

NILS HARTMANN

# React

**Einsteigen und loslegen - eine praktische Einführung**

Slides: <https://react.schule/jax2021-react>

JAX ONLINE | 6. MAI 2021 | @NILSHARTMANN

# NILS HARTMANN

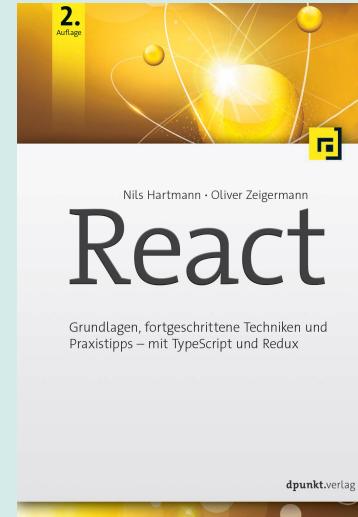
Kontakt: [nilshartmann.net](mailto:nilshartmann.net)

Twitter: [@nilshartmann](https://twitter.com/nilshartmann)

## Freiberuflicher Entwickler, Architekt, Trainer aus Hamburg

Java  
JavaScript, TypeScript  
React  
GraphQL

Trainings & Workshops



<https://reactbuch.de>

[HTTPS://NILSHARTMANN.NET](https://nilshartmann.net)

# React

# Einführung

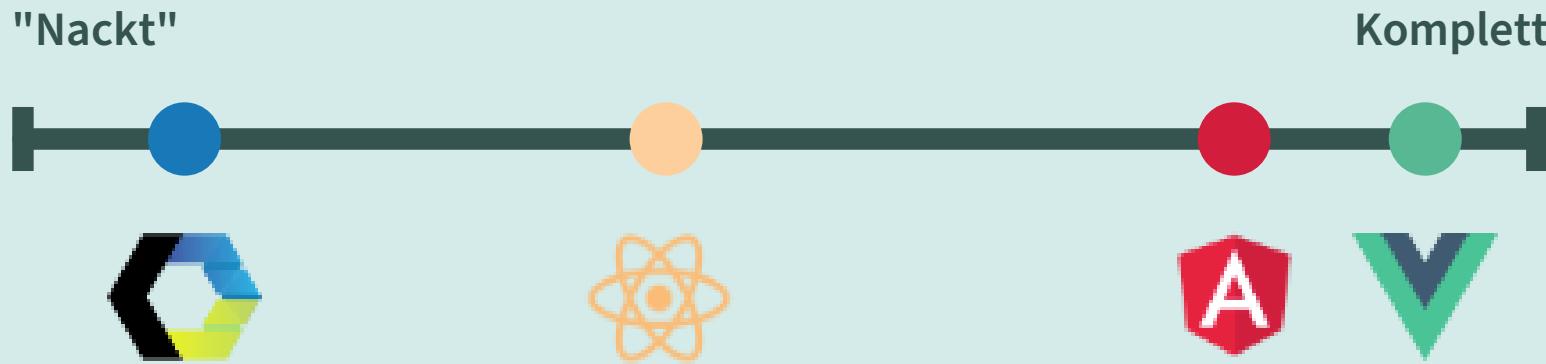
<https://reactjs.org>

- Minimales API
- Minimales Feature Set

# REACT

<https://reactjs.org>

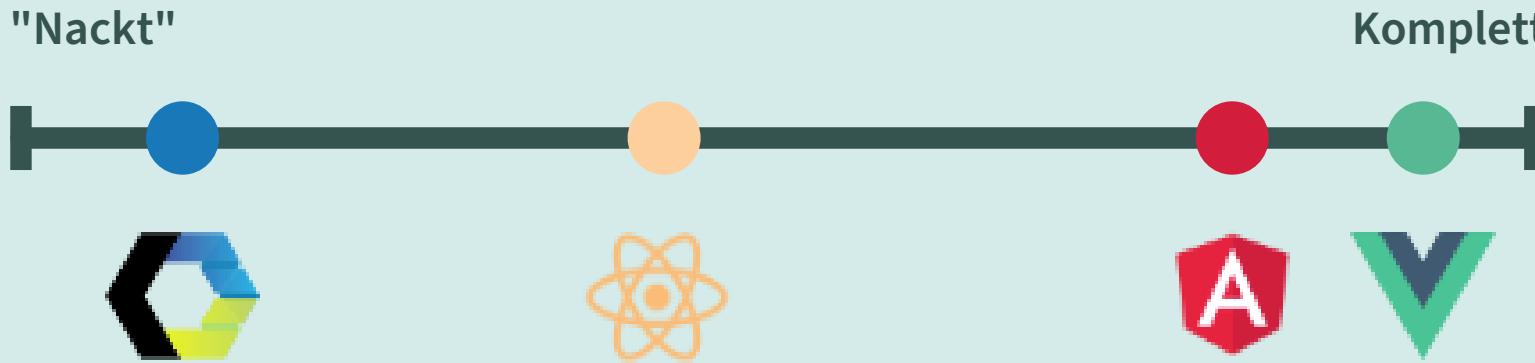
- Minimales API
- Minimales Feature Set



# REACT

<https://reactjs.org>

- Minimales API
  - Minimales Feature Set
- 👉 Man kann (muss ?) viele Entscheidungen selber treffen
- 👉 In ernsthaften Anwendungen werden weitere Bibliotheken benötigt



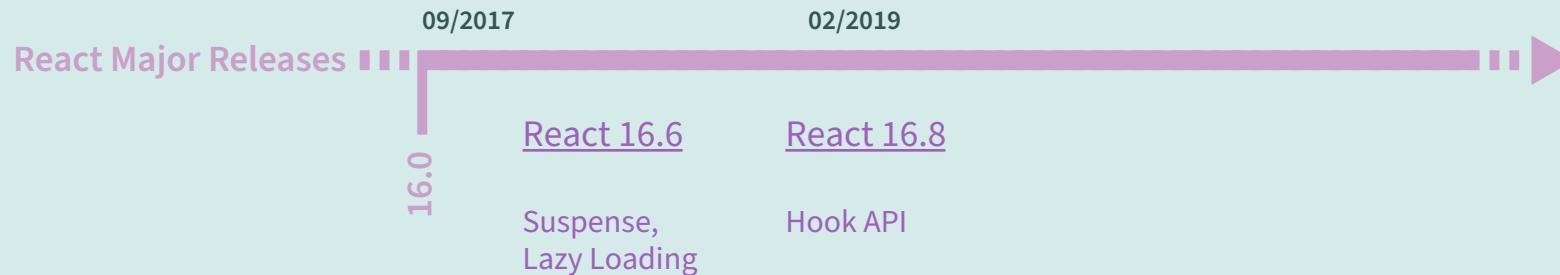
# RELEASEZYKLEN

## Sehr lange Release-Zyklen



# RELEASEZYKLEN

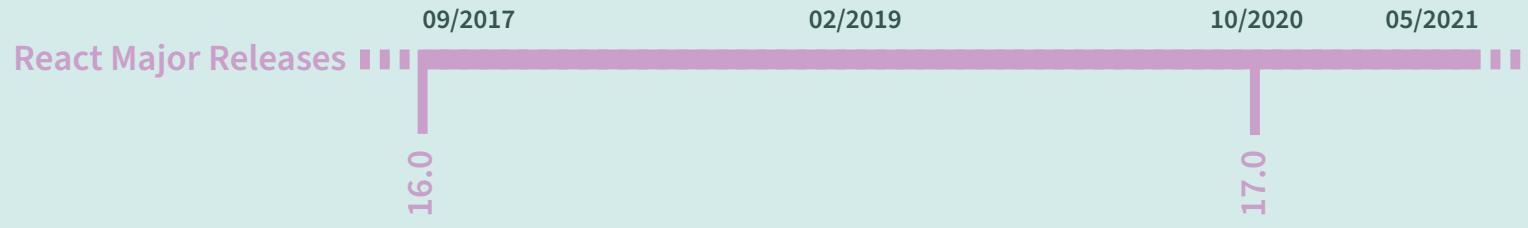
## Sehr lange Release-Zyklen



Neue Features und APIs in Minor-Versionen!

# RELEASEZYKLEN

## Sehr lange Release-Zyklen

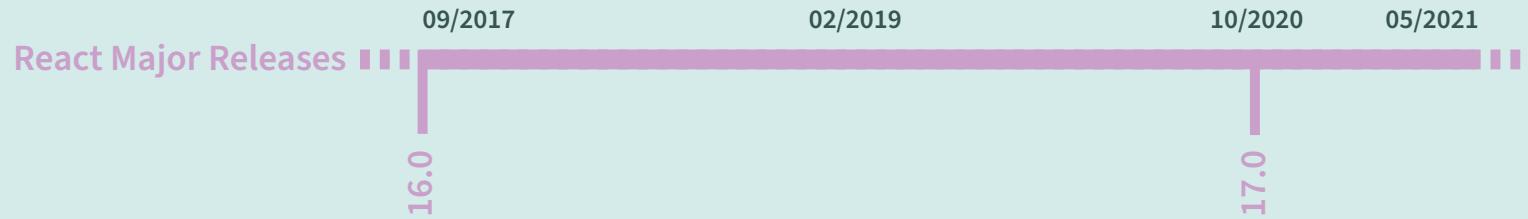


### ↗ No New Features

The React 17 release is unusual because it doesn't add any new developer-facing features. Instead, this release is primarily focused on **making it easier to upgrade React itself**.

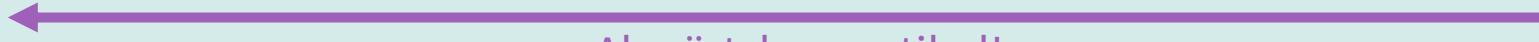
# RELEASEZYKLEN

## Sehr lange Release-Zyklen



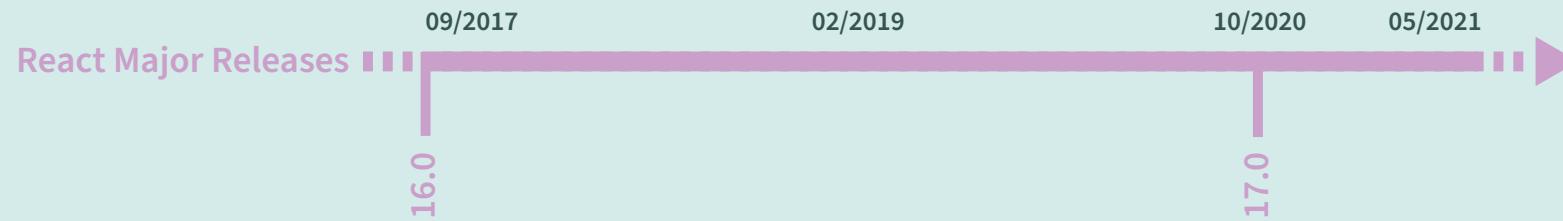
### ↳ No New Features

The React 17 release is unusual because it doesn't add any new developer-facing features. Instead, this release is primarily focused on **making it easier to upgrade React itself**.



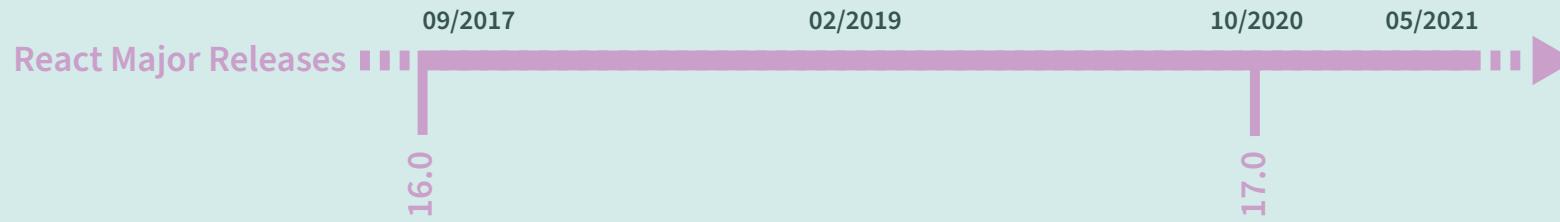
# RELEASEZYKLEN

## Sehr lange Release-Zyklen

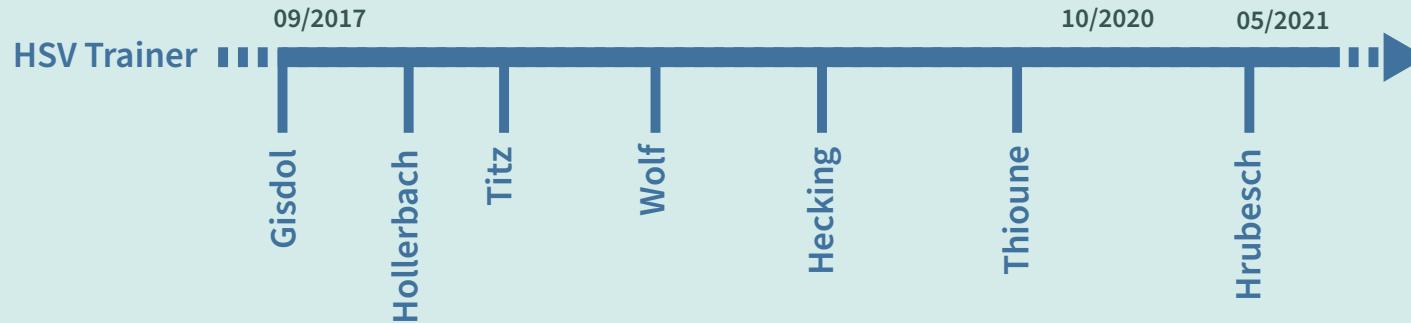


# RELEASEZYKLEN

## Sehr lange Release-Zyklen



## Sehr kurze Release-Zyklen (Symbolbild)



# Greeting App

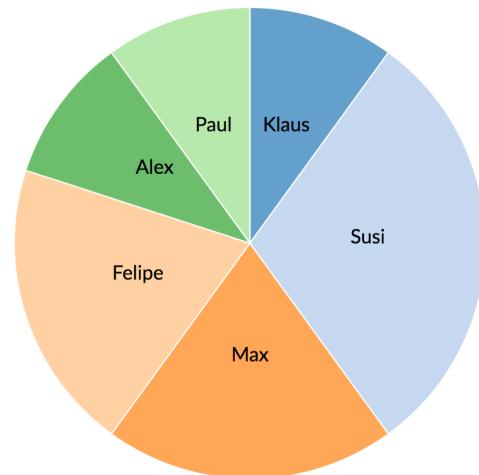
Showing 10 of 10 Greetings

Name	Greeting
Klaus	Moin
Susi	Hello!
Max	Bonjour
Susi	How are you?
Max	Bon soir
Felipe	Hola, ¿qué tal?
Alex	Happy Birthday
Felipe	¡buenos días
Paul	Wie gehts?
Susi	Have a nice day

(All greetings are shown. Click a row to filter)

Add

● Klaus   ● Susi   ● Max   ● Felipe   ● Alex   ● Paul



## BEISPIEL: DIE GREETING APP

# React Komponenten

App

## Greeting App

GreetingController

Name	Greeting
Klaus	Moin
Susi	Hello!
Max	Bonjour
Susi	How are you?
Max	Bon soir
Felipe	Hola, ¿qué tal?
Alex	Happy Birthday
Felipe	¡buenos días
Paul	Wie gehts?
Susi	Have a nice day

(All greetings are shown. Click a row to filter)

Add

FilterPanel

GreetingMaster

Counter

Showing 10 of 10 Greetings

Chart

The pie chart displays the proportion of greetings for each person. The segments are labeled with the names: Klaus, Susi, Max, Felipe, Alex, and Paul. The sizes of the segments correspond to the frequency of their greetings in the list.

Name	Proportion
Klaus	~10%
Susi	~30%
Max	~20%
Felipe	~20%
Alex	~10%
Paul	~10%

GREETING APP: KOMPONENTEN

# React Komponenten

Klassische Aufteilung

**Logik, Model**  
(Java, JS)



**View**  
(HTML, Template)



**Gestaltung**  
(CSS)



Aufteilung in Komponenten



**Button**

**GreetingEditor**

**Counter**

Grafik Inspiriert von: [https://pbs.twimg.com/media/DCXJ\\_tjXoAAoBbu.jpg](https://pbs.twimg.com/media/DCXJ_tjXoAAoBbu.jpg)

## KOMPONENTEN

# React-Komponenten

- bestehen aus Logik und UI
- keine Templatesprache
- werden **deklarativ** beschrieben



Button



GreetingEditor



Counter

# EINE EINFACHE REACT KOMPONENTE

Darstellung

Showing 3 of 11 Greetings

# EINE EINFACHE REACT KOMPONENTE

Darstellung

Showing 3 of 11 Greetings

Komponenten sind JavaScript-Funktionen

```
function Counter() {
```

```
}
```

# EINE EINFACHE REACT KOMPONENTE

Darstellung

Showing 3 of 11 Greetings

```
function Counter() {  
  return <div>Showing 3 of 11 Greetings</div>  
}  

```

# EINE EINFACHE REACT KOMPONENTE

Darstellung

Showing 3 of 11 Greetings

```
function Counter({filtered, total}) {  
}  
}
```

The diagram illustrates the mapping between the displayed state and the component's internal properties. Two dashed orange arrows originate from the word 'Properties' at the bottom center and point to the 'filtered' and 'total' parameters within the function signature of the Counter component.

# EINE EINFACHE REACT KOMPONENTE

Darstellung

Showing [3] of [11] Greetings

```
function Counter({filtered, total}) {  
  return filtered === total ?  
    <div>Showing all {total} Greetings</div>  
    :  
    <div>Showing {filtered} of {total} Greetings</div>  
}
```

# EINE EINFACHE REACT KOMPONENTE

Darstellung

Showing 3 of 11 Greetings

Counter.js

```
function Counter({filtered, total}) {  
  return filtered === total ?  
    <div>Showing all {total} Greetings</div>  
    :  
    <div>Showing {filtered} of {total} Greetings</div>  
}
```

Verwendung

```
<Counter filtered={3} total={11} />
```

# KOMPONENTEN WERDEN ZU APPLIKATIONEN AGGREGIERT

```
import Counter from './Counter';
import Greeting from './GreetingTable';
import Chart from './Chart';

function App() {
  return (
    <main>
      <header>
        <Counter filtered={3} total={11} />
      </header>
      <div className="Left">
        <GreetingTable />
      </div>
      <div className="Right">
        <Chart />
      </div>
    </main>
  )
}
```

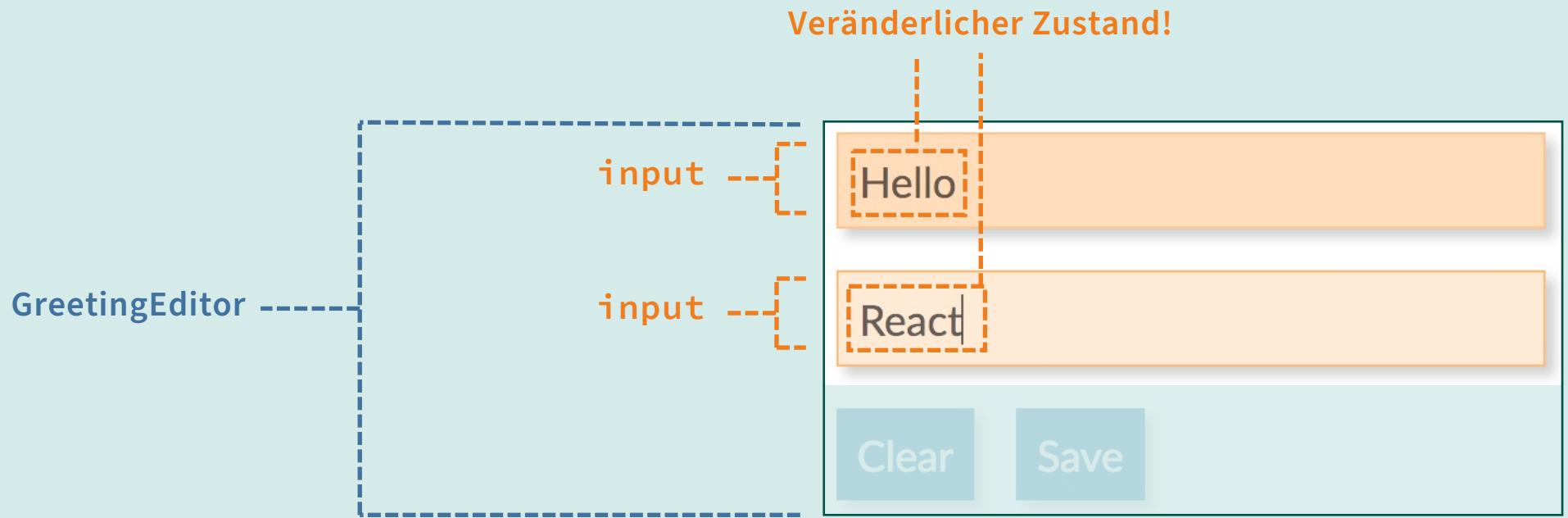
**Model a.k.a**

**State**

**ARBEITEN MIT VERÄNDERLICHEN DATEN**

# STATE

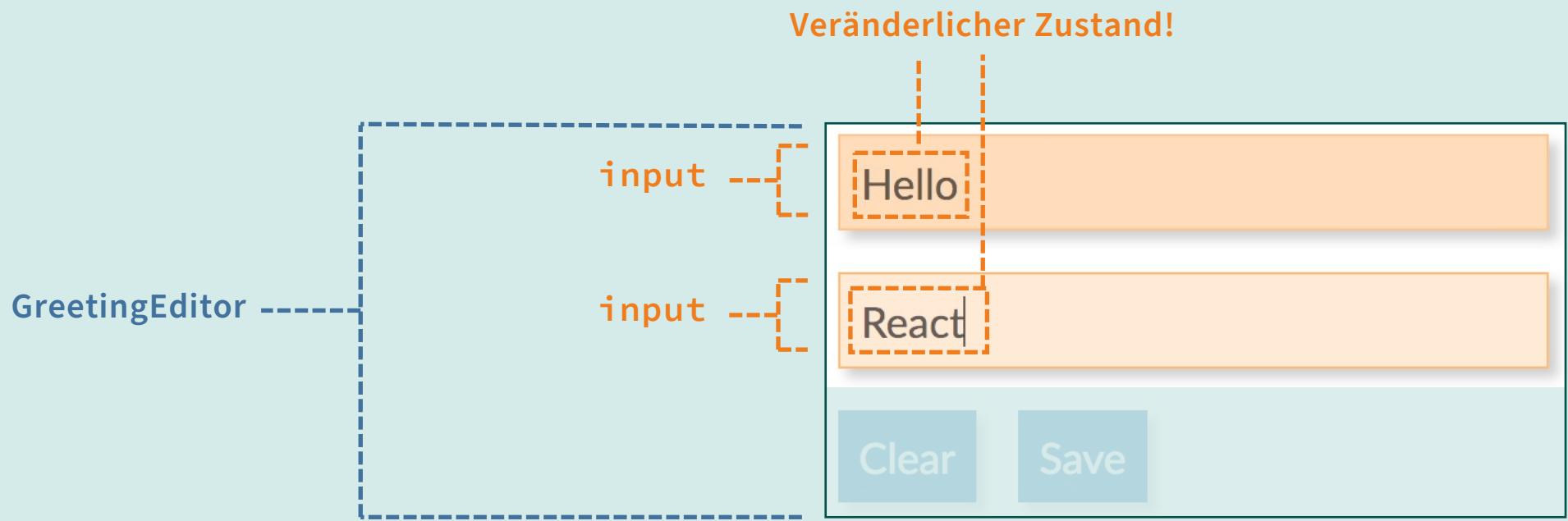
## Beispiele: Eingabefeld



- Mit `useState` wird ein "Model" erzeugt ("State" in React genannt)

# STATE

## Beispiele: Eingabefeld



### Andere typische Beispiele:

- geladene Daten vom Server
- Menü aufgeklappt/zugeklappt
- Modaler Dialog sichtbar/unsichtbar

## BEISPIEL: EINGABEFELD

Komponente

Hello

-- input

"Hook"-Funktion

```
function GreetingEditor() {  
  const [ phrase, setPhrase ] = React.useState("Hello");
```

|

Aktueller State    Setter

|

Initialer Wert

```
}
```

- Mit useState wird ein "Model" erzeugt ("State" in React genannt)

## BEISPIEL: EINGABEFELD

Komponente

Hello

]}-- input

"Hook"-Funktion

1. Input mit Wert aus State befüllen

2. Zustand neu setzen

```
function GreetingEditor() {  
  const [ phrase, setPhrase ] = React.useState("Hello");  
  
  return <div>  
    <input  
      value={phrase}  
      onChange={e => setPhrase(e.target.value)}  
    />  
  </div>  
}
```

# BEISPIEL: EINGABEFELD

Komponente

Hello

]-- input

"Hook"-Funktion

1. Input mit Wert aus State befüllen
2. Zustand neu setzen

```
function GreetingEditor() { ←----- Neu rendern  
  const [ phrase, setPhrase ] = React.useState("Hello");
```

```
  return <div>  
    <input  
      value={phrase} ↓  
      onChange={e => setPhrase(e.target.value)}  
    />  
  </div>  
}
```

Event

Zustand ändern

## Besonderheiten:

- Kein 2-Wege-Databinding
- Bei Änderung am Zustand wird Funktion neu ausgeführt

# RENDERING VON KOMPONENTEN

## Gerendert wird immer die **ganze** Komponente

- Inklusive aller Unterkomponenten
- Bei jeder Zustandsänderung
- Verhindert Inkonsistenzen

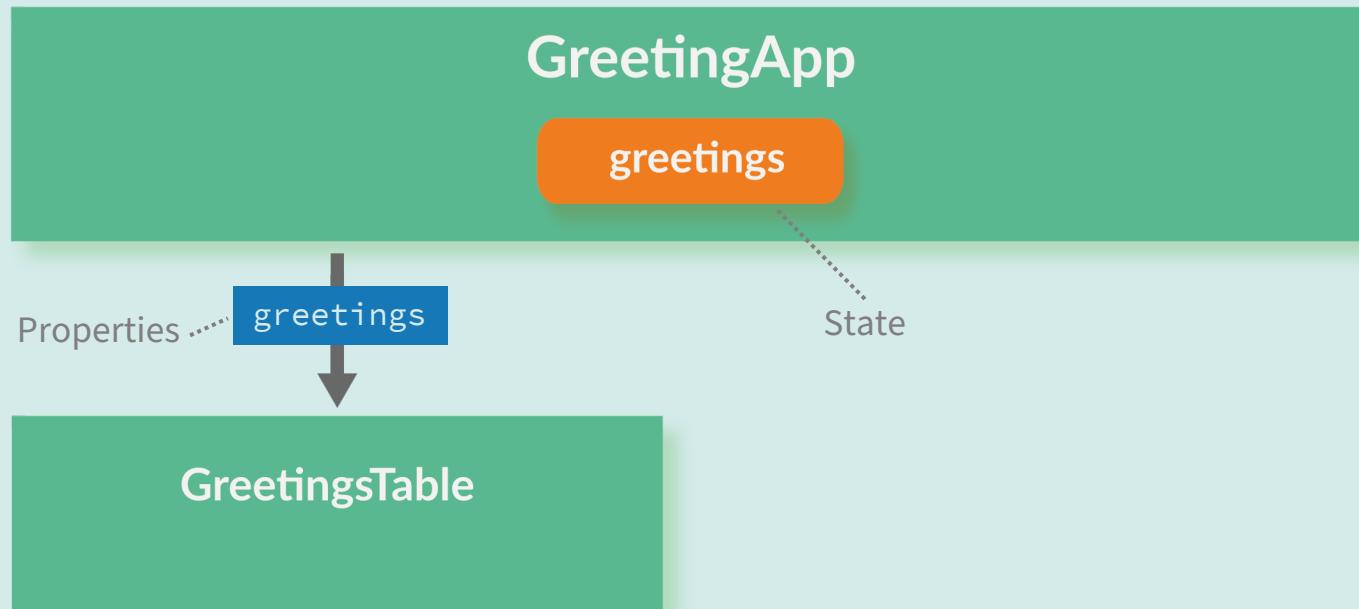


```
function GreetingEditor() {  
  const [ phrase, setPhrase ] = React.useState("Hello");  
  const [ name, setName ] = React.useState("React");  
  const saveDisabled = phrase === "" || name === "";  
  
  return <div>  
    <input value={phrase} onChange={e => setPhrase(e.target.value)} />  
    <input value={name} onChange={e => setName(e.garget.value)} />  
    <button disabled={saveDisabled}>Save</button>  
  </div>  
}
```

# HIERARCHIEN VON KOMPONENTEN

## Gemeinsame Daten wandern in gemeinsame Oberkomponenten

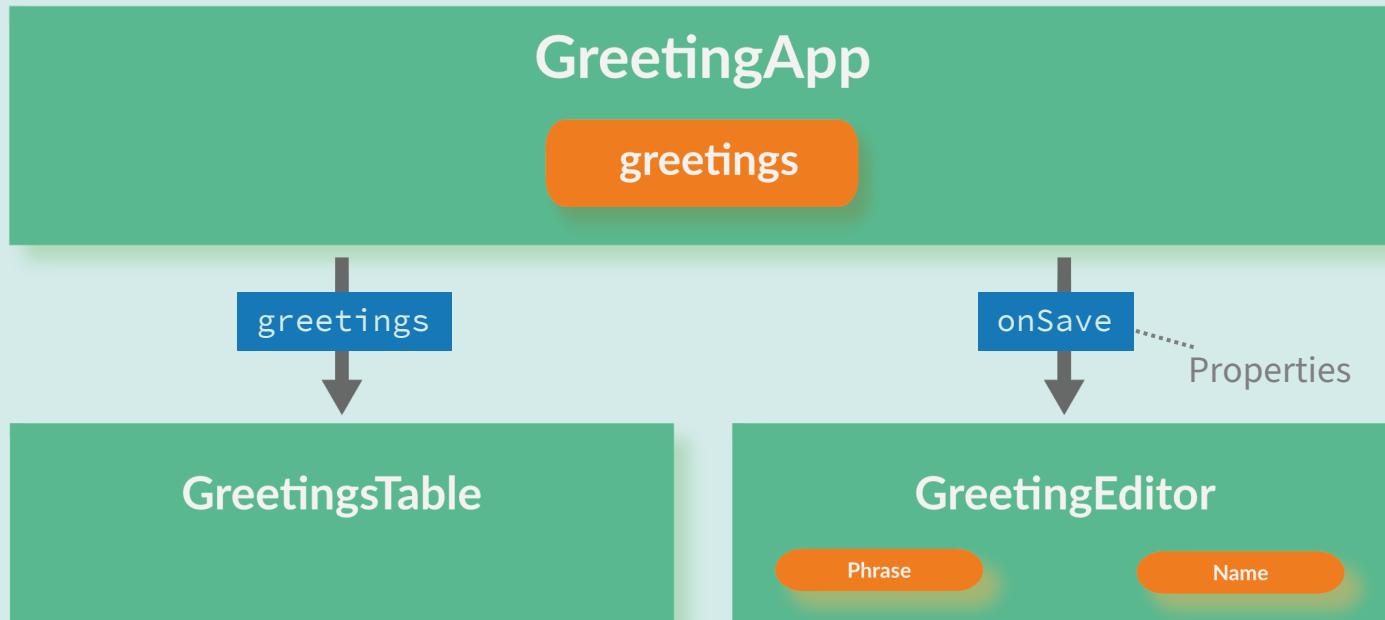
- Von dort per Properties nach unten



# HIERARCHIEN VON KOMPONENTEN

## Gemeinsame Daten wandern in gemeinsame Oberkomponenten

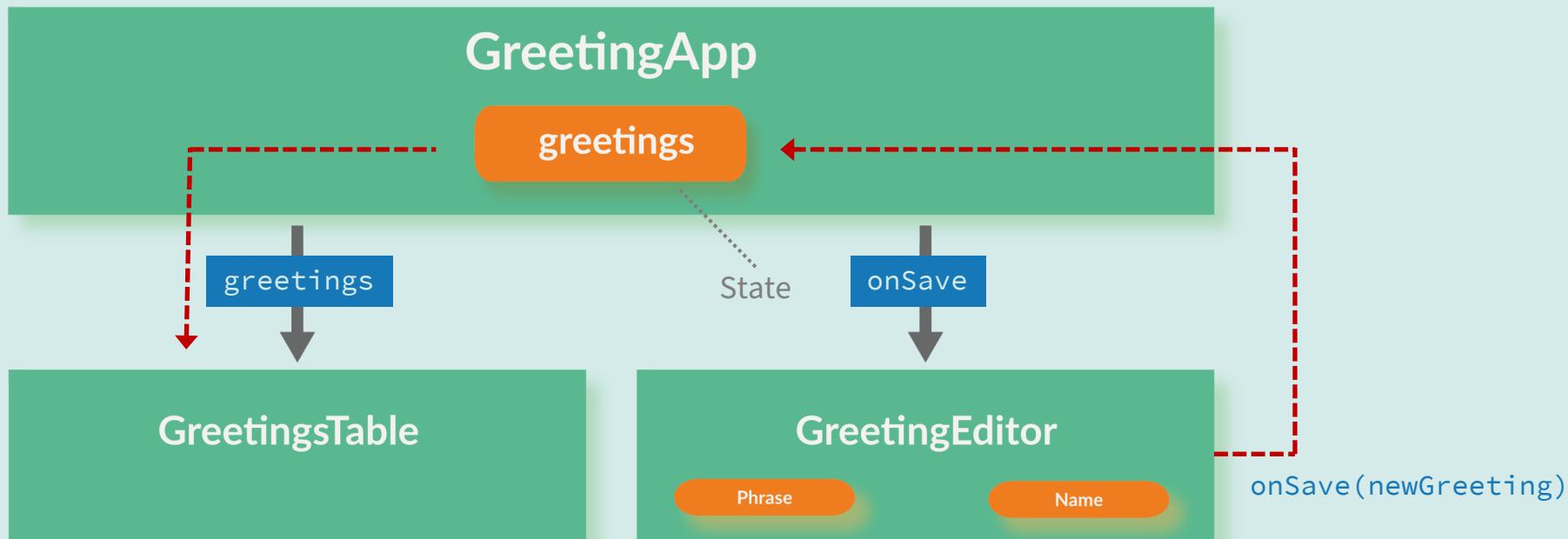
- Von dort per Properties nach unten
- Unterkomponenten informieren nach oben per Callback-Funktion



# HIERARCHIEN VON KOMPONENTEN

## Gemeinsame Daten wandern in gemeinsame Oberkomponenten

- Von dort per Properties nach unten
- Unterkomponenten informieren nach oben per Callback-Funktion
- Anwendung wird neu gerendert, alles konsistent!



# TYPESCRIPT

# Typsichere React-Anwendungen

- out-of-the-box Support für TypeScript

react-training — GreetingEditor.tsx ● https://nilshartmann.net

GreetingEditor.tsx 3, U ●

```
2
3 ✓ interface GreetingDetailProps {
4     initialName: string;
5     initialGreeting: string;
6 }
7
8 ✓ export default function GreetingDetail(props: GreetingDetailProps) {
9     const [name, setName] = React.useState<string>(props.initialName);
10    const [greeting, setGreeting] = React.useState<string>(props.initialGreeting);
11
12 ✓   function reset() {
13     setName(null);
14     setGreeting(props.invalidGreeting);
15   }
16
17   return (
18     <div>
19       <input
20         onChange={onChang
21         value={name}
22         name="name"
23         placeholder="Name"
24       />
any
Cannot find name 'nam'. Did you mean 'name'? ts(2552)
GreetingEditor.tsx(9, 10): 'name' is declared here.
View Problem (xF8) Quick Fix... (#3)
```

OUTPUT PROBLEMS 3 DEBUG CONSOLE

PROBLEMS

- TS GreetingEditor.tsx advanced/steps/5a-redux-hello-world/src 3
  - ✗ Argument of type 'null' is not assignable to parameter of type 'SetStateAction<string>'. ts(2345) [13, 13]
  - ✗ Property 'invalidGreeting' does not exist on type 'GreetingDetailProps'. Did you mean 'initialGreeting'? ts(2551) [14, 23]  
GreetingEditor.tsx[5, 3]: 'initialGreeting' is declared here.
  - ✗ Cannot find name 'nam'. Did you mean 'name'? ts(2552) [21, 16]  
GreetingEditor.tsx[9, 10]: 'name' is declared here.

## **Empfehlung:**

- verwenden!
  - zum Lernen evtl. erst "nur" JS

# MIT REACT LOSLEGEN

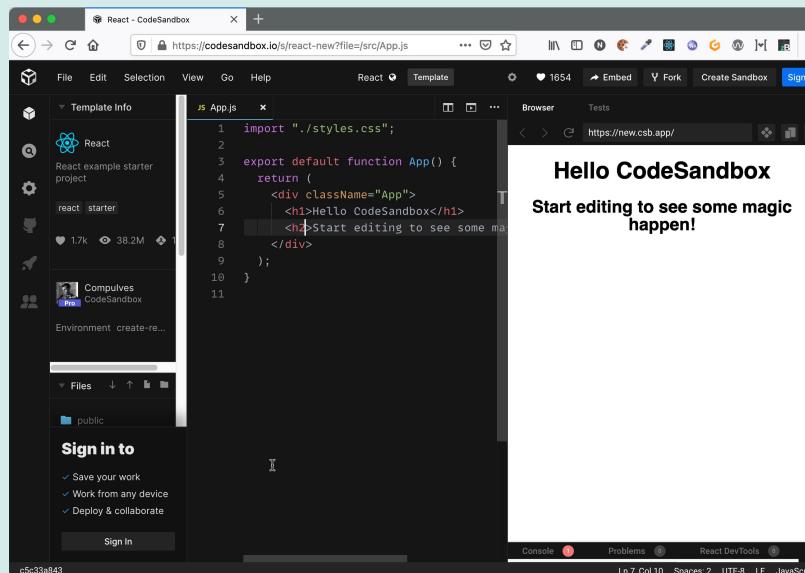
## Erzeugen vom Project mit `create-react-app`

- <https://create-react-app.dev/>
- Offizielles Starter Kit
- Erzeugt fertig konfiguriertes Projekt
  - Build, Linter (statische Code Überprüfung), Test, TypeScript

# MIT REACT LOSLEGEN

## Erzeugen vom Project mit creact-react-app

- <https://create-react-app.dev/>
- Offizielles Starter Kit
- Erzeugt fertig konfiguriertes Projekt
  - Build, Linter (statische Code Überprüfung), Test, TypeScript
- Alternative zum schnellen ausprobieren: <https://codesandbox.io/s/react-new>
  - React App im Browser entwickeln



**React**  
**Ökosystem**

## STYLING (CSS)

## STYLING (CSS)

# Option 1: CSS und CSS Modules

# Support für CSS (Modules) ist eingebaut

Unterstützt wird dabei auch SASS

```
GreetingTable.css      .GreetingTable {  
  (oder .scss)          background-color: orange;  
                      }  
                    }
```

```
GreetingTable.js      import styles from "./GreetingTable.css";  
(oder .tsx)  
function GreetingTable() {  
    return (  
        <table className={styles.GreetingTable}>  
            ...  
        </table>  
    )  
}
```

### Option 2: CSS-in-JS

- Styles werden direkt in JavaScript-Code geschrieben
- Vielleicht konsequenterste Umsetzung der Komponenten Idee
- Ermöglicht Anpassung der Styles zur Laufzeit, z.B. abhängig von Props
- Nicht unumstritten (was machen Leute, die CSS, aber kein JS können?)
- Bekannte Vertreter:
  - Styled Components <https://styled-components.com>
  - Emotion <https://emotion.sh/docs/@emotion/react>

## STYLING (CSS)

### Option 2: CSS-in-JS

- Beispiel: Styled Components

"Reguläre"  
Komponente

```
function RawGreetingTable(props) {  
  return <table className={props.className}> ... </table>;  
}
```

## STYLING (CSS)

### Option 2: CSS-in-JS

- Beispiel: Styled Components

"Reguläre"  
Komponente

```
function RawGreetingTable(props) {  
  return <table className={props.className}> ... </table>;  
}
```

Styling Angaben  
mit dynamischen  
Werten

```
import styled from "styled-components";  
  
const GreetingTable = styled(RawGreetingTable)`  
  background-color: orange  
  font-size: ${props => props.greetings.length > 10 ? "15px" : "20px"}  
`
```

## STYLING (CSS)

### Option 2: CSS-in-JS

- Beispiel: Styled Components

"Reguläre"  
Komponente

```
function RawGreetingTable(props) {  
  return <table className={props.className}> ... </table>;  
}
```

Styling Angaben  
mit dynamischen  
Werten

```
import styled from "styled-components";  
  
const GreetingTable = styled(RawGreetingTable)`  
  background-color: orange  
  font-size: ${props => props.greetings.length > 10 ? "15px" : "20px"}  
`
```

Verwendung  
wie gewohnt

```
function GreetingApp() {  
  return <GreetingTable greetings={...} />  
}
```

## Texte und Daten übersetzen und korrekt formatieren

Typische Vertreter:

- react-intl (<https://formatjs.io/docs/react-intl/>)
- react-i18next (<https://react.i18next.com/>)

Sehr ähnliches Feature-Set

- Einlesen von Sprach-Dateien vom Server
- Übersetzen von Texten inkl. Platzhaltern, Plural etc.
- Formatieren von Datum und Zahlen
- Dynamisches Umstellen des Locales zur Laufzeit

## Empfehlung

- Verwenden, was besser gefällt

# INTERNATIONALISIERUNG

## Beispiel: react-i18next

```
function TableHeader(greetings) {
  const { t } = useTranslation();

  return <h1>
    {t("tableHeader", { count: greetings.length })}
  </h1>
}

// i18n-Datei (en)
{
  "tableHeader": "Showing one greeting",
  "tableHeader_plural": "Showing {{count}} greetings"
}

// i18n-Datei (de)
{
  "tableHeader": "Ein Gruß",
  "tableHeader_plural": "Angezeigt werden {{count}} Grüße"
}
```

# TESTEN VON KOMPONENTEN

## Die Basis: Jest

- <https://jestjs.io/>
- Standard JavaScript Test-Framework

```
function greet() { return "Hello!" }

// entspricht Test-Methode in Junit @Test
test("it should work", () => {

    const greeting = greet();

    // entspricht Assertions
    expect(greeting).toEqual("Hello");

});
```

# TESTEN VON KOMPONENTEN

## React Testing Library

- <https://testing-library.com/docs/react-testing-library/intro/>
- (Unit-)Testen ohne Browser

# TESTEN VON KOMPONENTEN

## React Testing Library

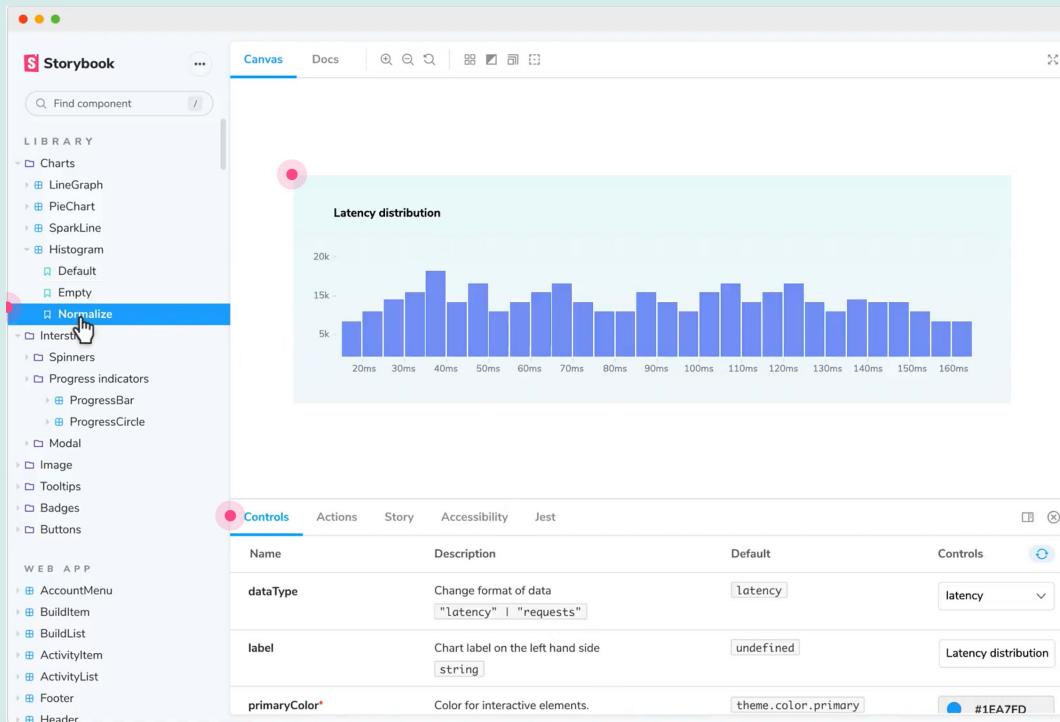
- <https://testing-library.com/docs/react-testing-library/intro/>
- (Unit-)Testen ohne Browser

```
test("it should behave fine", () => {  
  // Mock für Event-Handler  
  const onSaveHandler = jest.fn();  
  
  // Komponente rendern  
  render(  
    <GreetingEditor initialPhrase="Hello" initialGreeting="React"  
      onSave={onSaveHandler} />  
  );  
  
  // Ereignis simulieren  
  fireEvent.click(screen.getByText("Save"));  
  
  // Ergebnis überprüfen  
  expect(onSaveHandler).toHaveBeenCalledWith("Hello", "React");  
});
```

# DOKUMENTATION

## Storybook

- <https://storybook.js.org/>
- Interaktive Präsentation Eurer Komponenten
- Dokumentation und Beispiele direkt beim Source-Code



**Empfehlung:** besonders für Team-übergreifene Komponenten Libs geeignet

## Arbeiten mit URLs

- De-facto-Standard: React Router

# ROUTING

## Arbeiten mit URLs

- React Router <https://reactrouter.com/>
- Wie auf dem Server: Pfade auf Komponenten mappen

```
import {Route, Switch} from "react-router";

function App() {
  return (
    <>
      <h1>Greetings</h1>
      <Switch>
        <Route path="/greet/:greetingId"><GreetingDisplayPage/></Route>
        <Route exact path="/">          <GreetingMaster />      </Route>
        <Route>                      <NotFound />            </Route>
      </Switch>
    </>
  );
}
```

## DATA FETCHING

### Daten laden vom Server

- React macht keine Aussage, wie das geht

### SWR und React Query: vollständige Lösungen für React

- Folgen React Programmiermodell
- Globales Caching, Re-fetching
- Error Handling

# DATA FETCHING

## SWR und React Query: vollständige Lösungen für React

```
import useSWR from "swr";  
  
function GreetingApp() {  
  const { data, error } = useSWR("/api/v1/greetings");  
  
  if (error)  
    return <Error msg="Loading failed" />  
  
  if (!data)  
    return <h1>Greetings loading...</h1>  
  
  return <GreetingTable greetings={data} />  
}
```

Wenn Request Status sich ändert,  
wird Komponente neu gerendert,  
=> neue Daten kommen zurück!

## Empfehlung

- Bei Daten-lastigen Anwendung, React Query oder SWR
- Sonst fetch (Browser API) oder axios

# **GLOBALER ZUSTAND**

# GLOBALER ZUSTAND

- Beispiel: Globale Daten in der Anwendung

Screenshot of a web application titled "Greeting App" showing global state management.

The application interface includes:

- A header bar with tabs: "2-hierarchy" (active), "+", and a search bar.
- A toolbar with icons for navigation, refresh, search, and other application functions.
- A main content area with the title "Greeting App".
- A table showing greetings for "Susi":

Name	Greeting
Susi	Hello!
Susi	How are you?
Susi	Have a nice day
- A note below the table: "(Shown are greetings for Susi. Reset Filter)" with a circled "Susi".
- A blue "Add" button.
- A pie chart titled "2-hierarchy" showing the distribution of greetings by sender:
  - Klaus (blue)
  - Susi (light blue)
  - Felipe (orange)
  - Max (dark orange)
  - Alex (green)
  - Paul (light green)
- A status message: "Showing 3 of 10 Greetings" with a circled "Showing 3 of 10 Greetings".

# GLOBLAER ZUSTAND

- Beispiel: Globale Aktionen und Logik in der Anwendung

2-hierarchy

localhost 80% Search

## Greeting App

Showing 3 of 10 Greetings

Name	Greeting
Susi	Hello!
Susi	How are you?
Susi	Have a nice day

(Shown are greetings for Susi. Reset Filter)

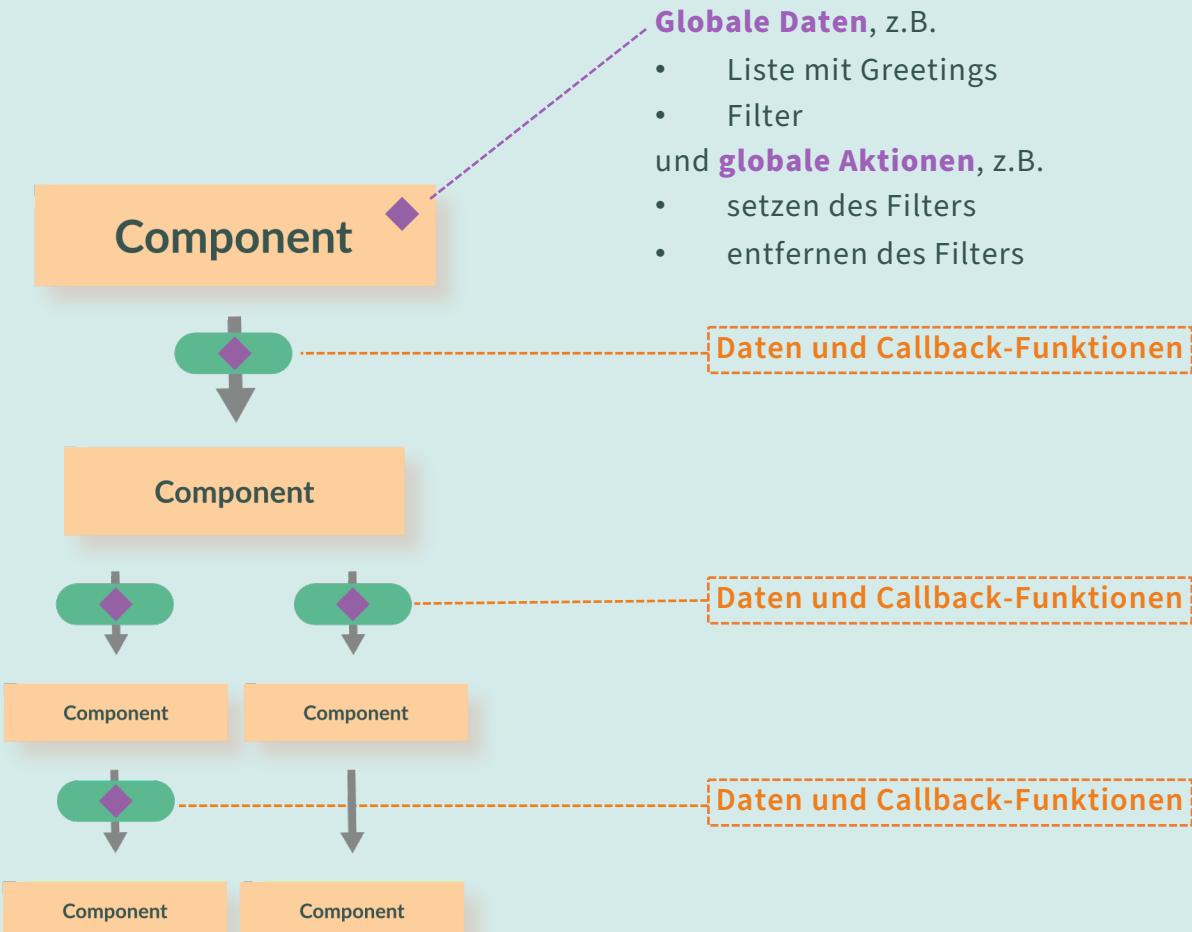
Add

A pie chart titled "2-hierarchy" showing the distribution of greetings. The legend indicates the names corresponding to the colors: Klaus (blue), Susi (light blue), Max (orange), Felipe (orange), Alex (green), and Paul (light green). The slices represent the proportion of greetings for each name.

Name	Proportion
Klaus	~10%
Susi	~35%
Max	~15%
Felipe	~20%
Alex	~5%
Paul	~5%

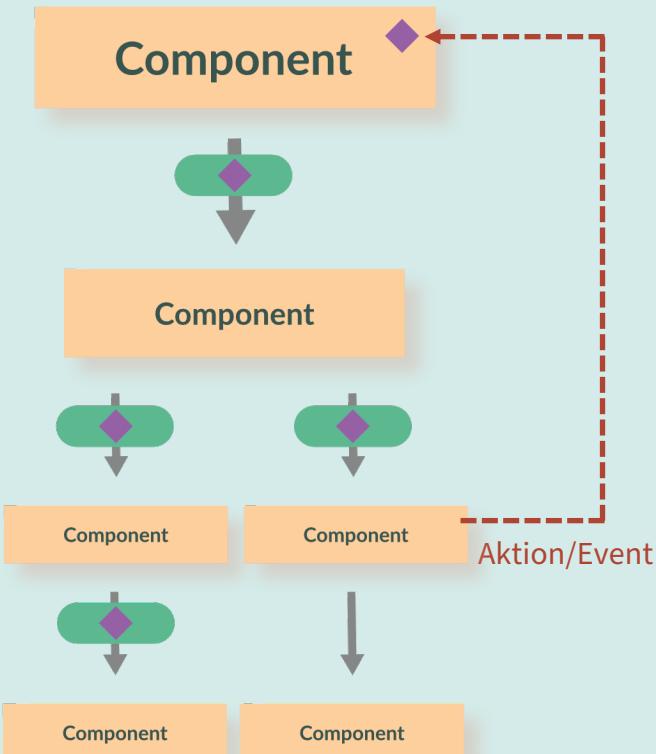
# GLOBALE DATEN

## Der klassische React Weg: Properties durchreichen



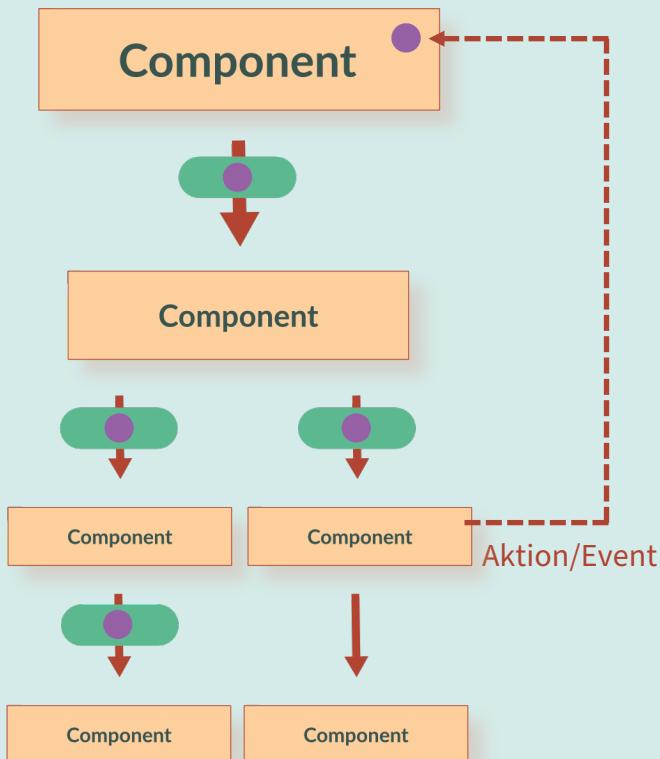
## Der klassische React Weg: Properties durchreichen

Unterkomponenten können Aktionen/Events auslösen, Aufruf von Callback-Funktionen



## Der klassische React Weg: Properties durchreichen

Typischer Datenfluss und Renderzyklus bleibt erhalten: Zustandsänderung -> Rendern



# GLOBALER ZUSTAND

**Externes Statemanagement:** Zustand wandert aus den Komponenten

Prominente Vertreter: [Redux](#) oder [MobX](#)

**Store** hält globale Daten und Logik, Komponenten lesen direkt daraus



## Beispiel: Redux

Zugriff mit Selector auf globalen Store

```
import {useSelector} from "react-redux";

function GreetingMaster() {
  const greetings = useSelector(state => state.greetings);
  const filter = useSelector(state => state.filter);

  return ... // Tabelle mit gefilterten Greetings rendern
}
```

## BEISPIEL: REDUX

### Verarbeitung von Aktionen

Logik ist in Reducer-Funktionen untergebracht

Keine Abhängigkeit auf React oder andere Bibliothek

Verarbeitet Aktionen und liefert neuen Zustand zurück

```
function greetingsReducer(oldState, action) {  
  if (action.type === "removeGreeting") {  
    return {  
      ...oldState,  
      greetings: oldState.greetings.filter(g => g.id !== action.greetingId)  
    }  
  }  
  
  // weitere Actions behandeln...  
}
```

### Globale Daten: Redux, MobX nur in "großen" Anwendungen

- Im Zweifel Redux, weil... "das machen alle"
- Mit Context gibt es eine React-eigene Alternative
  - nicht so mächtig
  - dafür einfache API

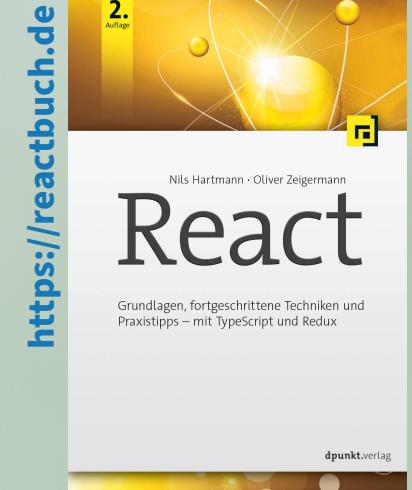
## ZUSAMMENFASSUNG

### Klein und einfach starten...

- React Bordmittel lernen und verwenden
- Für die allermeisten Use-Cases gibt mittlerweile De-facto-Standards

**NILS HARTMANN**

<https://nilshartmann.net>



# vielen Dank!

Slides: <https://react.schule/jax2021-react>

Source Code: <https://github.com/nilshartmann/react-training>

Fragen & Kontakt: [nils@nilshartmann.net](mailto:nils@nilshartmann.net)

@NILSHARTMANN