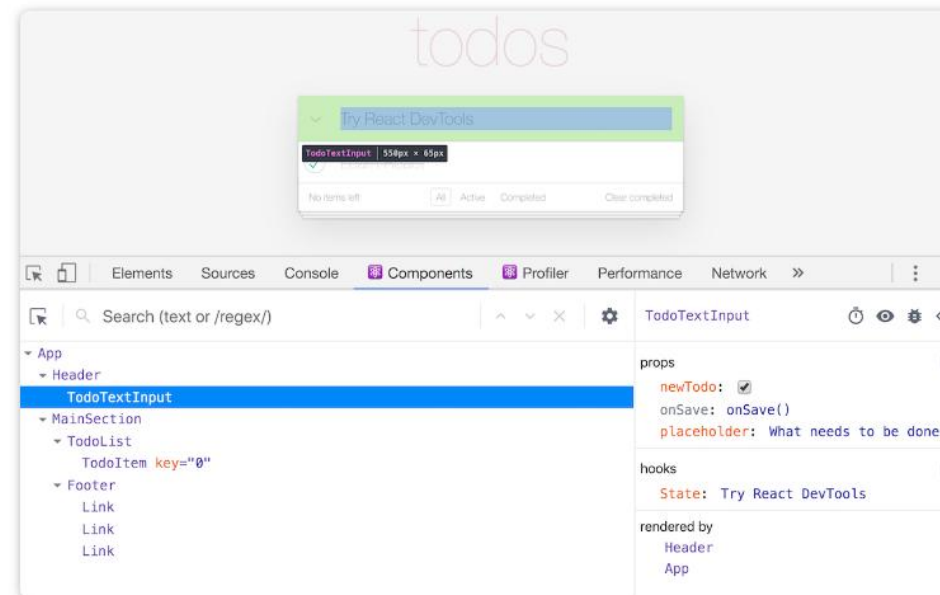


Empfohlen 4,0 ★ (1.489 Bewertungen)

Erweiterung

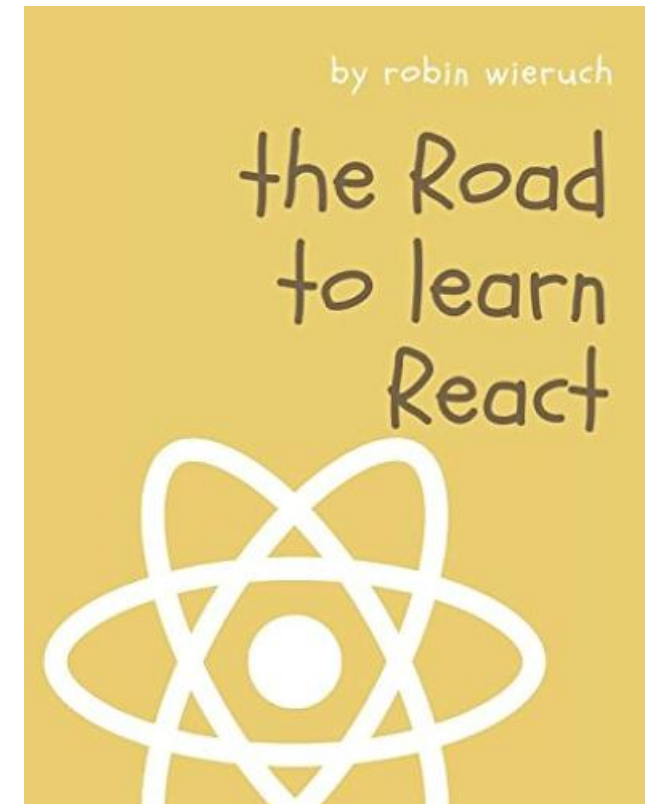
Entwicklertools

4.000.000 Nutzer



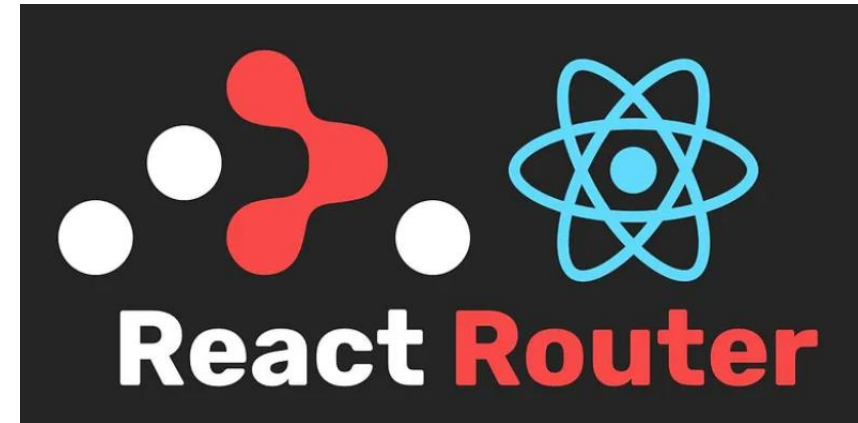
Zwischenfazit

- Wo befinden wir uns gerade?
 - Wir haben bereits die Grundlagen um erfolgreich ein modernes React Frontend aufzubauen gelernt
 - Einige tieferliegende Konzepte und Methoden bzw Frameworks werden wir, wenn gewünscht in den Vertiefungsthemen kennen lernen (Next.js)
- Was fehlt uns um ein Fullstack Projekt mit React aufzubauen?
 - **Routing:** Wie können wir von einer Seite auf eine andere navigieren ohne die Seite neu zu laden?
 - **Backendanbindung:** Wie können wir aus dem React Frontend heraus die Endpoints unseres Express servers aufrufen und Daten abfragen?



React Router DOM

- React Router DOM ist eine Bibliothek (NPM-Package) für das Routing in React-Anwendungen.
- Es ermöglicht das Definieren von Routes, die bestimmten URLs entsprechen.
- Mit React Router DOM werden verschiedene Komponenten basierend auf der aktuellen URL gerendert.
- Es bietet Funktionen wie `<BrowserRouter>`, `<Route>` und `<Link>`
- Diese Bibliothek ist weit verbreitet und wird von der React-Community aktiv unterstützt.



React Router DOM – Routing Konfiguration

1. Installiere das NPM-Paket „react-router-dom“ (npm i react-router-dom)
2. Innerhalb der App.js Datei wird die Routing Konfiguration innerhalb von return(...) definiert
 - <BrowserRouter> erstellt ein Browserverlauf, speichert die initiale URL im Zustand und abonniert die URL-Änderungen.
 - <Routes> durchläuft seine untergeordneten Routen, um eine Routenkonfiguration aufzubauen, gleicht diese Routen mit der aktuellen URL ab, erstellt einige Routenübereinstimmungen und rendert das Routenelement der ersten Übereinstimmung

```
export default function App() {  
  return(  
    <BrowserRouter>  
      <Routes>  
        <Route index element={<DataFetch />} />  
        <Route path="counter" element={<Counter />} />  
      </Routes>  
    </BrowserRouter>  
  )  
}
```

React Router DOM – Route

- `<Route>` definiert eine einzelne Route in deiner Anwendung. Es hat verschiedene Props, um zu bestimmen, wann die Route aktiviert wird und welche Komponente gerendert werden soll.
 - `path`: Diese Prop definiert den Pfad, der mit der aktuellen URL übereinstimmen muss, damit diese Route aktiviert wird.
 - `element`: Diese Prop definiert die Komponente, die gerendert wird, wenn die Route aktiviert ist.
 - `index`: Diese Prop definiert, dass die Route die Index-Route ist, was bedeutet, dass sie aktiviert wird, wenn der Pfad der Root-URL entspricht.

```
export default function App() {  
  return(  
    <BrowserRouter>  
      <Routes>  
        <Route index element={<DataFetch />} />  
        <Route path="counter" element={<Counter />} />  
      </Routes>  
    </BrowserRouter>  
  )  
}
```

React Path Params

- Um Pfadparameter verwenden zu können hänge einen Parameter beim Pfad mit `"/:NameDesPfadparameters"` an

Beispiel:

```
<Route path="specificjoke/:id" element={<SpecificJoke />} />
```

- Um den Pfadparameter in deiner React-Komponente, die bei diesem Pfad aufgerufen wird verwenden zu können verwende folgende Syntax:

```
const {NameDesPfadparameters} = useParams()
```

Beispiel:

```
const { id } = useParams();
```

- Danach kann die Variable ganz normal in der Komponente verwendet werden

React Query Params

- Um Query Parameter verwenden zu können benötigen wir zunächst folgendes importiertes Package

```
import { useLocation } from "react-router-dom";
```

- Danach werden die folgenden Codezeilen innerhalb der Komponente geschrieben die die Query Parameter verwendet

```
const location = useLocation();  
const queryParams = new URLSearchParams(location.search);
```

- Um die einzelnen in queryParams gespeicherten Variablen nutzen zu können wird die folgende Syntax verwendet: `const variableName = queryParams.get(„queryParamName“)`

Bsp:

```
const type = queryParams.get("type");
```

React Navigation

- Wie können wir innerhalb einer Komponente eine andere Route aufrufen ohne dabei die Seite neu zu laden?
- React Router DOM bietet 2 Möglichkeiten:
- Als Link:
`<Link to="endpoint">`
- Über Funktionsaufruf:
`navigate(„endpoint“)`

```
import React, { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
import { Link } from "react-router-dom";

export default function MenuComponent() {
  const navigate = useNavigate();
  return (
    <div>
      <Link to="randomjoke">Zum Counter</Link>
      <button onClick={() => navigate("specificjoke/1")}>Zum Witz 1</button>
      <button onClick={() => navigate("filteredjokes?type=Science")}>Zu den
    </div>
  );
}
```

React / Express Backend Integration

- Um Aufrufe aus dem React Frontend an den Express Server und dessen Endpoints machen zu können sind folgende Schritte nötig:
 1. Starten des Express Servers (nodemon server.js) auf beliebigen Port (Bsp: 3000)
 2. Starten der React App (npm start) auf unterschiedlichem Port (Bsp:3001)
 3. Mit Hilfe der bekannten fetch Methode werden Daten wie gewohnt aus dem Backend geladen
 - Häufig werden Daten beim Initialen Rendering (useEffect) einer Komponente oder als Folge einer Benutzeraktion (Event) geladen

```
useEffect(() => {  
  fetch("http://localhost:3000/filter?type=" + type)  
    .then((response) => response.json())  
    .then((data) => setJoke(data));  
}, []);
```

```
function myFetchFunction() {  
  fetch("http://localhost:3000/jokes/" + id)  
    .then((response) => response.json())  
    .then((data) => setJoke(data.jokeText));  
}
```

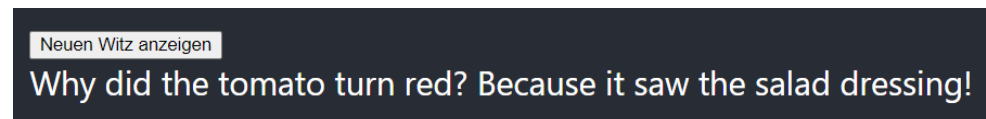
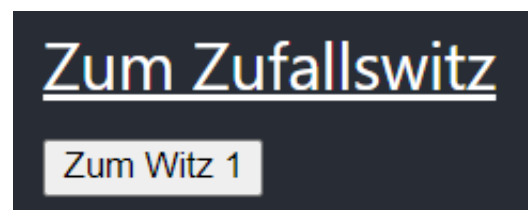
Cross-Origin Resource Sharing (CORS)

- Wenn wir 2 unterschiedliche Server in Entwicklung für Backend (Express) und Frontend (React) verwenden, nutzen wir das sogenannte Cross-Origin Resource Sharing (CORS)
- Diese Konfiguration muss explizit innerhalb unseres Express Servers „genehmigt“ werden
- Dafür verwenden wir eine Middleware die es uns erlaubt CORS Anfragen zu nutzen
- Dafür wird ein zusätzliches Paket installiert (npm i cors)
- Im Anschluss wird das Package in unserem Server importiert und über eine Middleware registriert (siehe rechts)
- Nun können wir Anfragen von unterschiedlichen Quellen (unterschiedlichen Webseiten) mit unserem Server verarbeiten

```
import cors from "cors"  
app.use(cors())
```

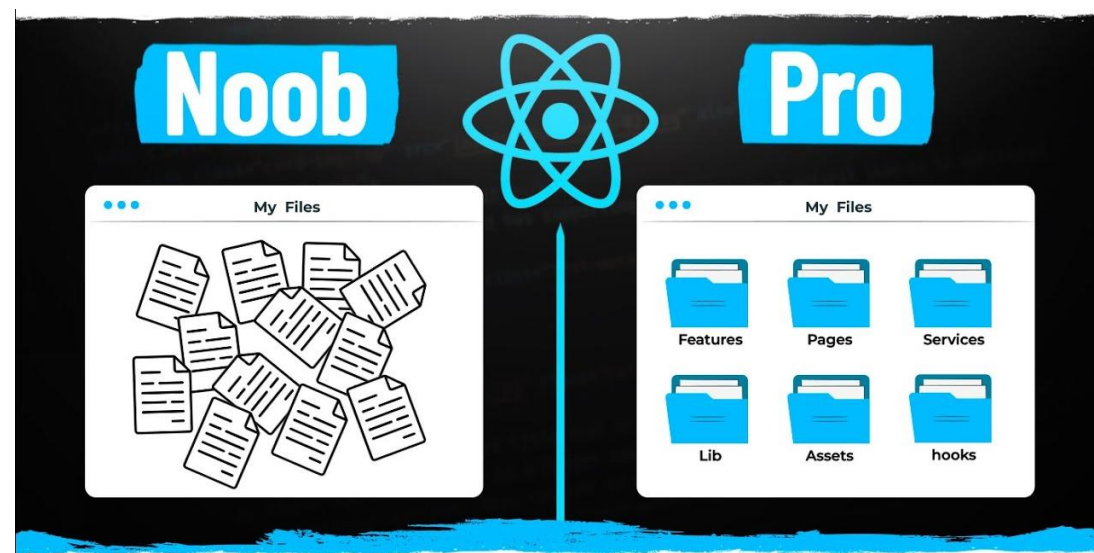

Übung: Witze API Aufruf mit fetch() und React

- Implementiere ein React Frontend mit 3 Routes (= 3 Komponenten)
 - Menu
 - RandomJoke
 - SpecificJoke/1
- Die Menu Komponente beinhaltet einen Link zur Random Joke Route und einen Button "Zum Witz 1" zum Route SpecificJoke/1
- Die RandomJoke Komponente beinhaltet einen Button der beim Klick einen neuen Zufallswitz vom Backend lädt und ausgibt
- Die SpecificJoke Komponente nutzt Pfad Parameter um einen speziellen Witz mit der id = 1 vom Backend zu laden und anzuzeigen



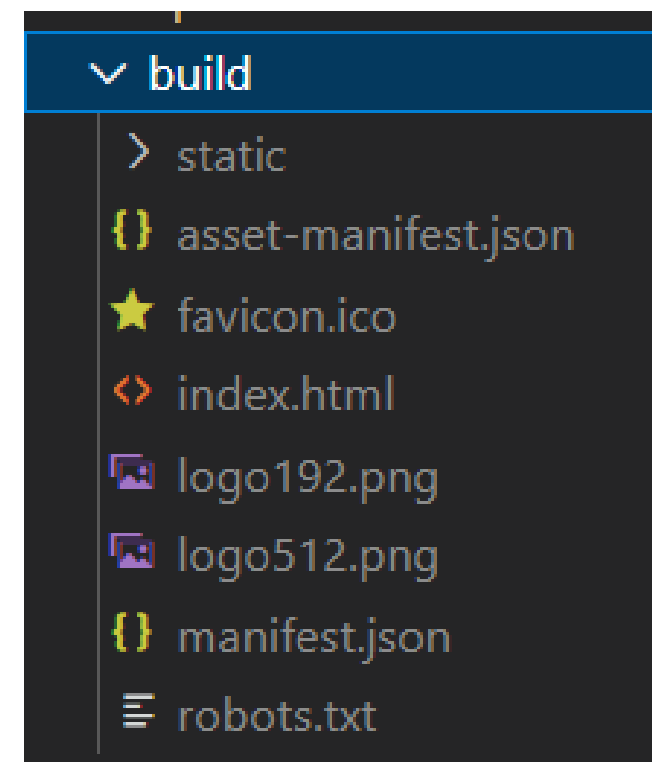
React Ordner-Struktur

- Code sollte in Ordner gruppiert werden um einen besseren Überblick zu bekommen und den Code wartbarer zu gestalten
- Welche Möglichkeiten bieten sich uns beim Gruppieren unserer Dateien?
 - nach Dateityp (CSS, React Komponenten)
 - nach Routes / Seiten (Bsp: ./home)
 - nach Fachlichkeit (Authentifizierung, Benutzerverwaltung)
- Wichtig ist dass Dateien nicht willkürlich in Ordner gespeichert werden sondern überlegt und einem ständigen Refactoring unterzogen werden
- Tipp: Versuche Ordnerstrukturen zu schaffen, um so wenig Abhängigkeiten wie möglich zwischen den Ordnern zu bekommen (Imports sollten innerhalb des Ordners passieren)



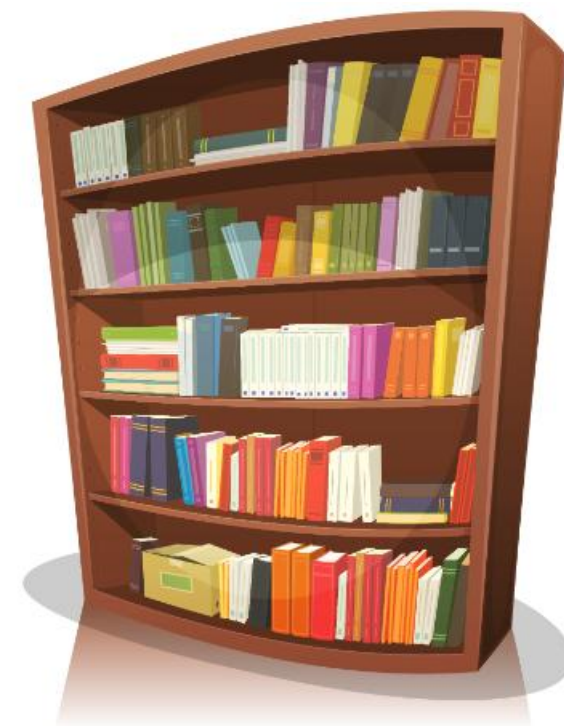
Befehl „npm run build“

- Wird in React-Anwendungen verwendet, um den Produktionsbuild der Anwendung zu erstellen.
- Dieser Befehl kompiliert und optimiert den React-Code sowie seine Abhängigkeiten für die Bereitstellung in einer Produktionsumgebung.
- Es führt verschiedene Aufgaben aus, einschließlich des Zusammenführens und Minimierens von JavaScript-, CSS- und Bilddateien, um die Leistung und Ladezeiten der Anwendung zu verbessern.
- Der erzeugte Build wird normalerweise in einem "build" oder "dist" Verzeichnis im Projekt gespeichert und kann dann auf einem Webserver bereitgestellt werden.
- Durch die Ausführung von "npm run build" können Entwickler sicherstellen, dass ihre React-Anwendung für den Einsatz in der Produktion optimiert ist



React Component Libraries

- **Wiederverwendbare UI-Komponenten:** React-Komponentenbibliotheken bieten vorgefertigte UI-Komponenten, die in React-Anwendungen wiederverwendet werden können.
- **Beschleunigte Entwicklung:** Sie beschleunigen die Entwicklung, indem sie Entwicklern ermöglichen, häufig verwendete UI-Muster und -Elemente schnell zu implementieren, ohne von Grund auf neu zu beginnen.
- **Konsistente Benutzeroberfläche:** Durch die Verwendung einer Bibliothek können Entwickler eine konsistente Benutzeroberfläche erstellen, die den Designprinzipien der Bibliothek folgt.
- **Modularität und Anpassbarkeit:** Diese Bibliotheken bieten modulare Komponenten, die leicht anpassbar sind, um den Anforderungen und dem Stil eines bestimmten Projekts gerecht zu werden.





Material UI



grommet

Grommet



React Redux



React Router



Blueprint UI



Fluent UI



React Bootstrap



Semantic UI React



chakra

Chakra UI



React Admin

<https://ably.com/blog/best-react-component-libraries>

Alternative: <https://atlaskit.atlassian.com/>

Was bietet Material UI?

- **React-Komponentenbibliothek:** Material-UI bietet eine Vielzahl von vorgefertigten React-Komponenten, die die Entwicklung von Benutzeroberflächen in React-Anwendungen vereinfachen.
- **Material Design-Prinzipien:** Es folgt den Designprinzipien von Google's Material Design, was bedeutet, dass die Komponenten ein einheitliches und ansprechendes Erscheinungsbild haben.
- **Anpassbare Themen:** Material-UI ermöglicht die einfache Anpassung des Erscheinungsbilds der Anwendung durch die Verwendung von individuellen Designthemen und Farbschemata.
- **Responsive und barrierefreie Komponenten:** Die Komponenten von Material-UI sind responsive und unterstützen eine einfache Integration von barrierefreien Features, was zu einer benutzerfreundlichen und zugänglichen Benutzeroberfläche führt.



<https://mui.com/>

Material UI Übung

- Nutze in deiner React Todo App mindestens eine Material UI Komponenten
 - [Button](#)
 - [Textfeld](#) (Bsp: Eingabe eines neuen Todos)
 - [Liste](#) (Bsp: Anzeige aller Todos)
 - ...
- (Fortgeschritten aber sehr hilfreich!!)
Baue eine Maske mit der in einem MUI-X Datagrid alle Todos und deren Inhaber gelistet sind. Zusätzlich kann nach Todos/Benutzern gesucht und gefiltert werden



Vertiefungsthemen

- Abstimmung wie würdet ihr die Priorität der Themen reihen
- <https://codersbay.feedcube.net/survey?id=293&course=213&trainer=73&language=de&target=Kursteilnehmer&anonym=true>



Viel Erfolg beim Entwickeln!

