

Moderne Web-Anwendungen laufen häufig vollständig im Browser ab, um höchstmöglichen Ansprüchen an UI und UX zu genügen. Anwender sollen so den gleichen Bedienkomfort erfahren, wie sie es von Desktop-Anwendungen gewohnt sind. Entwickelt werden diese Single-Page-Anwendungen in JavaScript, häufig mit Hilfe eines spezialisierten Frameworks wie React oder Angular. In diesem Abendvortrag stelle ich die Konzepte und Entwicklung von Single-Page-Anwendungen am Beispiel von React vor. Nach einer Einführung in die Grundlagen dieser Bibliothek sehen wir uns an, welche neuen Anforderungen sich an Code und Architektur von Single-Page-Anwendungen ergeben und wie diese gelöst werden können. Dazu betrachten wir verschiedene Architektur-Muster und werfen einen Blick auf die Sprache TypeScript, die JavaScript um ein Typensystem erweitert. Der Vortrag richtet sich an Entwickler und Architekten, die sich mit der Entwicklung von Single-Page-Anwendungen beschäftigen möchten. Zur Illustration der vorgestellten Konzepte werden JavaScript Code-Beispiele gezeigt, zum Verständnis reichen aber grundsätzliche Programmierkenntnisse aus.

18:00 Uhr Einlass

18:30 Uhr Beginn des ersten Teils des Vortrags von Nils Hartmanni 45 MINUTEN

19:15 Uhr Pause und Networking

19:35 Uhr zweiter Teil des Vortrages 25 Minuten

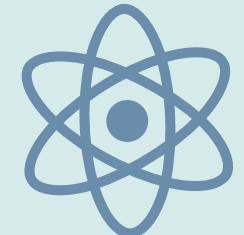
20:00 Uhr Fragen & Antworten

bis 20:45 Uhr Kontakte knüpfen oder einfach den Abend ausklingen lassen bei einem kühlen Getränk

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

SINGLE-PAGE-ANWENDUNGEN MIT

React

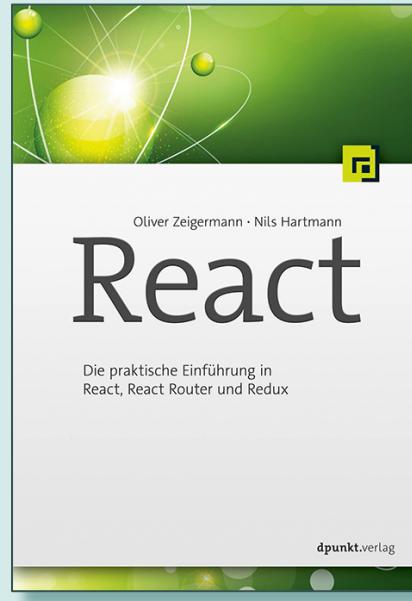


Slides: [TODO](#)

# **NILS HARTMANN**

## **Programmierer aus Hamburg**

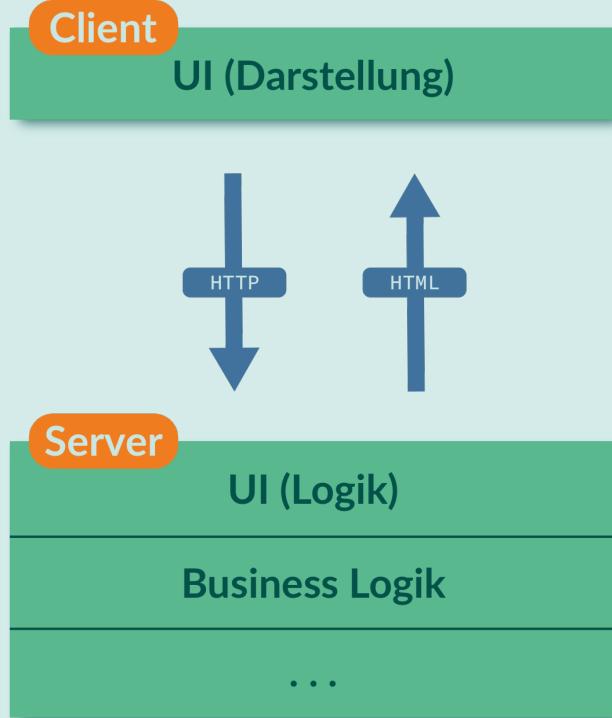
**JavaScript, Java  
Trainings und Workshops**



**@NILSHARTMANN**

# SINGLE PAGE APPLICATIONS

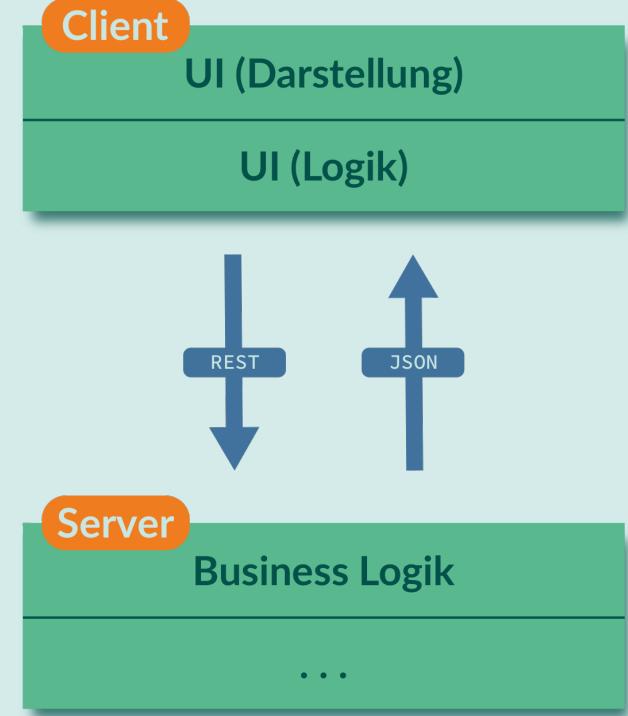
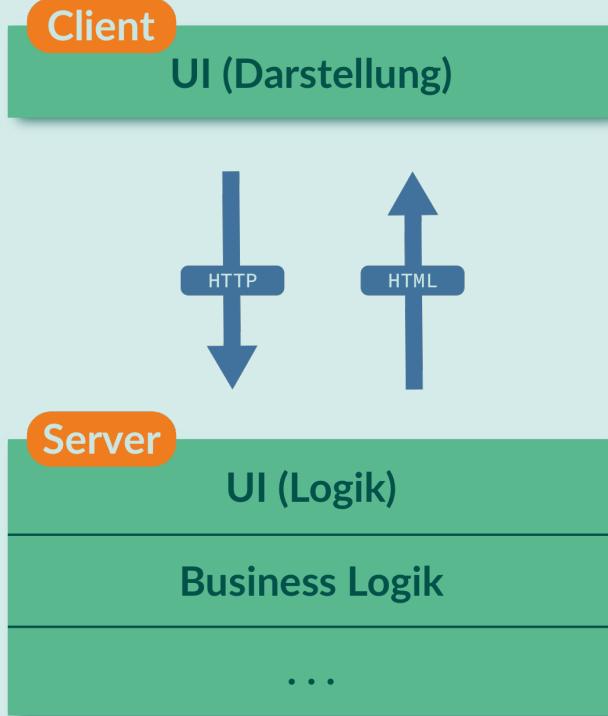
# SINGLE PAGE APPLICATIONS



## Klassische Webanwendung

- JSP, Thymeleaf, JSF
- jQuery

# SINGLE PAGE APPLICATION



## Klassische Webanwendung

- JSP, Thymeleaf, JSF
- jQuery

## Single Page Application

- REST API
- React, Angular, Vue

# Single-Page-Apps

BEISPIELE

# Warum SPAs

- UI/UX wie auf'm Desktop
- Schnelle Antwortzeiten

tm TUS Schwachhausen - SG Aumund-Vegesack Nils

← → ⏪ Sicher | https://www.ticketmaster.de/event/tus-schwachhau... ☆

Kundenservice SPRACHE DE KONTO AUSLOGGEN

**ticketmaster®**

KUNZERTE SPORT KULTUR FREIZEIT MESSEN & AUSSTELLUNGEN COMEDY MEHR ▾

WAS? WANN? WO?

Konzerte Alle Zeiträume Alle Orte LOS

Home > Sport > Fussball > Tus Schwachhausen > 24 Sep 2017, 14:30

 SEP SONNTAG, 14:30  
24 TUS Schwachhausen - SG Aumund-Vegesack i  
2017 Sportanlage TuS Schwachhausen, Bremen

## BESTPLATZBUCHUNG

Preis inkl. Gebühren. Kostenloser Versand. Weitere Infos bei Klick auf "i".

HILFE

Normalpreis	4.75 EUR*	0
ermäßigt i	2.50 EUR*	2

**HTTPS://WWW.TICKETMASTER.DE**

Bibliothek – Playlists

Sicher <https://open.spotify.com/collection/playlists>

PLAYLISTS DEIN MIXTAPE SONGS ALBEN KÜNSTLER

NEUE PLAYLIST

ZULETZT ABGESPIELT

Pro-Pain KÜNSTLER

Pro-Pain — Voice Of... PLAYLIST

Iron Maiden HH 2017 PLAYLIST

Haukur Tomasson PLAYLIST

App installieren

nils\_hartmann

Fliegen Dritte Wahl +

Suchen

Start

Deine Musik

BLACK FLAG Deaf Kenny's DAMAGED NERVOUS BREAKDOWN

Nazi Punks Fuck Off Von its\_diegoo

Dein Mix der Woche Von Spotify

DRITTE WAHL 3 Dritte Wahl DRITTE WAHL DRITTE WAHL

Dritte Wahl Von nils\_hartmann

PINK FLOYD 50 YEARS 50 ESSENTIAL SONGS

Pink Floyd - 50 Years | 50

Molvaer und von Oswald - 1/1

Rage Against The Machine –

0:00 5:21

A screenshot of the Spotify web player interface. The top navigation bar includes 'Bibliothek – Playlists', a search bar, and a user profile 'Nils'. Below the header are tabs for 'PLAYLISTS' (which is green), 'DEIN MIXTAPE', 'SONGS', 'ALBEN', and 'KÜNSTLER'. A large green button labeled 'NEUE PLAYLIST' is centered. On the left, a sidebar shows recently played tracks: 'Pro-Pain' by 'KÜNSTLER', 'Pro-Pain — Voice Of...' (PLAYLIST), 'Iron Maiden HH 2017' (PLAYLIST), and 'Haukur Tomasson' (PLAYLIST). Below this is an 'App installieren' button and a user profile 'nils\_hartmann' with a play button icon. The main content area features several playlists: 'BLACK FLAG' by 'Deaf Kenny's' (with tracks 'DAMAGED' and 'NERVOUS BREAKDOWN'), 'Dein Mix der Woche' by 'Spotify' (with track 'Nazi Punks Fuck Off' by 'its\_diegoo'), 'DRITTE WAHL' by 'nils\_hartmann' (with tracks '3' and 'DRITTE WAHL'), 'PINK FLOYD 50 YEARS 50 ESSENTIAL SONGS' (with track 'Pink Floyd - 50 Years | 50'), 'Molvaer und von Oswald - 1/1' (with track '1/1'), and 'Rage Against The Machine –' (with track 'rage against the machine'). At the bottom, a playback bar shows a progress of 0:00 to 5:21.

[HTTPS://OPEN.SPOTIFY.COM/COLLECTION/PLAYLISTS](https://open.spotify.com/collection/playlists)

Sample File – Figma

Sicher | https://www.figma.com/file/pQLiW7fU1VQwYIJjs2epDl/Sample-File?node-id=0%... Share 100%

DESIGN PROTOTYPE CODE

CONSTRANTS LAYER FILL STROKE EFFECTS EXPORT

Click + to add an export setting

Bring Forward ⌘]

Bring to Front ⌘⌘]

Send Backward ⌘[

Send to Back ⌘[

Group Selection ⌘G

Ungroup Selection ⌘⌘G

Flatten Selection ⌘E

Use as Mask ⌘⌘M

Outline Stroke ⌘⌘O

Create Component ⌘⌘K

Go to Master Component

Reset Instance

Detach Instance ⌘⌘B

Show/Hide Selection ⌘⌘H

Lock/Unlock Selection ⌘⌘L

Flip Horizontal ⌘H

Flip Vertical ⌘V

Copy Style as CSS

Copy as SVG

Copy Style ⌘⌘C

Paste Style ⌘⌘V

Log-In Page

Instructions

- Intro Text
- Step 1
- Step 2
- Step 3
- Step 4
- Step 5
- Step 6
- Step 7
- Step 8

Log-In Page

- Status Bar
- App Info
  - 151 3rd St San Fran...
  - YOUR ART MUSEUM
- Log-In Fields
- Log-In Button

Background Image

Home

Menu

Exhibition

Shop

Share

100%

DESIGN PROTOTYPE CODE

CONSTRANTS

LAYER

FILL

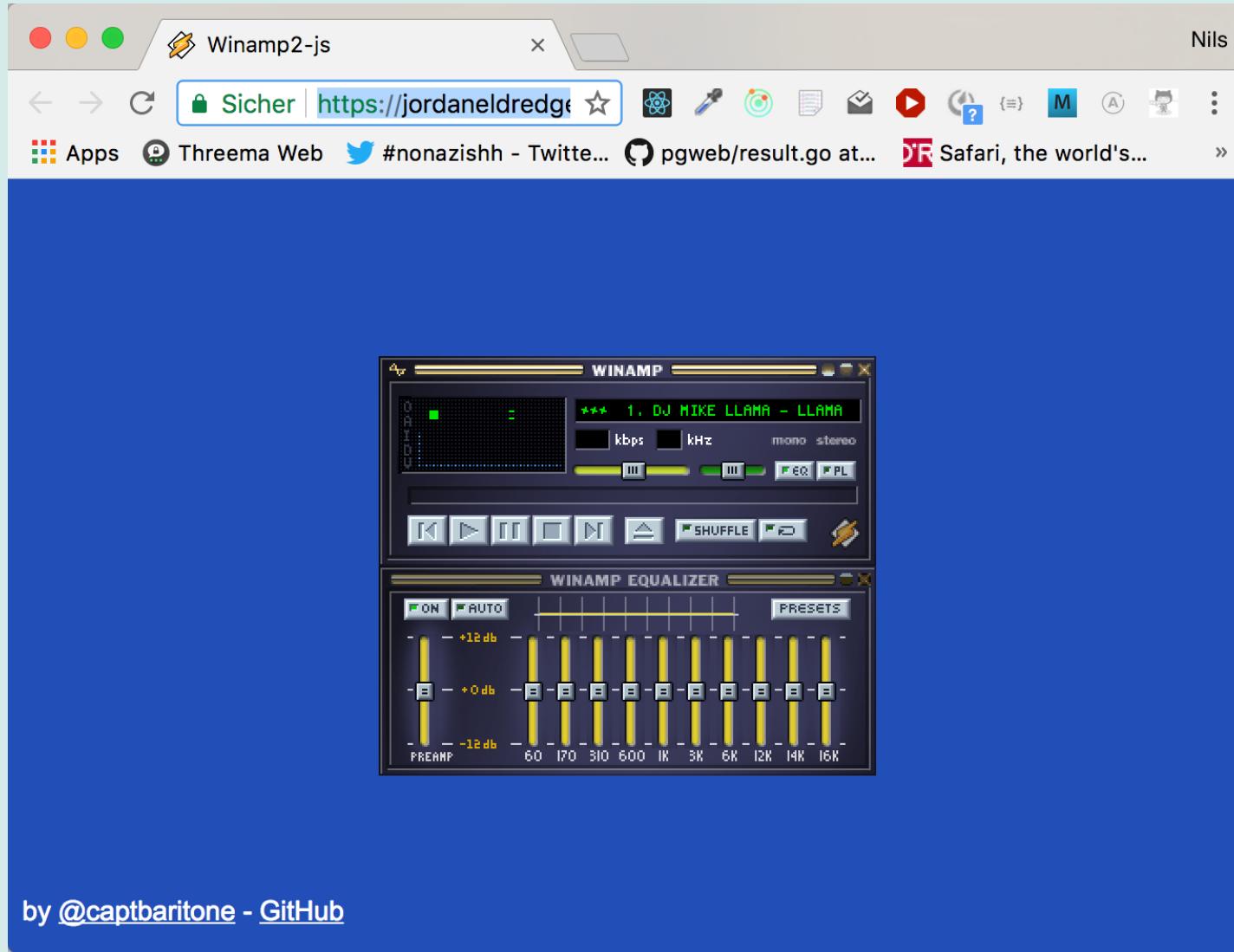
STROKE

EFFECTS

EXPORT

Click + to add an export setting

HTTPS://WWW.FIGMA.COM



**HTTPS://JORDANELDREDGE.COM/PROJECTS/WINAMP2-JS/**

# RETHINKING BEST PRACTICES

Klassische Aufteilung

Logik, Model  
(JS)



View  
(HTML, Template)



Gestaltung  
(CSS)



Aufteilung in Komponenten

Button



Eingabefeld



Label



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

## SEPARATION OF CONCERNES

**Layout**

# Greeting App

**Greeting**

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**

**Counter**

Showing 10 of 10 Greetings

**Chart**

The pie chart displays the proportion of greetings for each name. The segments are labeled with the names: Klaus (blue), Susi (light blue), Max (orange), Felipe (light orange), Alex (green), and Paul (light green). The segments are roughly equal in size.

Name	Proportion
Klaus	~16%
Susi	~16%
Max	~16%
Felipe	~16%
Alex	~16%
Paul	~16%

Legend:

- Klaus (Blue)
- Susi (Light Blue)
- Max (Orange)
- Felipe (Light Orange)
- Alex (Green)
- Paul (Light Green)

## KOMPONENTEN

## Inhalt

### Warum SPAs vs klassische Webanwendung

Klassische Webanwendung hat zu viele Limitationen

Beispiel in unserer Anwendung: hin- und her klicken, nicht offlinefähig etc

Wir wollten bestest UI und UX!

Anwendung soll schnell sein, dh bei klick soll was passieren, nicht erst Server roundtrip

### Wie sehen klassische Webapps aus?

Rendering auf dem Server => Roundtrip

Client-seitige Logik mit jQuery rangefrickelt

### Was macht eine SPA aus

Läuft auf dem Client

In JavaScript programmiert

Logik ist "first-class citizen" und nicht mehr "rangeflanscht"

Komponenten!

## React

JSX

Funktionskomponenten

Klassen

keine große Unterscheidung zwischen Props und State

### Architektur 1: Smart und Dumb Components

#### Externes State Management

Kurze Demo Redux DevTools muss sein

Eventuell nur sehr abstrakt:

Uni-Directional Dataflow als zentrales Element

Store als zentrale Datenbank

Actions

Reducers als "ein Stück Code", der auf Actions reagiert und neuen State liefert

Problem: Wartbarkeit / Architektur

Problem: Wechseln des Uis-Frameworks ("alle zwei wochen gibt es was neues")

Logik wandert aus der UI

Reducer ausschließlich fachlich

Keine Abhängigkeit auf konkretes UI Framework

Architektur-Idee und –Framework

#### These: Back-Button geht nicht

Eventuell zum Schluss als "Zugabe"

Bei jedem Button

## Step 1: Choose new password

A screenshot of a mobile application interface titled "Step 1: Choose new password". At the top is a password input field containing "R I". Below it is a list of five validation checks:

- At least 8 characters long.
- Contains uppercase letters. (highlighted in green)
- Contains lowercase letters.
- Contains numbers.
- Contains punctuation.

Below the list, a message says "4 checks failed". At the bottom is a large button labeled "SET PASSWORD".

```
<PasswordView>
  <PasswordForm>
    <input />
    <CheckLabelList>
      <CheckLabel />
      <CheckLabel />
    </CheckLabelList>
    <Label />
    <Button />
  </PasswordForm>
</PasswordView>
```

PASSWORD FORM WEATHER REPORT CHART EXAMPLE

Step 1: Choose new password

- ✓ At least 8 characters long.
- ✓ Contains uppercase letters.
- ✓ Contains lowercase letters.
- ✓ Contains numbers.
- ✓ Contains punctuation.

4 checks failed

SET PASSWORD

The form is enclosed in a dashed orange border, and the entire application interface is enclosed in a dashed orange border.

<Application>

<Navigation />

<ViewContainer>

<PasswordView>

• • •

• • •

</PasswordView>

</ViewContainer>

</Application>

## React-Komponenten

- bestehen aus Logik und UI
- keine Templatesprache
- werden deklarativ beschrieben
- werden immer komplett gerendert
- können auf dem Server gerendert werden („universal webapps“)



Button



Eingabefeld



Label

- ✓ At least 8 characters long.

- ✓ At least 8 characters long.
- ✓ Contains uppercase letters.

REACT!

- ✓ At least 8 characters long.
- ✓ Contains uppercase letters.

**REACT SCHRITT FÜR SCHRITT**

# DIE JSX SPRACHERWEITERUNG

Anstatt einer Template Sprache: HTML in JavaScript integrieren

- Erlaubt Schreiben von HTML-artigen Ausdrücken im JavaScript-Code
- Wird zu regulärem JavaScript Code kompiliert (z.B. Babel, TypeScript)
- Optional

JSX

```
const name = 'Lemmy';
const greeting = <h1>Hello, {name}</h1>;
```

Übersetztes JavaScript

```
var name = 'Lemmy';
var greeting = React.createElement('h1', null, 'Hello, ', name);
```

# EINE REACT KOMPONENTE: ALS FUNKTION

Komponente CheckLabel

✓ At least 8 characters long.

Komponentenfunktion

```
function CheckLabel() {  
  return <div  
    className="CheckLabel-unchecked">  
      At least 8 characters long.  
    </div>;  
}
```

JSX

# KOMPONENTE EINBINDEN

- ✓ At least 8 characters long.

index.html

```
<html>
  <head> . . .</head>
  <body>
    <div id="mount"></div>
  </body>
  <script src="dist/dist.js"></script>
</html>
```

# KOMPONENTE EINBINDEN

✓ At least 8 characters long.

app.js

```
import React from 'react';
import ReactDOM from 'react-dom';

import CheckLabel from './CheckLabel';

ReactDOM.render(
  <CheckLabel />,
  document.getElementById('mount')
);
```

## KOMPONENTEN: PROPERTIES

✓ At least 8 characters long.

```
{  
  checked: false,  
  label: 'At least 8 characters long.'  
}  
  
function CheckLabel(props) {  
  return <div  
    className=  
      {props.checked?'CheckLabel-checked':'CheckLabel-unchecked'}>  
      {props.label}  
    </div>;  
}
```

# KOMPONENTEN: PROPERTIES

✓ At least 8 characters long.

```
function CheckLabel(props) {  
  . . .  
}
```

```
import PropTypes from 'prop-types';  
  
CheckLabel.propTypes = {  
  label: PropTypes.string.isRequired,  
  checked: PropTypes.bool  
};
```

Properties beschreiben

Überprüfung zur Laufzeit

✖ ► Warning: Failed propType: Required prop `label` was not specified in `CheckLabel`. Check the render method of `CheckLabelList`.  
[main.js:12889](#)

# KOMPONENTEN VERWENDEN

- Komponenten sind **zusammensetzbare**



```
function CheckLabelList() {  
  return <div>  
    <CheckLabel checked={false}  
      label='At least 8 characters long' />  
  
    <CheckLabel checked={true}  
      label='Contains uppercase letters.' />  
  </div>;  
}
```

## BEISPIEL: KOMPONENTENLISTEN

- ✓ At least 8 characters long.
- ✓ Contains uppercase letters.

```
checks: [  
  { checked: false, label: 'At least 8 characters long.' },  
  { checked: true, label: 'Contains uppercase letters' }  
]
```

```
function CheckLabelList(props) {  
  return <div>  
    // . . .  
  </div>;  
}
```

## BEISPIEL: KOMPONENTENLISTEN

- ✓ At least 8 characters long.
- ✓ Contains uppercase letters.

```
checks: [  
  { checked: false, label: 'At least 8 characters long.' },  
  { checked: true, label: 'Contains uppercase letters' }  
]
```

```
function CheckLabelList(props) {  
  return <div>  
    {props.checks.map(c => <CheckLabel  
      label={c.label}  
      checked={c.checked}  
      key={c.label} />)  
  }  
  </div>;  
}
```

# KOMPONENTEN KLASSEN

ECMAScript 2015 Klasse

Properties über Konstruktor  
(optional)

Lifecycle Methoden  
(optional)

Render-Methode (pflicht)

Properties über **props** Objekt

```
class CheckLabelList extends React.Component {  
  constructor(props) {  
    super(props);  
  }  
  
  componentDidMount() { . . . }  
  componentWillReceiveProps() { . . . }  
  shouldComponentUpdate() { . . . }  
  
  render() {  
    return <div>  
      {this.props.checks.map(c => <CheckLabel . . ./>)}  
    </div>;  
  }  
}
```

# ZUSTAND VON KOMPONENTEN

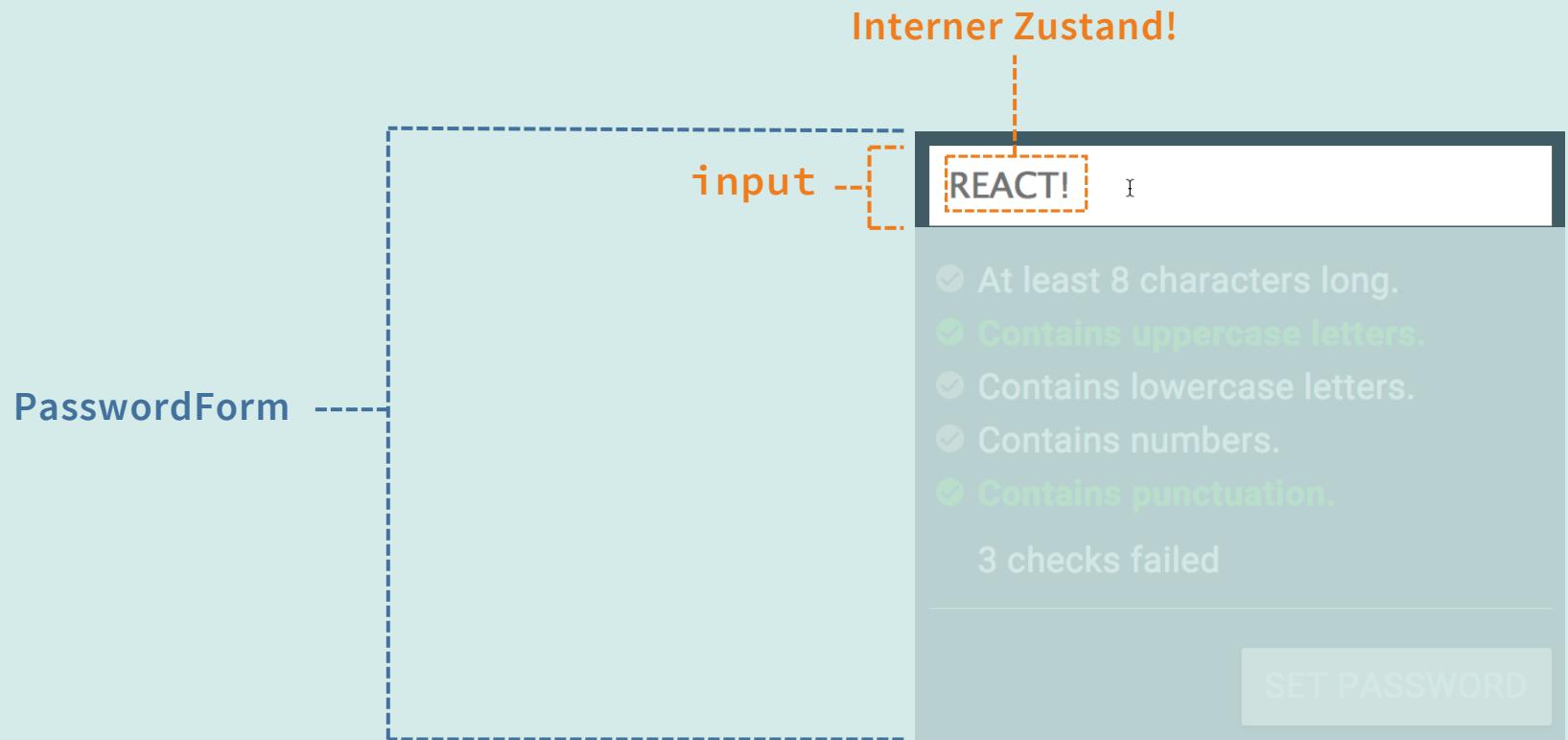
## Zustand („state“): Komponenten-intern

- Beispiel: Inhalt von Eingabefeld, Antwort vom Server
- Objekt mit Key-Value-Paaren
- Zugriff über `this.state / this.setState()`
- Nur in Komponenten-Klassen verfügbar
- `this.setState()` triggert erneutes Rendern
  - auch alle Unterkomponenten
  - Kein 2-Wege-Databinding

## Zum Vergleich: Properties

- Von außen übergeben
- Unveränderlich
- Zugriff über `this.props` (Key-Value-Paare)

## BEISPIEL: EINGABEFELD



## BEISPIEL: EINGABEFELD



1. Input mit Wert aus State befüllen

```
class PasswordForm extends React.Component {  
  render() {  
    return <div>  
      <input  
        value={this.state.password}  
      />  
      . . .  
    </div>;  
  }  
}
```

## BEISPIEL: EINGABEFELD



```
class PasswordForm extends React.Component {  
  render() {  
    return <div>  
      <input  
        value={this.state.password}  
        onChange={e=>this.onPasswordChange(e.target.value)}  
      />  
      . . .  
    </div>;  
  }  
}
```

1. Input mit Wert aus State befüllen

2a. Event Handler registrieren

2b. Event Handler

```
onPasswordChange(newPassword) {
```

```
}
```

```
}
```

# BEISPIEL: EINGABEFELD



```
class PasswordForm extends React.Component {  
  render() {  
    return <div>  
      <input  
        value={this.state.password}  
        onChange={e=>this.onPasswordChange(e.target.value)}  
      />  
      . . .  
    </div>;  
  }  
  
  onPasswordChange(newPassword) {  
    this.setState({password: newPassword});  
  }  
}
```

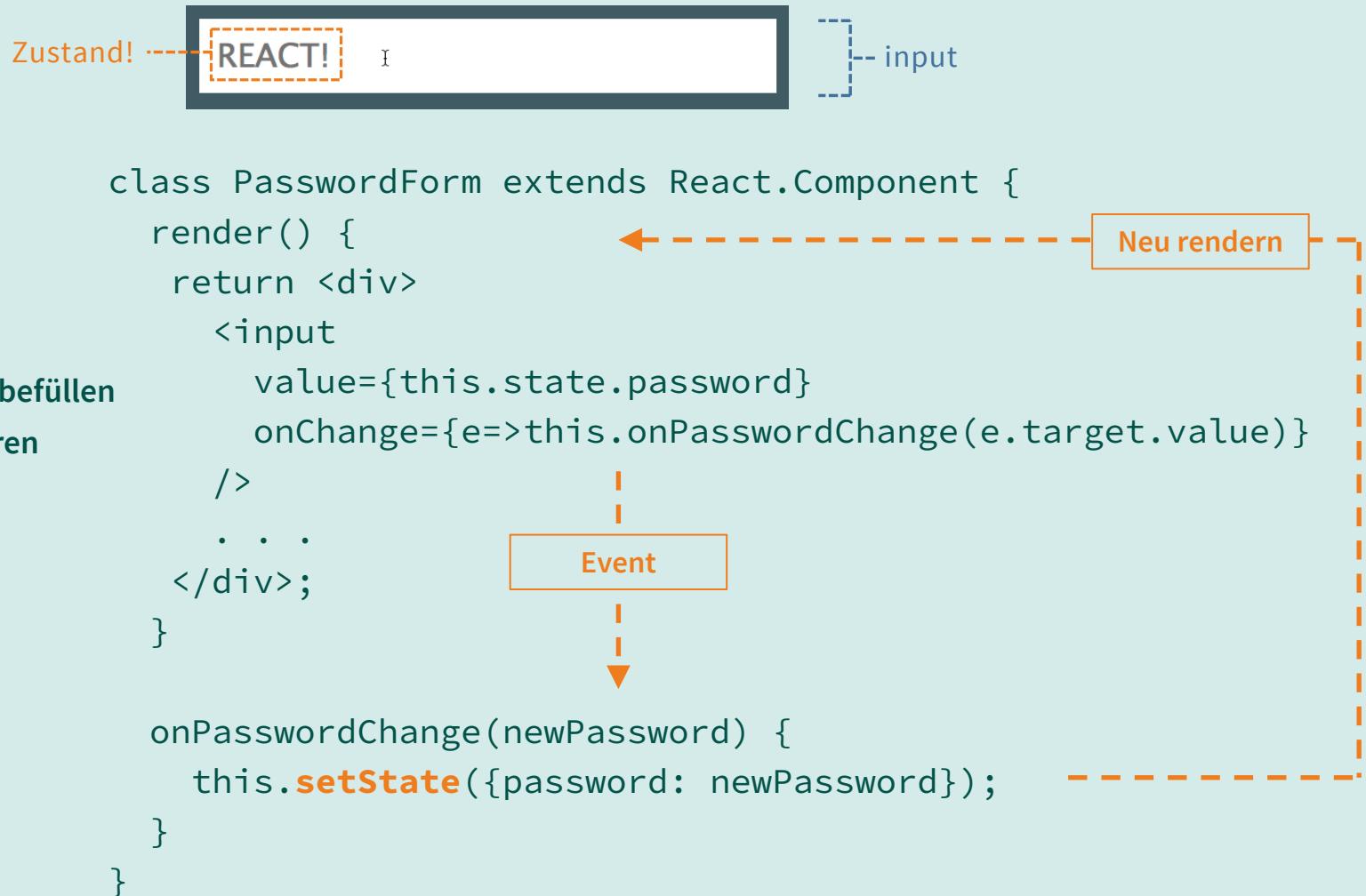
1. Input mit Wert aus State befüllen

2a. Event Handler registrieren

2b. Event Handler

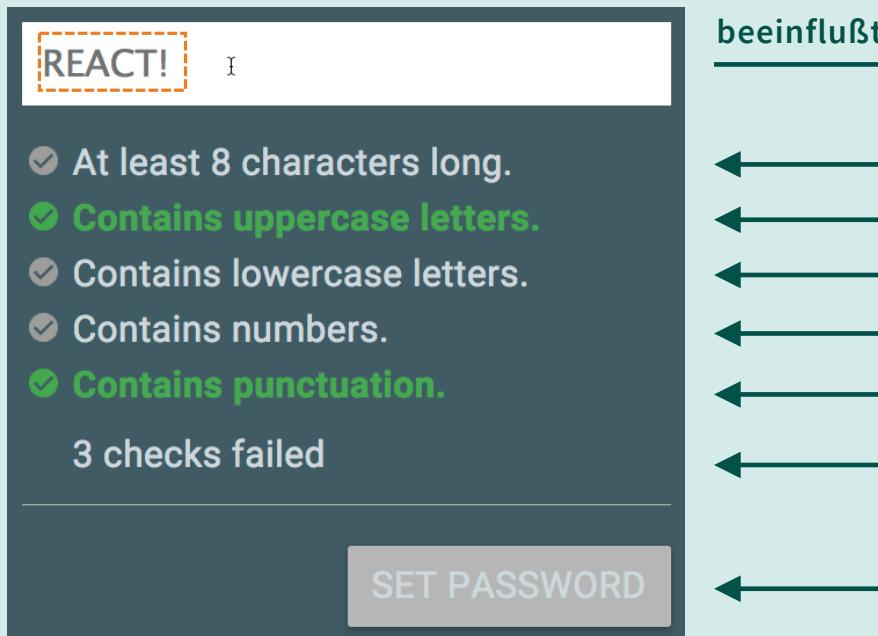
3. Zustand neu setzen

# ZUSTAND: EINGABEFELD

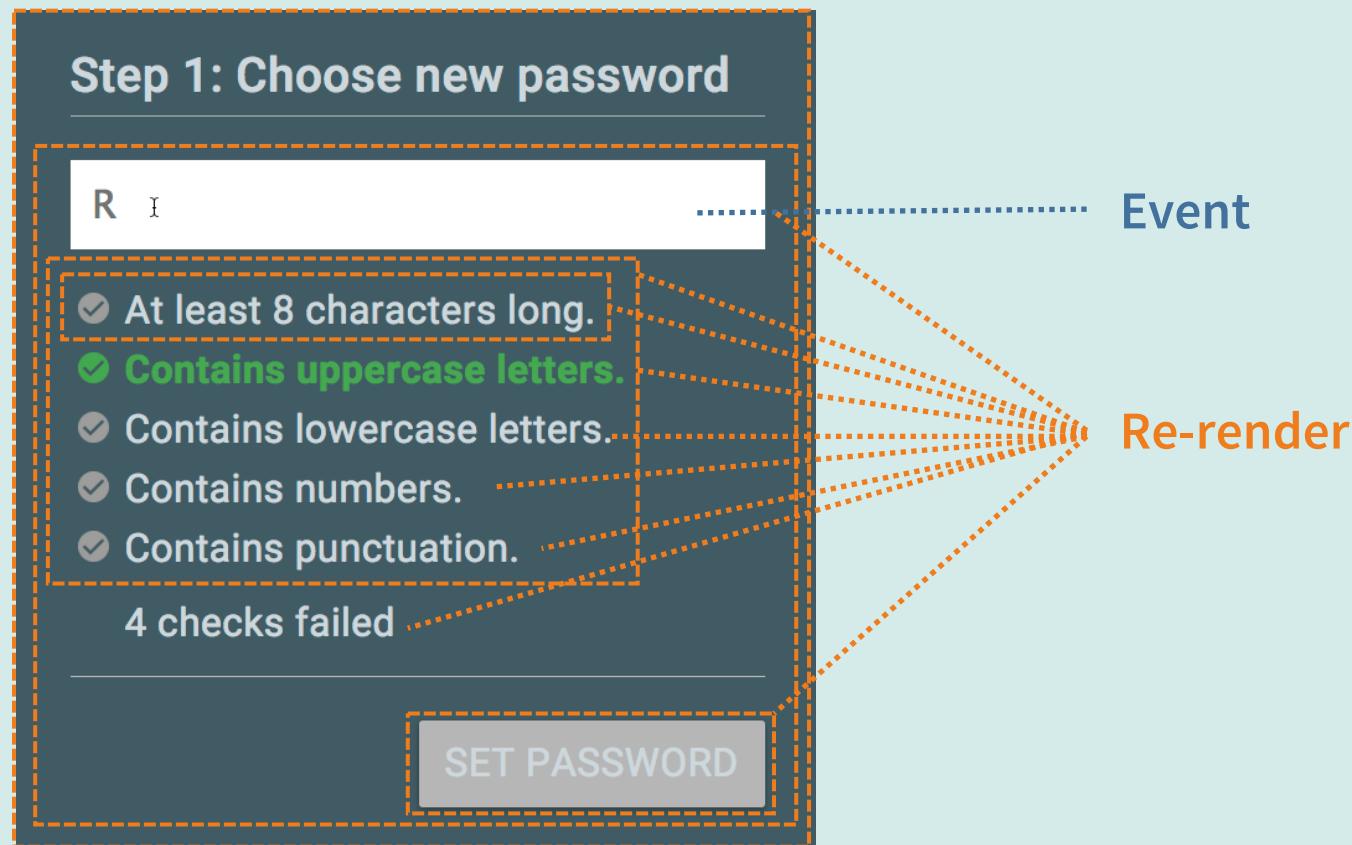


# KONSISTENE UI

## Beispiel: Password Formular

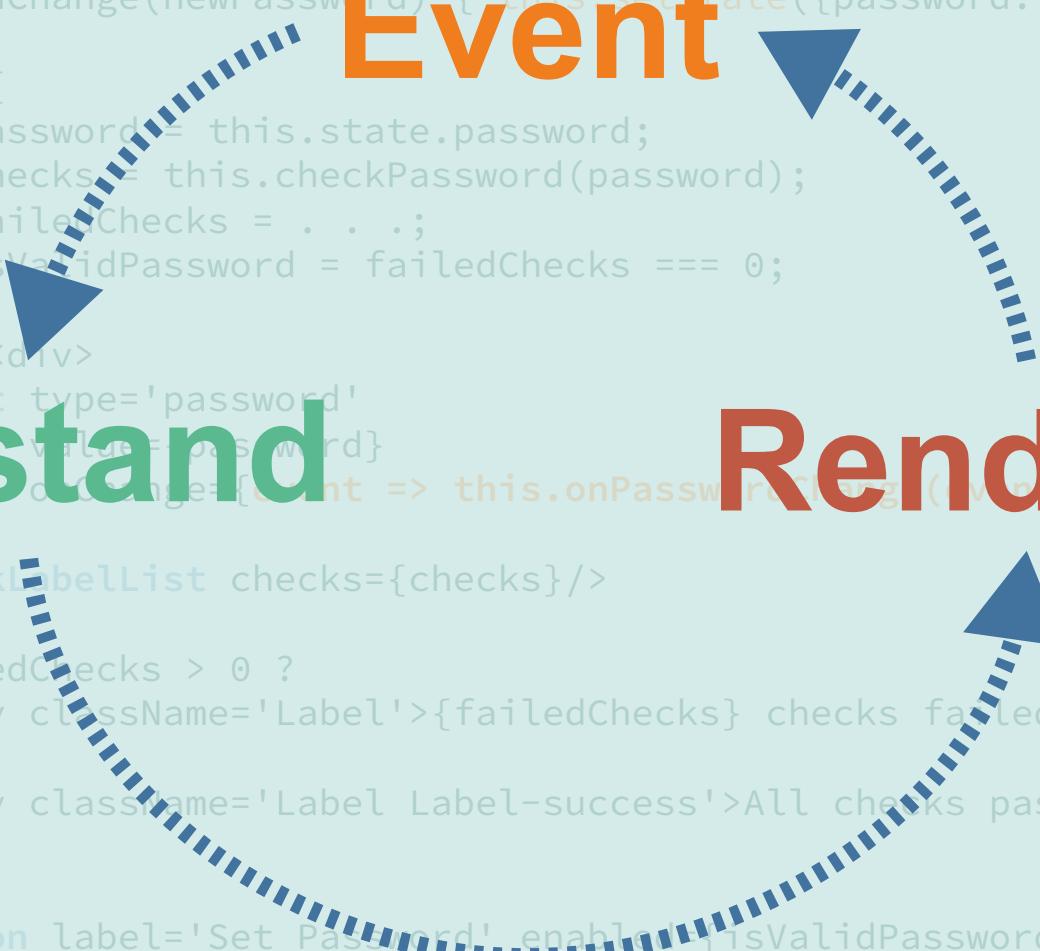


# GANZ EINFACH: ALLES RENDERN



# REACT: UNI DIRECTIONAL DATAFLOW

```
class PasswordForm extends React.Component {  
  onPasswordChange(newPassword) { this.setState({password: newPassword}); }  
  ...  
  render() {  
    const password = this.state.password;  
    const checks = this.checkPassword(password);  
    const failedChecks = ...;  
    const isValidPassword = failedChecks === 0;  
  
    return <div>  
      <input type='password'  
             value={password}  
             onChange={event => this.onPasswordChange(event.target.value)} />  
  
      <CheckLabelList checks={checks}>  
        {failedChecks > 0 ?  
          <div className='Label'>{failedChecks} checks failed</div>  
          :  
          <div className='Label Label-success'>All checks passed!</div>  
        }  
  
      <Button label='Set Password' enabled={isValidPassword} />  
    </div>;  
  }  
}
```



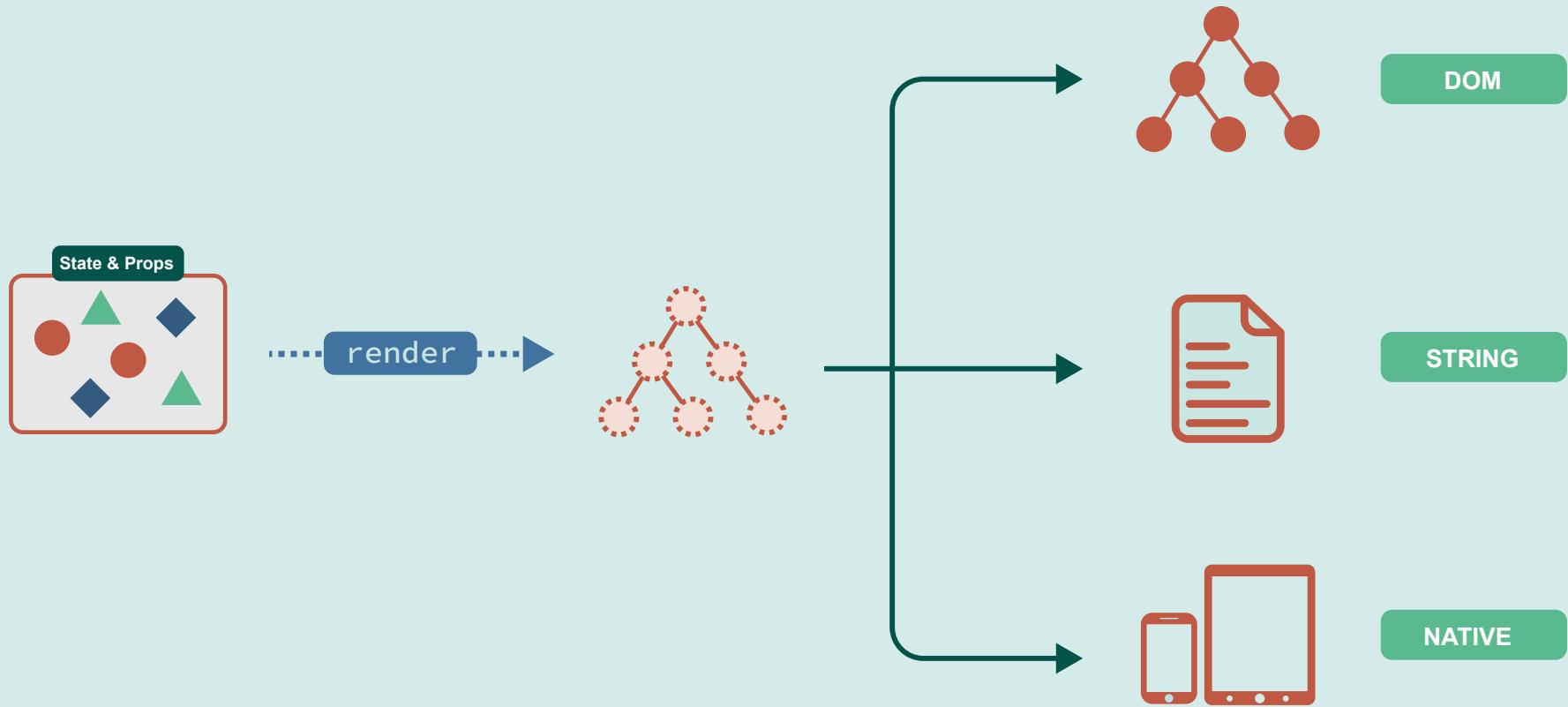
**Event**

**Zustand**

**Rendern**

**RESPOND TO EVENTS & RENDER UI**

# HINTERGRUND: VIRTUAL DOM



## RENDERN IN VERSCHIEDENE FORMATE

## HINTERGRUND: VIRTUAL DOM

### Virtual DOM

- Render-Methode liefert ein **virtuelles** DOM-Objekt zurück
- Trennung von Darstellung (DOM) und Repräsentation (virtueller DOM)

### Vorteile

- Erlaubt performantes neu rendern der Komponente
- Ausgabe in andere Formate (z.B. String) möglich
- Kann auf dem Server gerendert werden (Universal Webapps)
- Kann ohne DOM/Browser getestet werden

## ZUGRIFF AUF DOM-ELEMENTE

### Gearbeitet wird auf *virtuellem* DOM

Zugriff auf *nativen* DOM nötig, z.B.

- Für Integration mit 3rd-Party-Libs (z.B. D3.js)
- Zum Aufruf von Funktionen (z.B. focus())

### Die **ref**-Callback-Funktion

- Kann an Elementen gesetzt werden
- Wird nach dem Rendern aufgerufen
- Übergeben wird Referenz auf natives DOM-Element (oder null)

# BEISPIEL: ZUGRIFF AUF DOM-ELEMENTE

## 1. DOM-Node speichern

```
class PasswordForm extends React.Component {  
  render() {  
    return <div>  
      <input  
        ref={ domNode => this.inputNode = domNode }  
        . . .  
      />  
    </div>;  
  }  
}
```

REACT! I



# BEISPIEL: ZUGRIFF AUF DOM-ELEMENTE



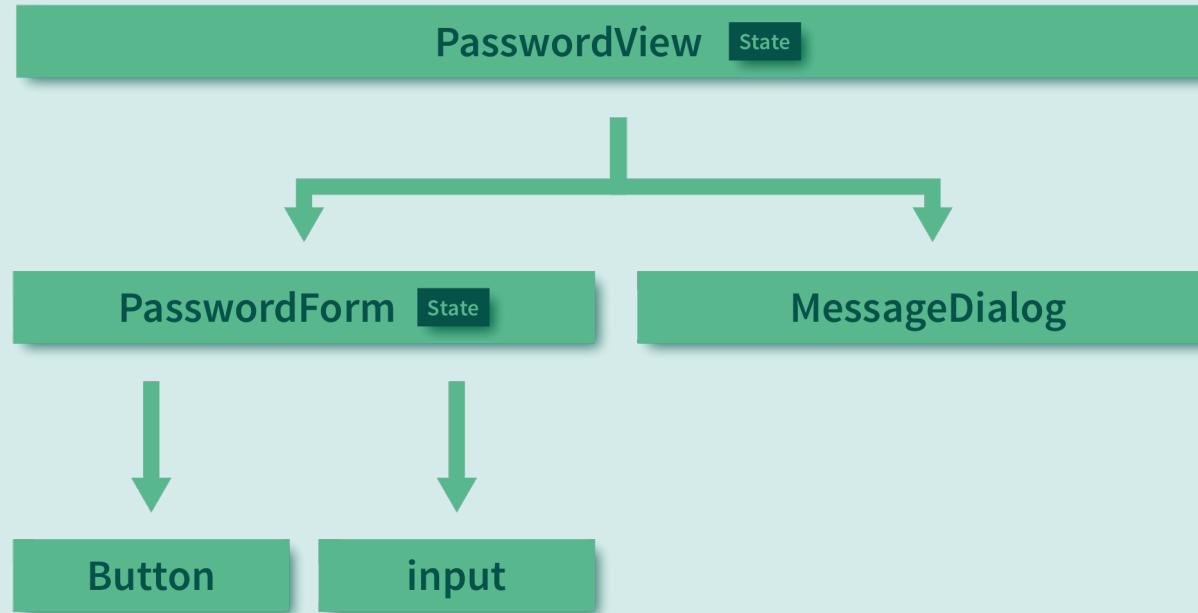
```
class PasswordForm extends React.Component {  
  render() {  
    return <div>  
      <input  
        ref={ domNode => this.inputNode = domNode }  
        . . .  
      />  
    </div>;  
  }  
  
  componentDidMount() {  
    if (this.inputNode) { this.inputNode.focus(); }  
  }  
}
```

1. DOM-Node speichern

2. DOM-Node verwenden

# **TYPISCHE ARCHITEKTUREN**

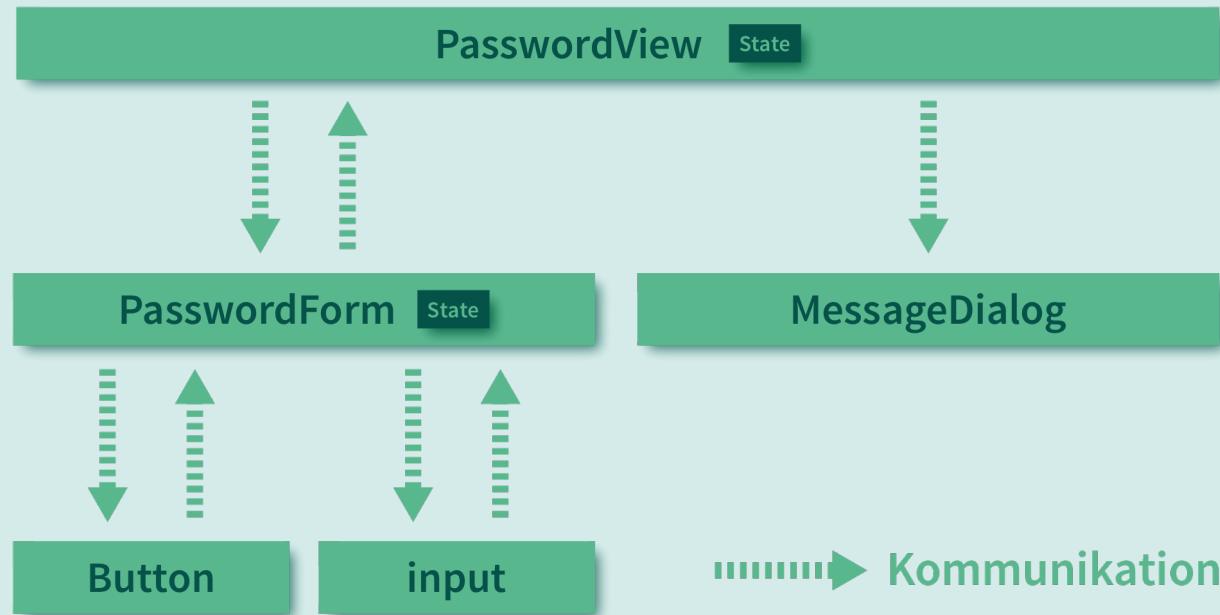
# KOMPONENTENHIERARCHIEN



**Typische React Anwendungen:** Hierarchisch aufgebaut

- State möglichst weit oben („Smart Components“)
- Mehrere Komponenten mit State möglich
  - Beim neu rendern bleibt State erhalten
- Einfache Komponenten, die nur UI enthalten ("Dumb Components")

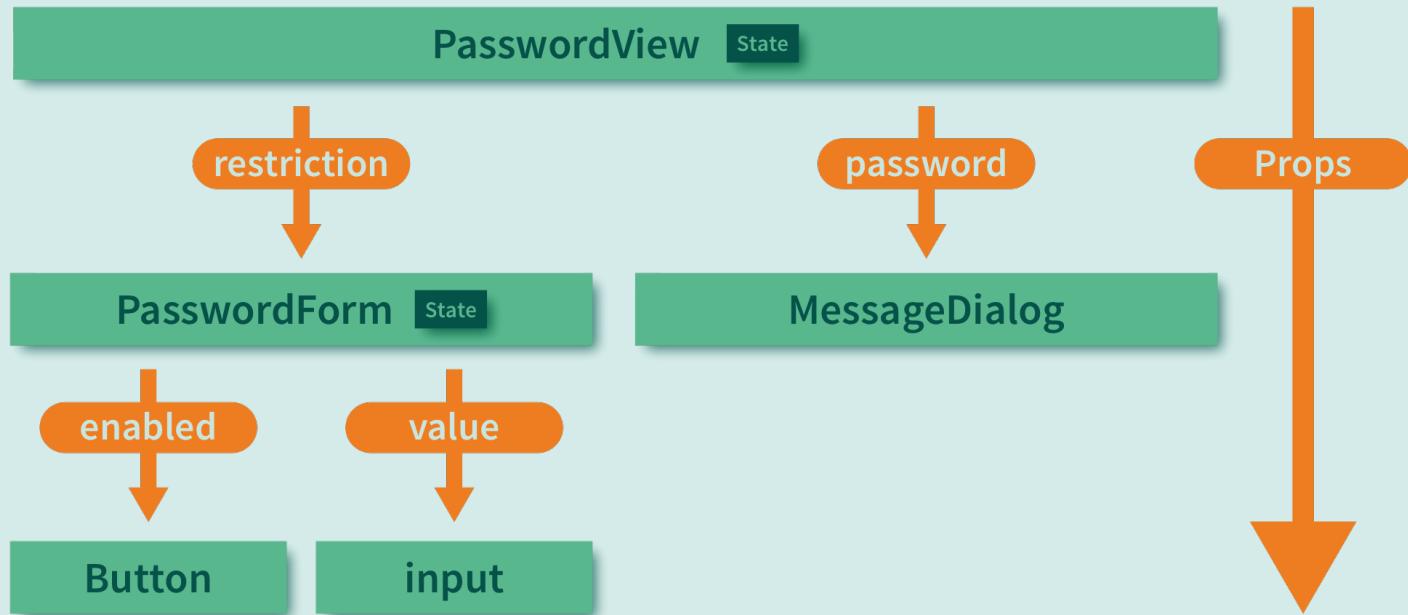
# KOMMUNIKATION ZWISCHEN KOMPONENTEN



**Typische React Anwendungen:** Hierarchisch aufgebaut

- State möglichst weit oben („Smart Components“)
- Mehrere Komponenten mit State möglich
  - Beim neu rendern bleibt State erhalten
- Wie wird kommuniziert?

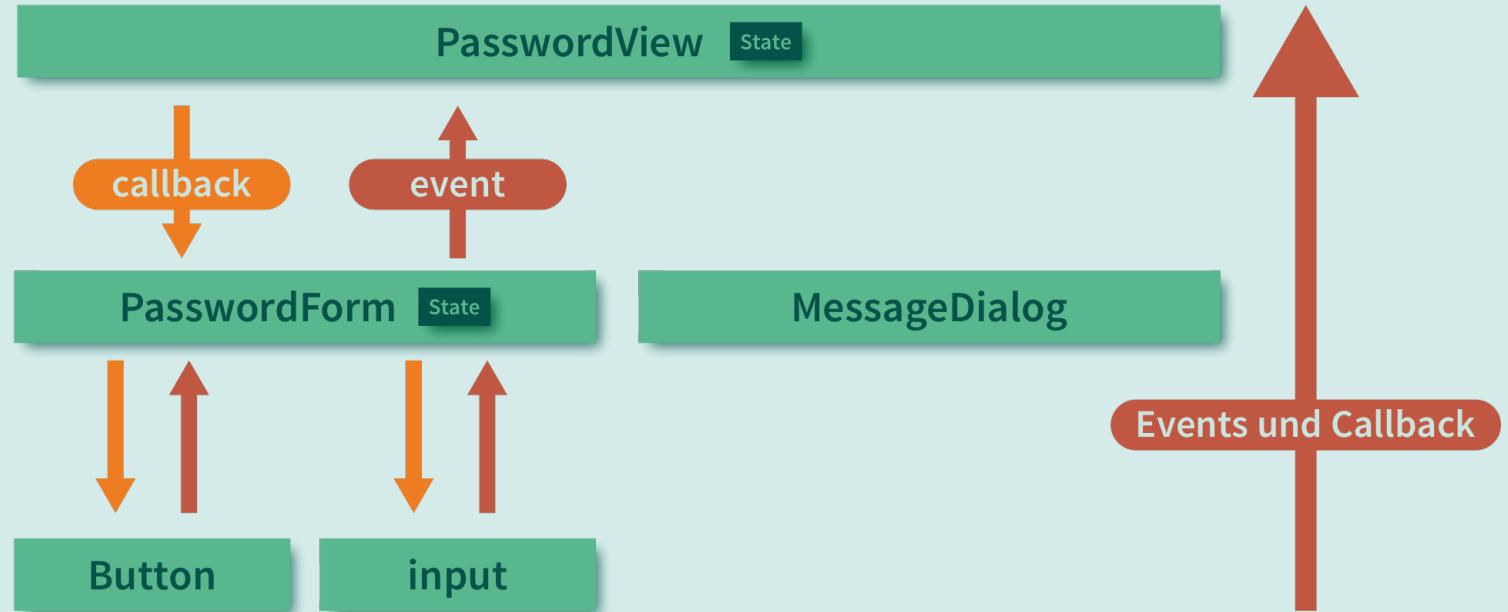
# KOMMUNIKATION: PROPERTIES



Von oben nach unten: Properties

```
<Button enabled={. . .}>Set Password</Button>
```

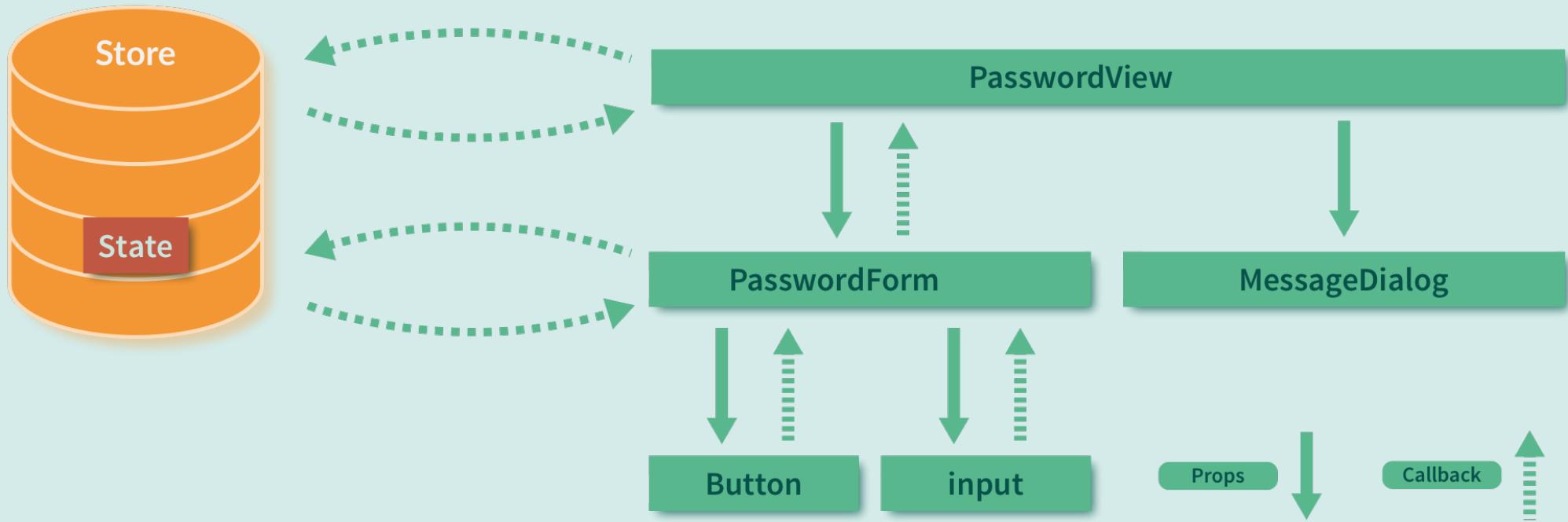
# KOMMUNIKATION: EVENTS



Von unten nach oben: Events und Callbacks

- **Callback-Funktion als Property**
- **Event: Aufruf der Callback-Funktion**

# EXTERNES STATE-MANAGEMENT



Zustand und Logik wird aus den Komponenten ganz raus verschoben

- Prominente Vertreter: **Redux** und **MobX**
- Bessere Testbarkeit (Logik außerhalb von UI Komponenten)
- Geringere Kopplung an das UI-Framework
  - Redux gibt es auch für Angular, Vue, ...

"JavaScript that scales"

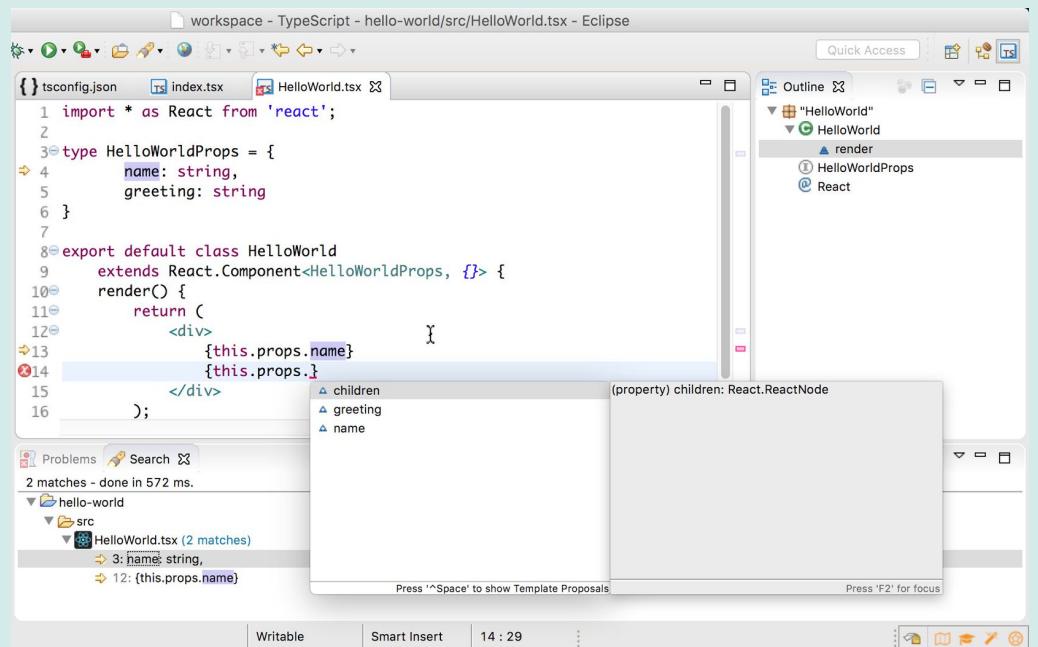
# TypeScript

[HTTP://WWW.TYPESCRIPTLANG.ORG/](http://www.typescriptlang.org/)

# HINTERGRUND: TYPESCRIPT

## TypeScript: Obermenge von JavaScript mit Typ-System

- Gültiger JavaScript-Code auch gültiger TypeScript-Code
- Compiler übersetzt TypeScript in JavaScript-Code
  - Unterstützt auch JSX
- Sehr guter IDE Support
  - z.B. IDEA, Eclipse, VS Code



## Typen verwenden

### Variablen

```
let foo: string; // eingebaute Typen z.B: string, number, boolean  
foo = "yo";  
foo = 10; // Fehler: Type 'number' is not assignable to type 'string'
```

# TYPESCRIPT - SYNTAX

## Typen verwenden

### Variablen

```
let foo: string; // eingebaute Typen z.B: string, number, boolean
```

### Funktionen

```
function sayIt(what: string) {  
    return `Saying: ${what}`;  
}  
sayIt('Klaus'); // OK  
sayIt(10); // Fehler (10 is not a string)
```

# TYPESCRIPT - SYNTAX

## Typen verwenden

### Variablen

```
let foo: string; // eingebaute Typen z.B: string, number, boolean
```

### Funktionen

```
function sayIt(what: string) {  
    return `Saying: ${what}`;  
}
```

**Angabe von Typen ist optional, Typen werden dann abgeleitet:**

```
let result = 7; abgeleiteter Typ: number  
result = sayIt('Lars') // Fehler (abgeleiteter Typ von sayIt: string)
```

## Eigene Typen definieren

```
type Person = {           // Alternativ: interface
    firstName: string,
    lastName: string|null, // nullable Typ ("ein String oder null")
    age?: number          // optionaler Typ
}
```

## Eigene Typen definieren und verwenden

```
type Person = {                      // Alternativ: interface
    firstName: string,
    lastName: string|null,        // nullable Typ ("ein String oder null")
    age?: number                  // optionaler Typ
}

function sayHello(p: Person) {
    console.log(`Hello, ${p.lastName}`);
    p.lastName.toUpperCase(); // Fehler: Object is possibly null
}

sayHello({firstName: 'Klaus', lastName: null}); // OK
sayHello({firstName: 'Klaus', lastName: 777}); // Fehler: lastName kein String
sayHello({firstName: 'Klaus', lastName: 'Mueller', age: 32}); // OK
```

# TYPESCRIPT - SYNTAX

## Generics

```
type Person = { name: string };
type Movie = { title: string };

let persons:Array<Person> = [];
let movies:Array<Movie> = [];

persons.push({name: 'Klaus'});      // OK
movies.push({title: 'Batman'});     // OK
persons.push({title: 'Casablanca'}) // error ('title' not in Person)
```

# TYPESCRIPT - SYNTAX

## Generics

```
type Person = { name: string };
type Movie = { title: string };

let persons:Array<Person> = [];
let movies:Array<Movie> = [];

persons.push({name: 'Klaus'});      // OK
movies.push({title: 'Batman'});     // OK
persons.push({title: 'Casablanca'}) // error ('title' not in Person)
```

# TypeScript

für React-Anwendungen

# TYPESCRIPT UND REACT: PROPERTIES

## Zur Erinnerung: PropTypes in React

✓ At least 8 characters long.

```
function CheckLabel(props) {  
  . . .  
}
```

Properties beschreiben

```
import PropTypes from 'prop-types';  
  
CheckLabel.propTypes = {  
  label: PropTypes.string.isRequired,  
  checked: PropTypes.bool  
};
```

Überprüfung zur Laufzeit

✖ ► Warning: Failed propType: Required prop `label` was not specified in `CheckLabel`. Check the render method of `CheckLabelList`.  
[main.js:12889](#)

## Properties als Typen in TypeScript

✓ At least 8 characters long.

```
function CheckLabel(props: CheckLabelProps) {  
    . . .  
}
```

```
type CheckLabelProps = {  
    label: string,  
    checked?: boolean  
};
```

Typ definieren

Überprüfung zur  
Compile-Zeit  
(auch direkt in der IDE)

```
[ts]  
Type '{ checked: false; }' is not assignable to type 'IntrinsicAttributes & CheckLabelProps'.  
  Type '{ checked: false; }' is not assignable to type 'CheckLabelProps'.  
    Property 'label' is missing in type '{ checked: false; }'.  
(JSX attribute) checked: boolean
```

```
<CheckLabel checked={false} />;
```

## Komponenten-Klassen als Generics

- Typ für Properties und State

1. Typen definieren

```
type PasswordFormProps = {
    restrictions: Restriction[];
    onPasswordSet: (password: string) => void;
};

type PasswordFormState = {
    password: string;
};
```

# TYPESCRIPT UND REACT: PROPERTIES & STATE

## Komponenten-Klassen als Generics

- Typ für Properties und State

### 1. Typen definieren

```
type PasswordFormProps = {
  restrictions: Restriction[];
  onPasswordSet: (password: string) => void;
};
```

```
type PasswordFormState = {
  password: string;
};
```

### 2. Typen als Parameter angeben

```
class PasswordForm extends
  Component<PasswordFormProps, PasswordFormState> {
  . . .
}
```

# TYPESCRIPT UND REACT: PROPERTIES & STATE

## Typische Fehler, die durch TypeScript aufgedeckt werden

### Potentielle Fehler

```
// Properties sind read-only  
this.props.restrictions = null;  
  
// Nur bekannte Properties dürfen verwendet werden  
const x = this.props.not_here;  
  
// State muss vollständig initialisiert werden  
this.state = {};// password fehlt  
  
// this.state darf nur im Konstruktor verwendet werden  
this.state.password = null; // außerhalb des Cstr  
  
// Elemente im State müssen korrekten Typ haben  
this.setState({password: 7}); // 7 is not a string  
  
// Unbekannte Elemente dürfen nicht in den State  
gesetzt werden  
this.setState({notHere: 'invalid'});
```

# Vielen Dank!

<http://bit.ly/bedcon-react>

# Fragen?

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