

NILS HARTMANN

MODERNE WEB-ANWENDUNGEN MIT

React



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

NILS HARTMANN

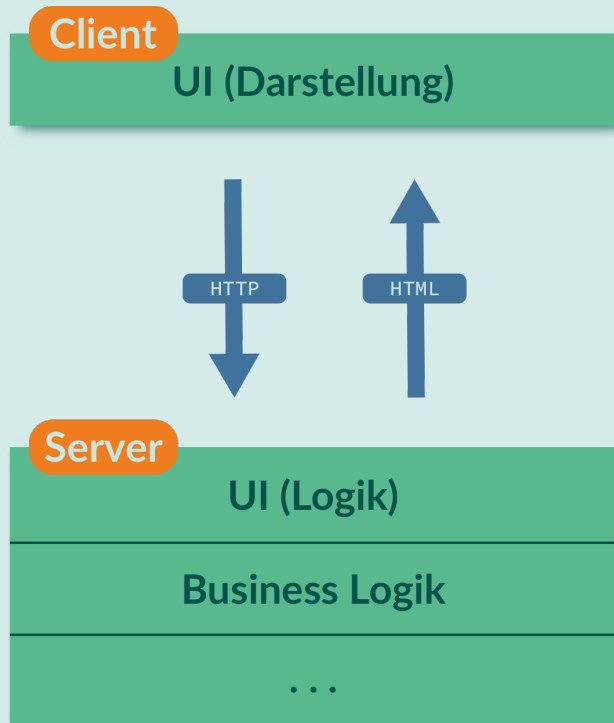
Programmierer aus Hamburg

Java
JavaScript, TypeScript
Trainings und Workshops

<https://reactbuch.de>

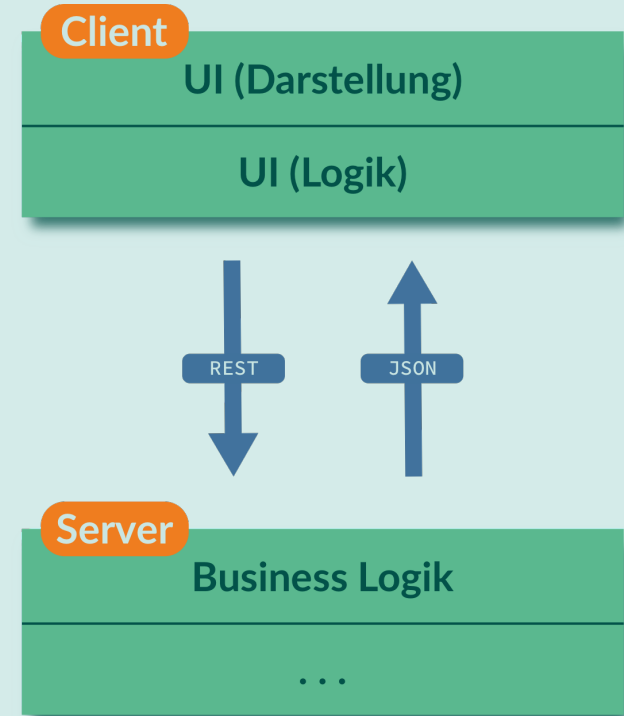
@NILSHARTMANN

HINTERGRUND: SINGLE PAGE APPLICATIONS



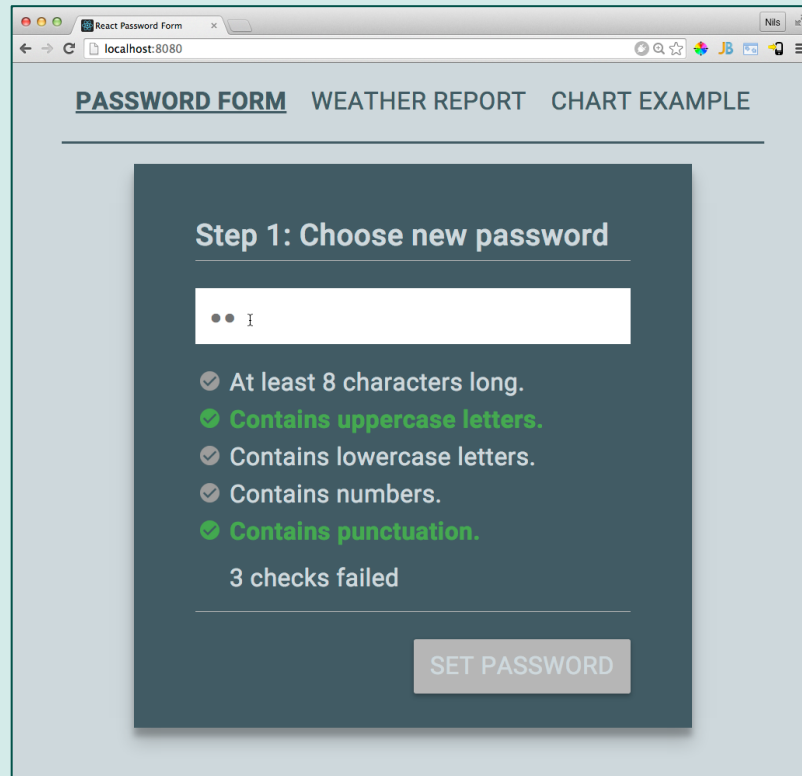
Klassische Webanwendung

- JSP, Thymeleaf, JSF
- jQuery



Single Page Application

- REST API
- React, Angular, Vue



<https://github.com/nilshartmann/react-example-app/tree/typescript>

BEISPIEL ANWENDUNG

Step 1: Choose new password

R I

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

4 checks failed

SET PASSWORD

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

RETHINKING BEST PRACTICES

Klassische Aufteilung

Logik, Model
(JS)



View
(HTML, Template)



Gestaltung
(CSS)



Aufteilung in Komponenten



Button



Eingabefeld



Label

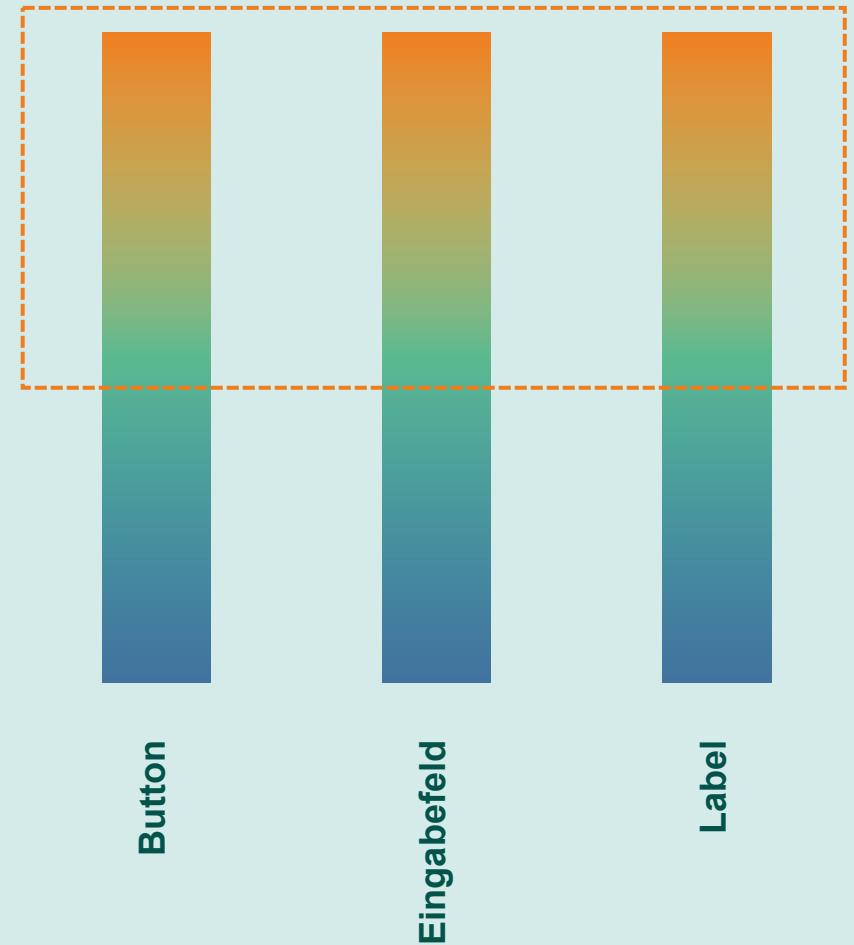
Grafik Inspiriert von: https://pbs.twimg.com/media/DCXJ_tjXoAAoBbu.jpg

SEPARATION OF CONCERNS

React-Komponenten

- bestehen aus **Logik und UI**
 - kein CSS
- **keine Templatesprache**
- werden **deklarativ** beschrieben
- werden immer **komplett gerendert**

- können auf dem **Server gerendert** werden („universal webapps“)



✓ At least 8 characters long.

✓ At least 8 characters long.

✓ **Contains uppercase letters.**

REACT!

✓ At least 8 characters long.

✓ **Contains uppercase letters.**

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

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.

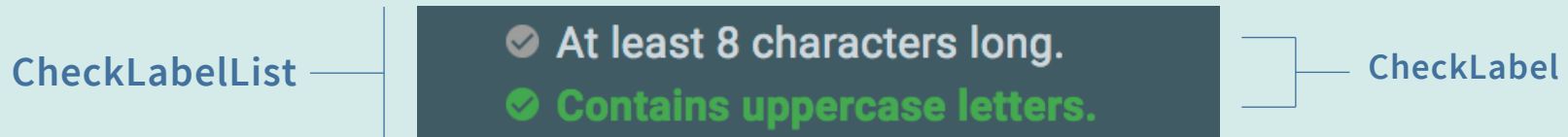
```
{  
  checked: false,  
  label: 'At least 8 characters long.'  
}
```



```
function CheckLabel(props) {  
  return <div  
    className=  
    {props.checked? 'CheckLabel-checked' : 'CheckLabel-unchecked'}>  
    {props.label}  
  </div>;  
}
```

KOMPONENTEN VERWENDEN

- Komponenten sind **zusammensetzbar**



```
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

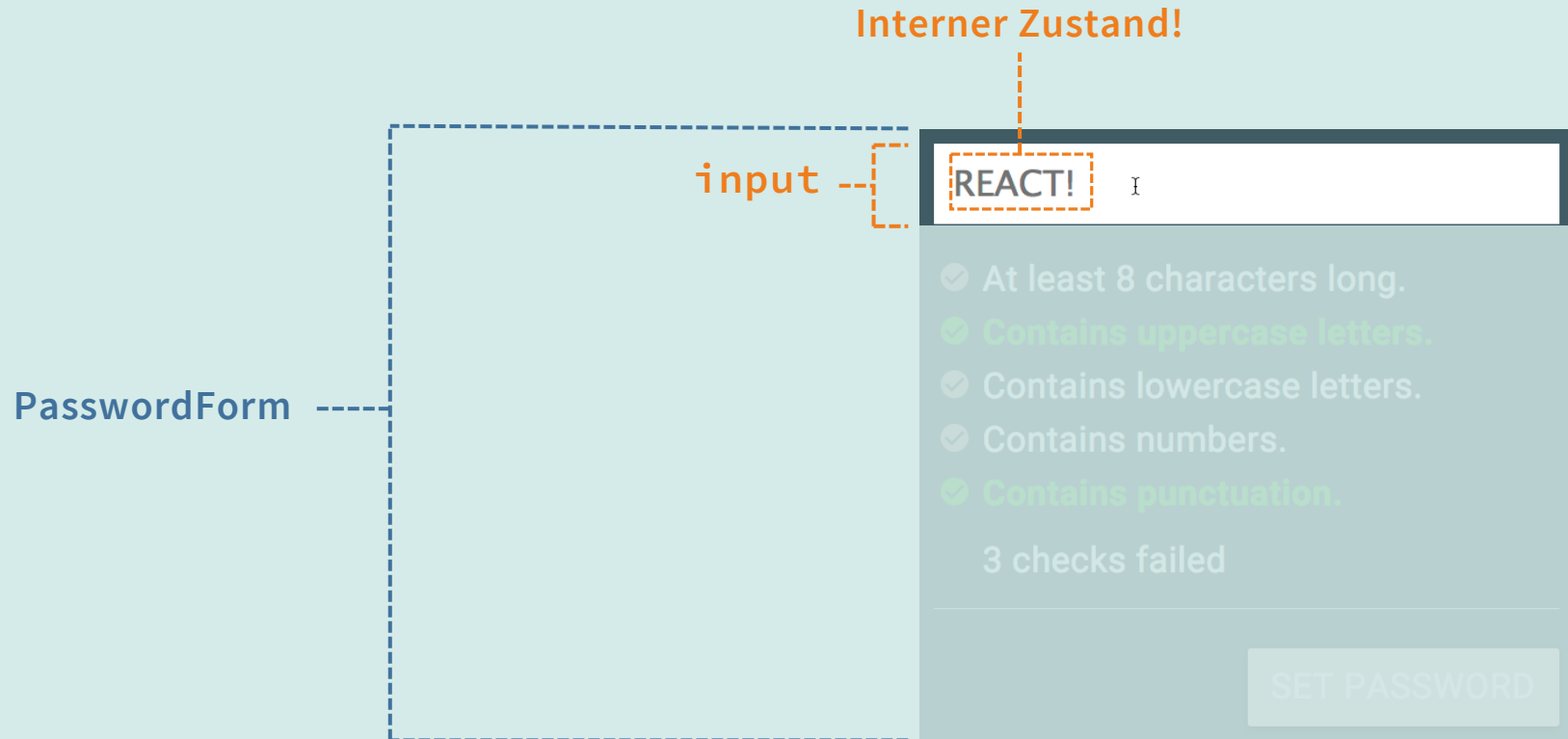
Zustand („state“): Komponenten-intern

- 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



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

1. Input mit Wert aus State befüllen

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) {  
  
  }  
}
```

1. Input mit Wert aus State befüllen

2a. Event Handler registrieren

2b. Event Handler

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



```
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

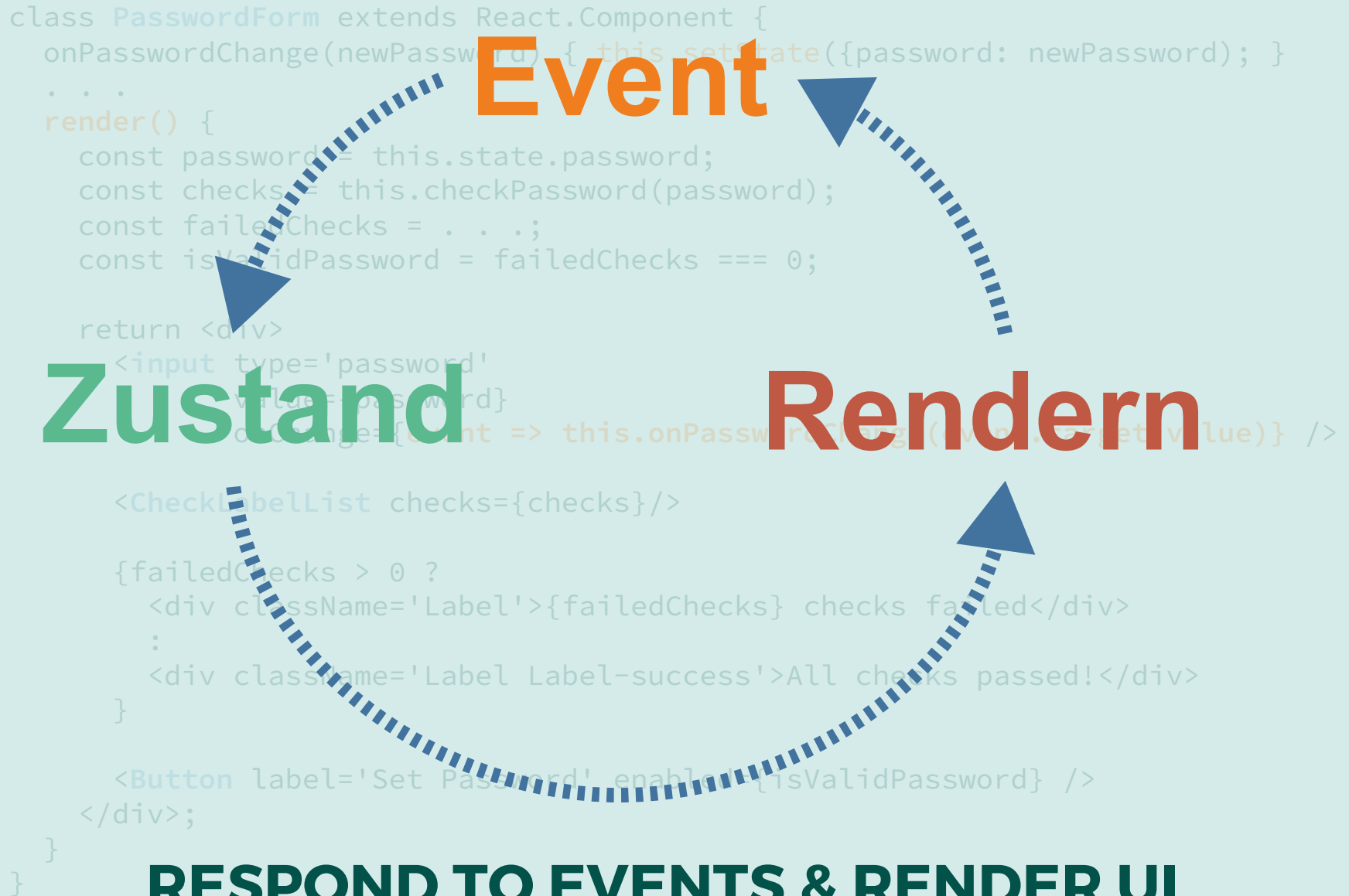
3. Zustand neu setzen

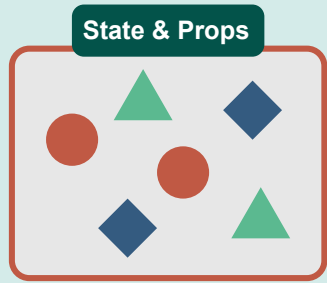
Event

Neu rendern



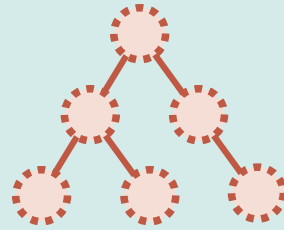
REACT: UNI DIRECTIONAL DATAFLOW



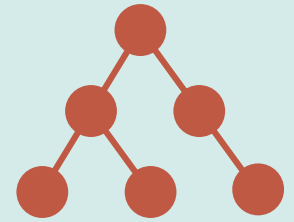


Komponente

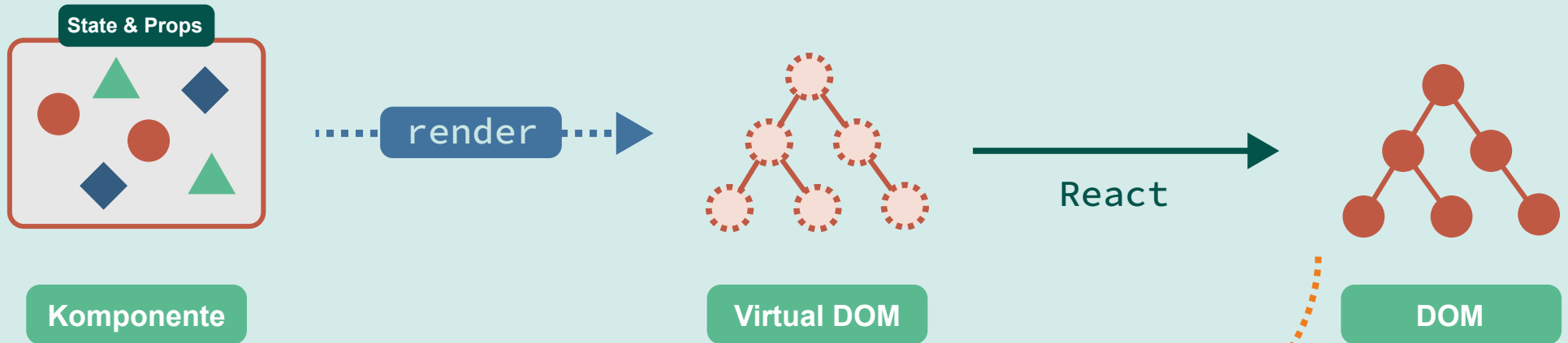
render



React



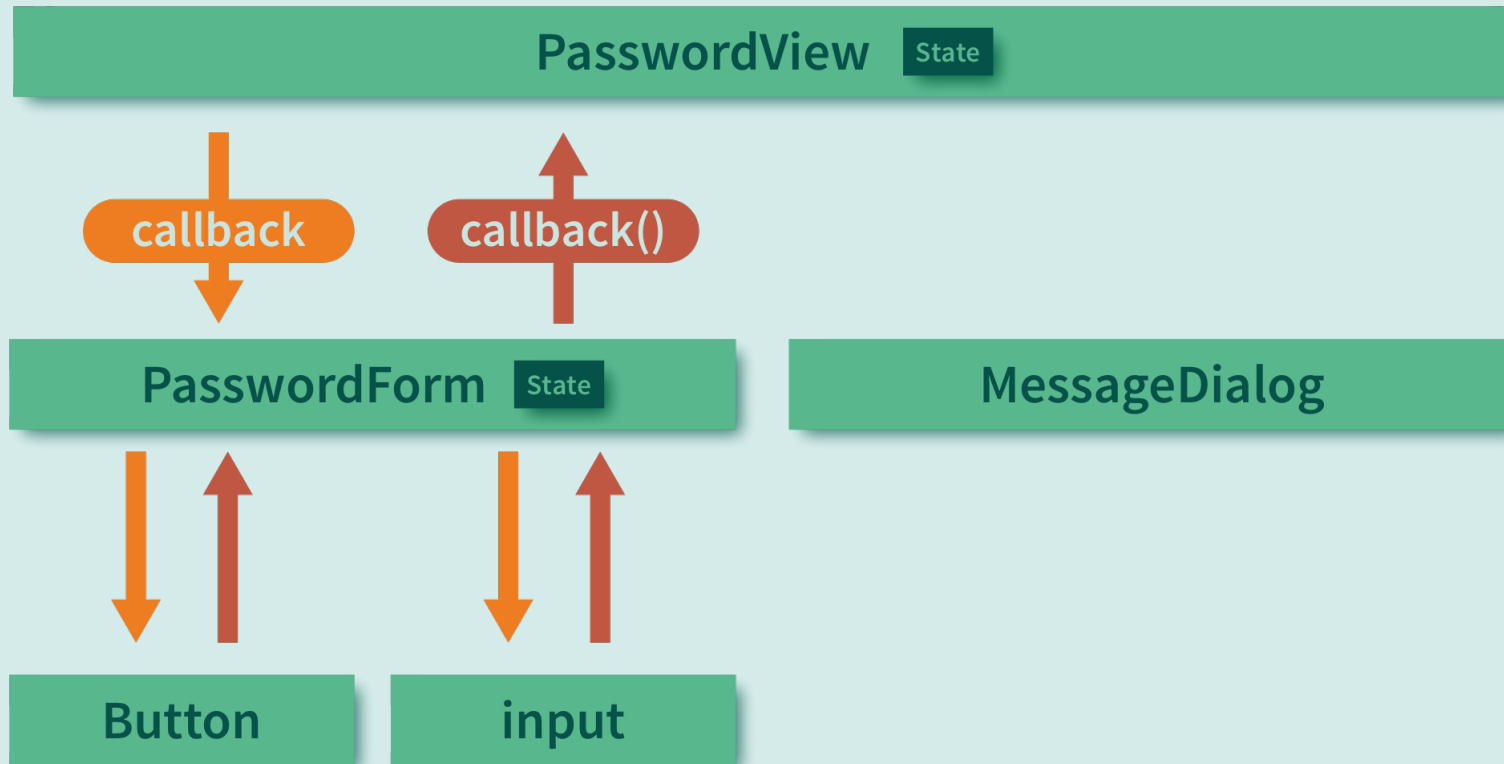
VIRTUAL DOM



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

ZUGRIFF AUF NATIVE DOM ELEMENTE

KOMMUNIKATION



Kommunikation zwischen Komponenten: Callbacks

PasswordForm

Step 1: Choose new password

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

3 checks failed

TypeScript

by Example

PASSWORD FORM

Vielen Dank!

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

Fragen?

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

Anhang

"JavaScript that scales"

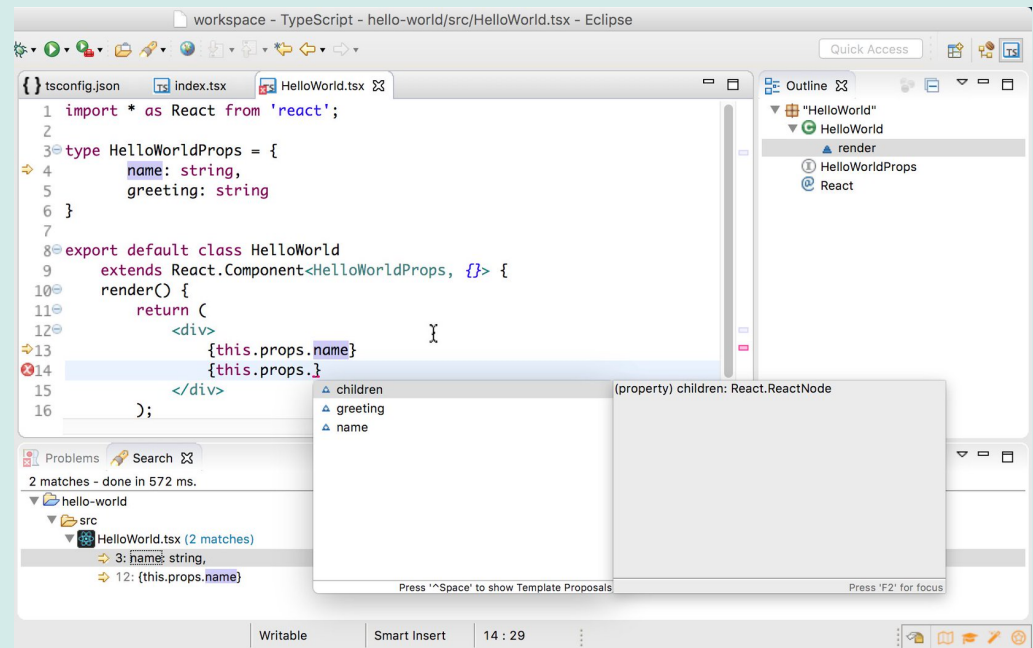
TypeScript

[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'
```

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

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

TYPESCRIPT - SYNTAX

Eigene Typen definieren und verwenden

```
interface Person { // Alternativ: type
  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

für React-Anwendungen

TYPESCRIPT UND REACT: PROPERTIES

Properties als Typen in TypeScript

✓ At least 8 characters long.

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

Typ definieren

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

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

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

```
interface PasswordFormState {  
  password?: string;  
};
```

Komponenten-Klassen als Generics

- Typ für Properties und State

1. Typen definieren

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

```
interface PasswordFormState = {  
  password?: string;  
};
```

2. Typen als Parameter angeben

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

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'});
```