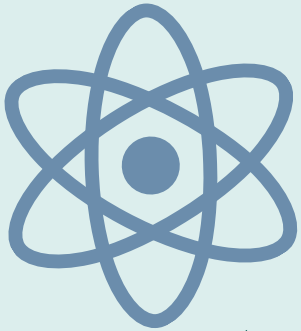
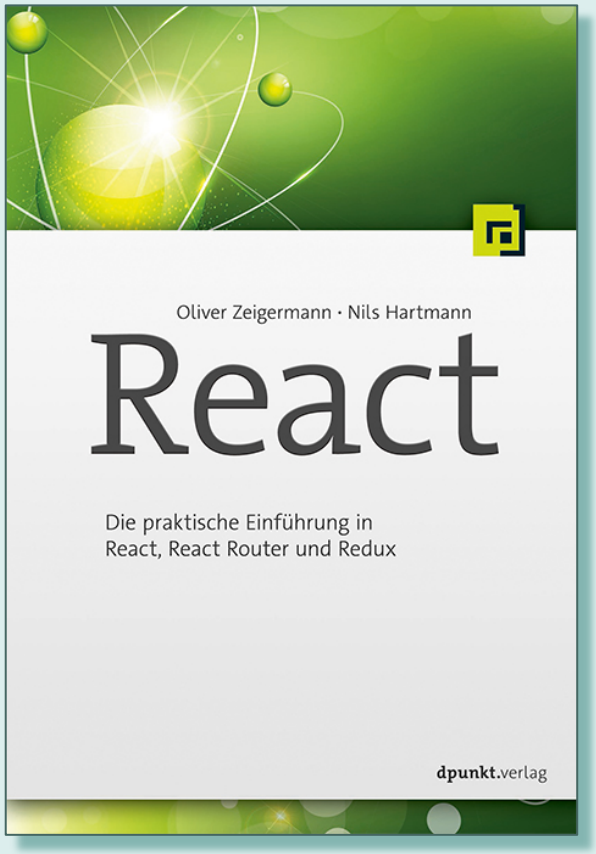
EINSTIEG IN



React

OLIVER ZEIGERMANN | MD DEV DAYS | MAI 2016



HTTP://REACTBUCH.DE

SINGLE PAGE APPLICATIONS

React

OPEN SOURCE VON FACEBOOK

https://facebook.github.io/react

React

0.3

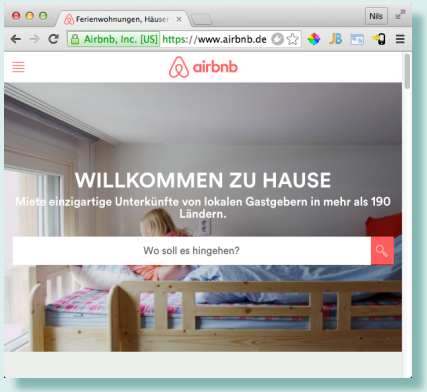
05 | 2013 – OPEN SOURCE

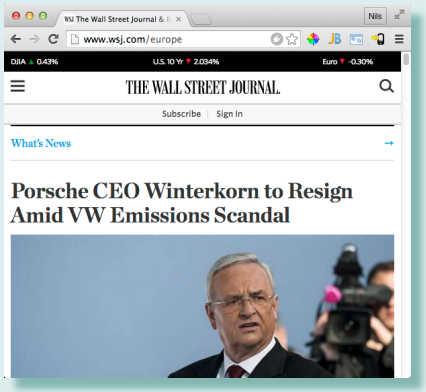
0.14.8

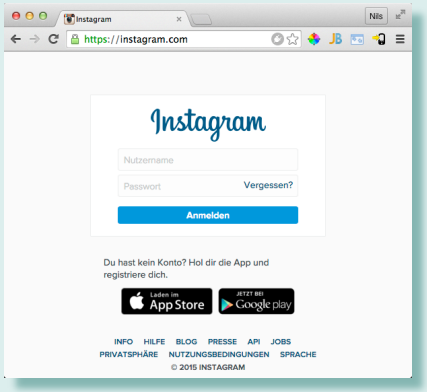
v 15.0

05 | 2016 – NEUE VERSIONIERUNG

AKTUELLE VERSION







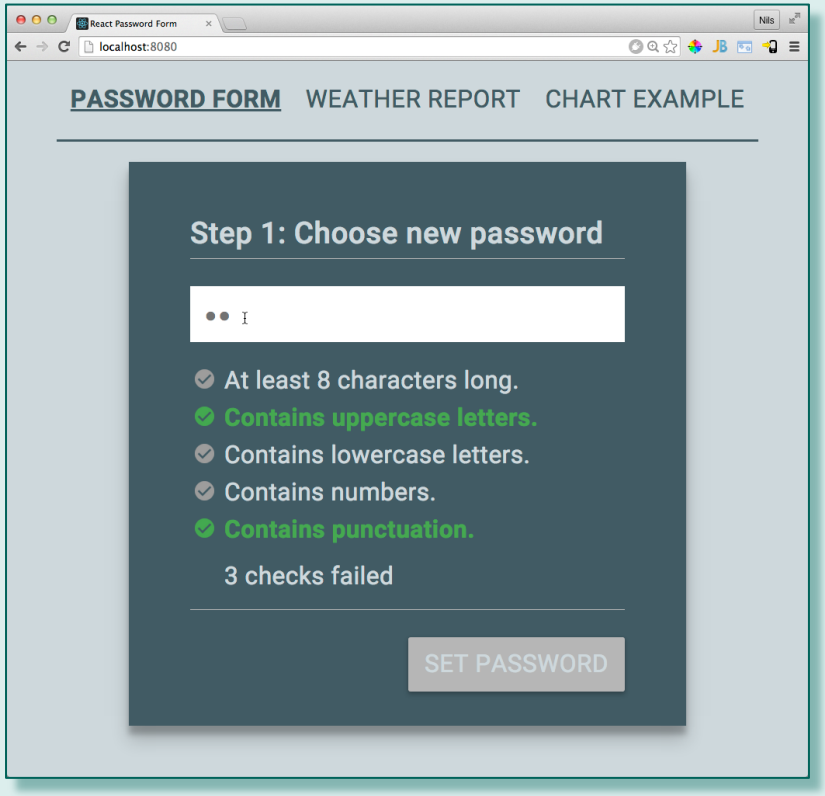
REACT IM EINSATZ

V in MVC

NUR VIEW-SCHICHT

ES6+

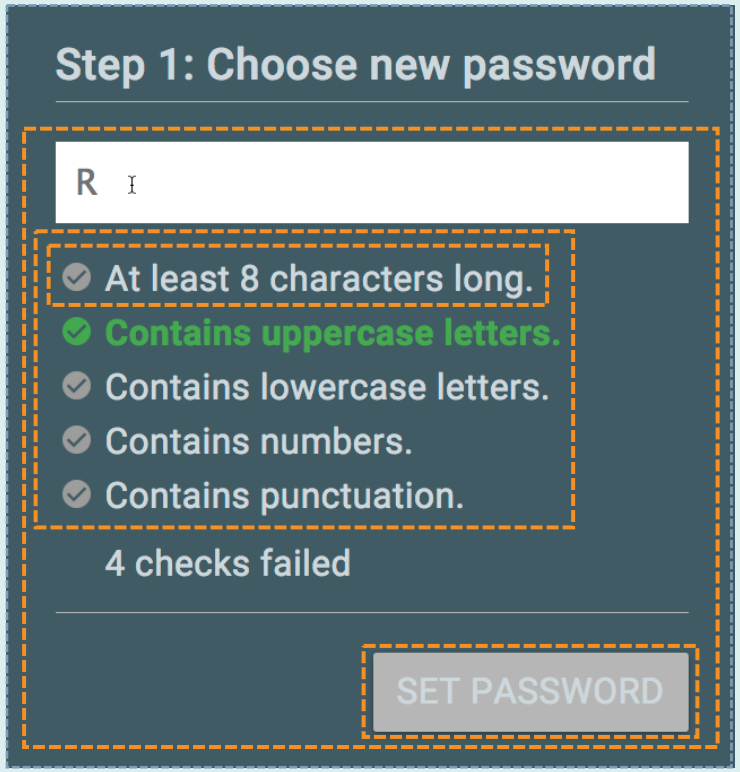
ECMASCRIPT 2015



Code: https://github.com/nilshartmann/react-example-app

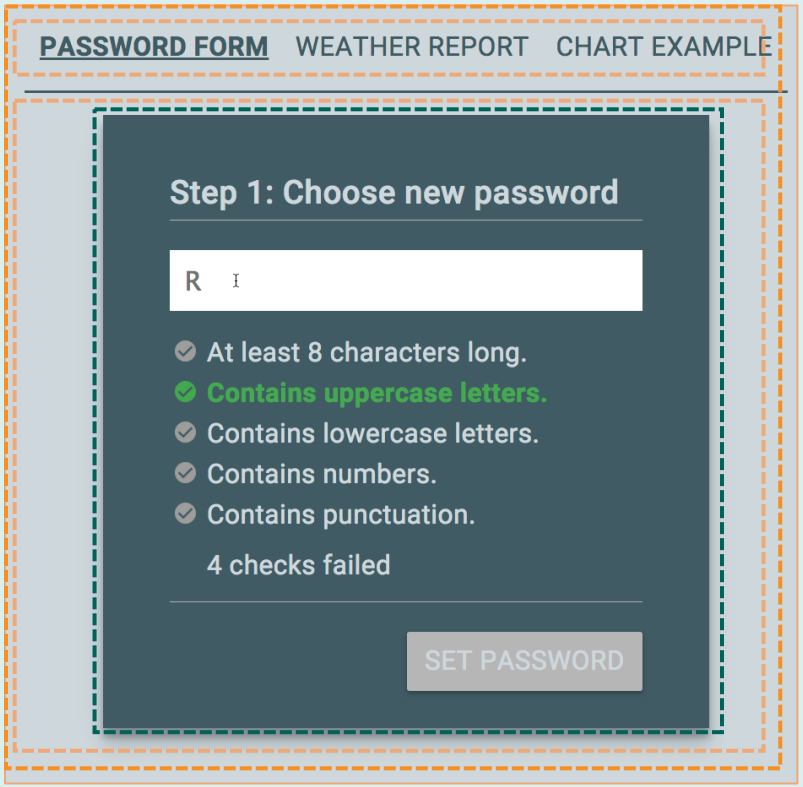
Demo: https://nilshartmann.github.io/react-example-app/

BEISPIEL ANWENDUNG




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

WIEDERVERWENDBARE KOMPONENTEN



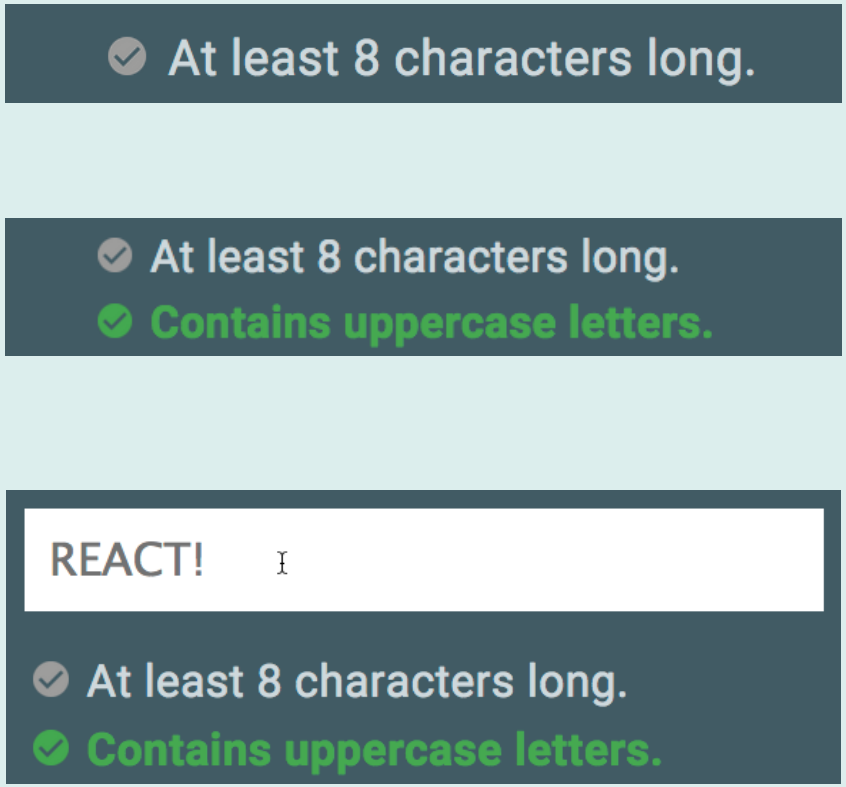

```
<Application>
  <Navigation />
  <ViewContainer>
    <PasswordView>
      . . .
      . . .
    </PasswordView>
  </ViewContainer>
</Application>
```

ANWENDUNGEN AUS KOMPONENTEN KOMPONIERT

React-Komponenten

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

- können auf dem Server gerendert werden


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 compiliert (z.B. Babel, TypeScript)**
- **Optional**

JSX

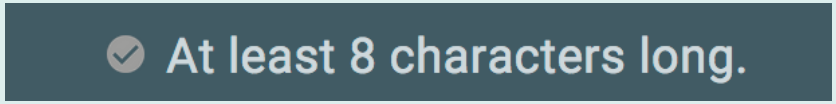
```
const name = 'Lemmy';
const greeting = <h1>Hello, {name}</h1>;
```

Übersetztes JavaScript

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

EINE REACT KOMPONENTE: ALS FUNKTION

Komponente CheckLabel

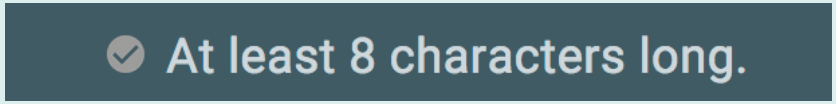


Komponenten-funktion

JSX

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

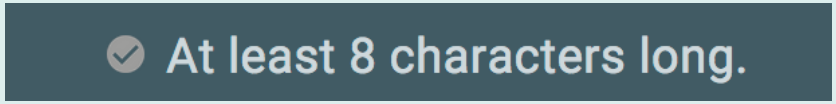
KOMPONENTE EINBINDEN



index.html

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

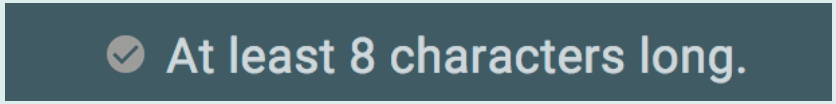
KOMPONENTE EINBINDEN



app.js

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

import CheckLabel from './CheckLabel';

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

KOMPONENTEN: PROPERTIES

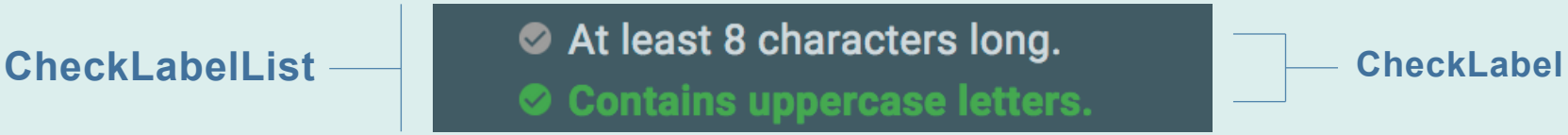



```
{
 checked: false,
 label: 'At least 8 characters long.'
}
```

```
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>;
}

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

KOMPONENTEN KLASSEN

ECMAScript 2015 Klasse

Properties über Konstruktor (optional)

Lifecycle Methoden (optional)

Render-Methode (pflicht)

Properties über this

```
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

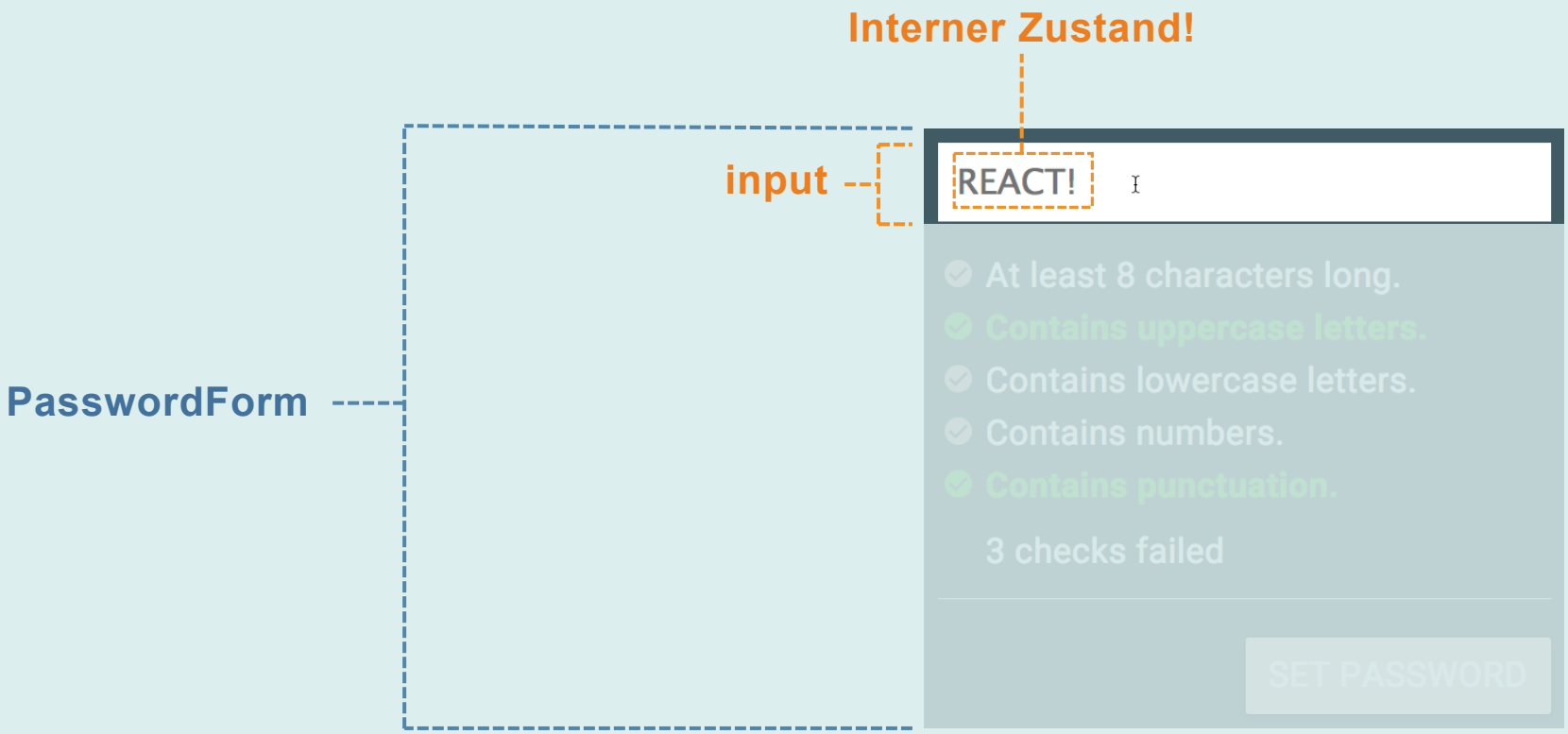
Zustand („state“): Komponenten-intern

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

Zum Vergleich: Properties

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

BEISPIEL: EINGABEFELD



BEISPIEL: EINGABEFELD

1. Input mit Wert aus State befüllen

2. Event Listener

```
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});
 }
}
```

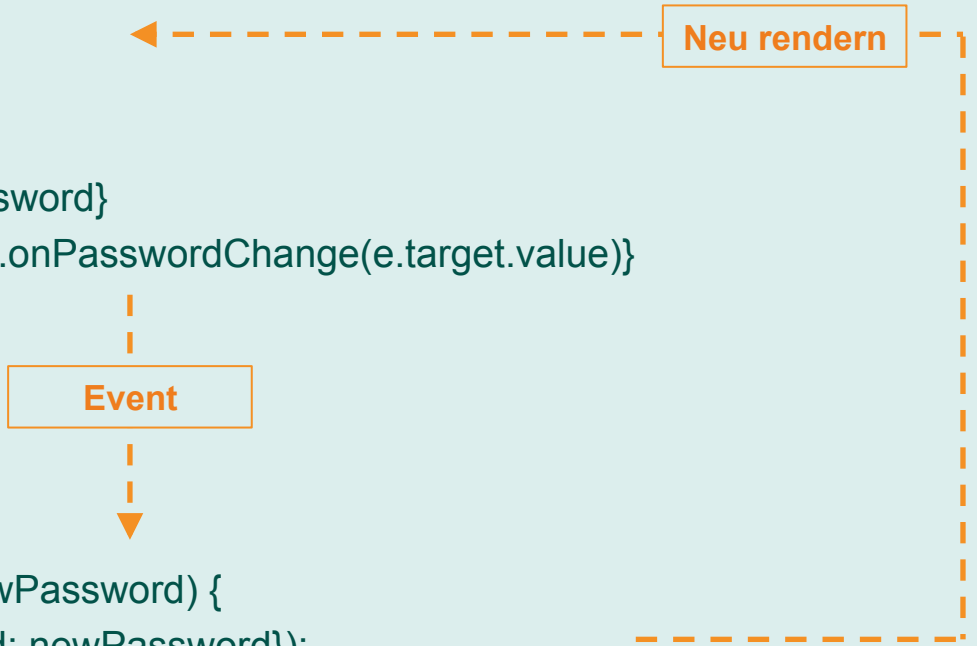
ZUSTAND: EINGABEFELD


```
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});
 }
}
```

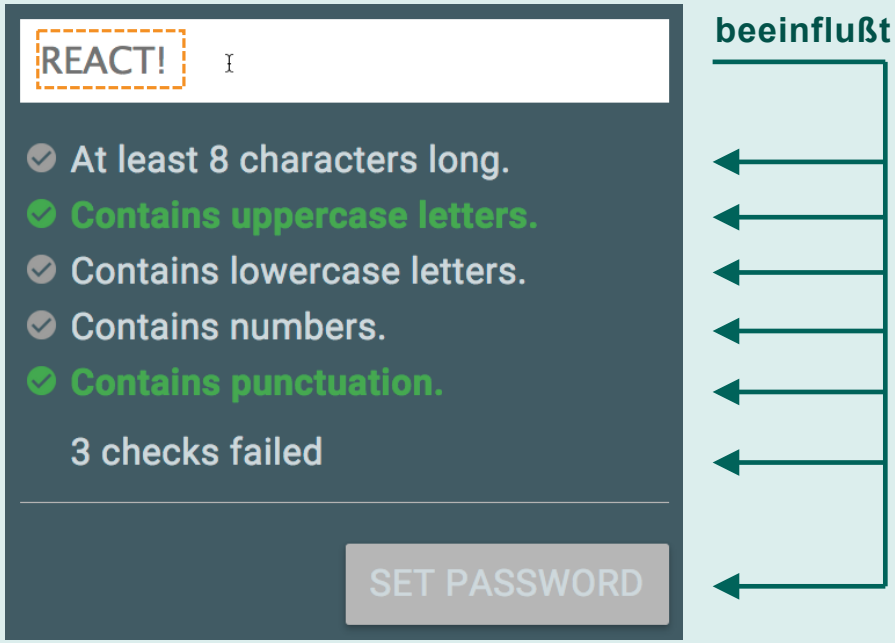
1. Input mit Wert aus State befüllen

2. Event Listener

3. Zustand neu setzen



Beispiel: Password Formular

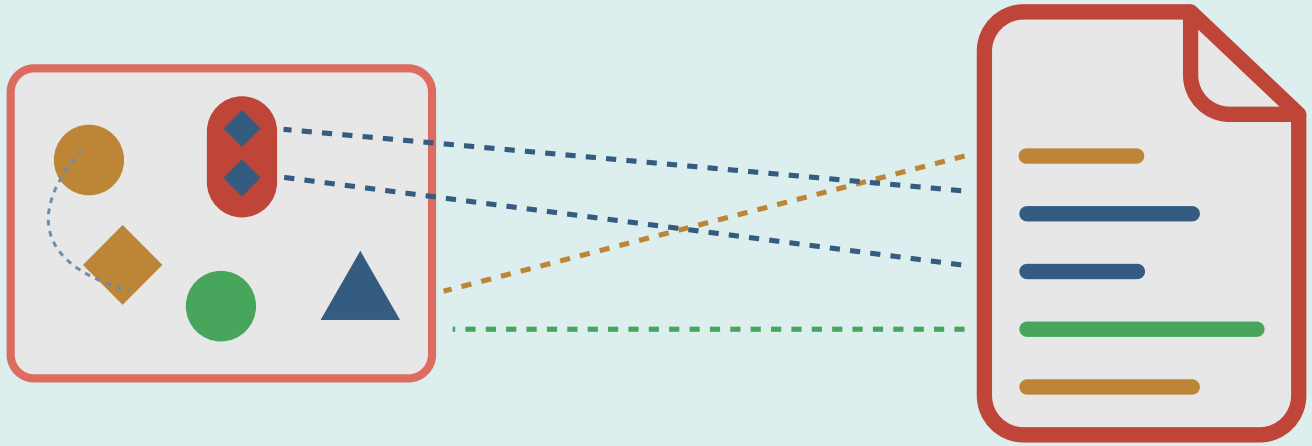


„KLASSISCHE“ OBSERVER LÖSUNG

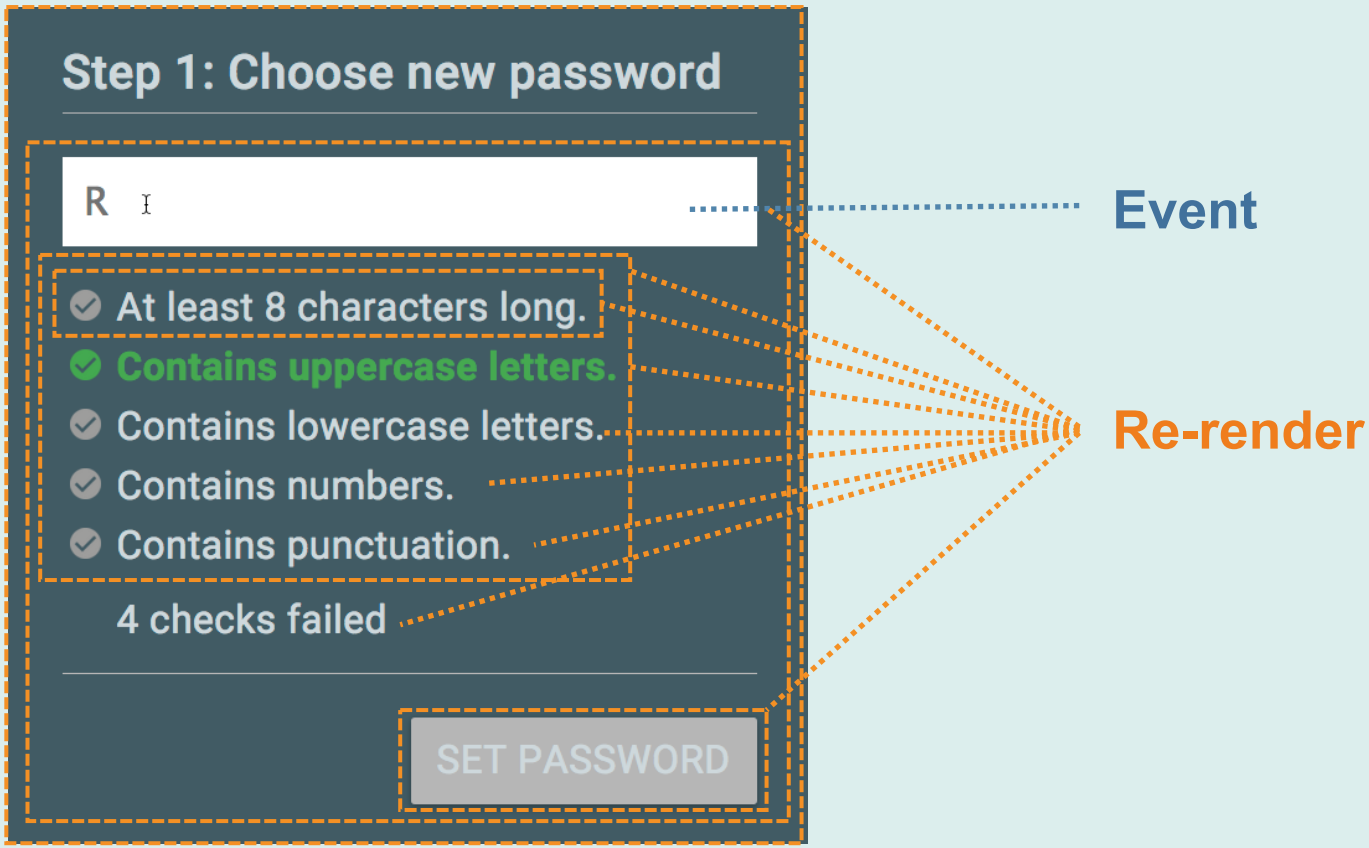
Verbinden von Model und View

- Wann wird was gebunden?
- Wie genau funktioniert das Binding?
    - Zum Beispiel: Element in Liste oder ganze Liste
- Reihenfolge von Events

Wird schnell komplex, schwer zu durchschauen



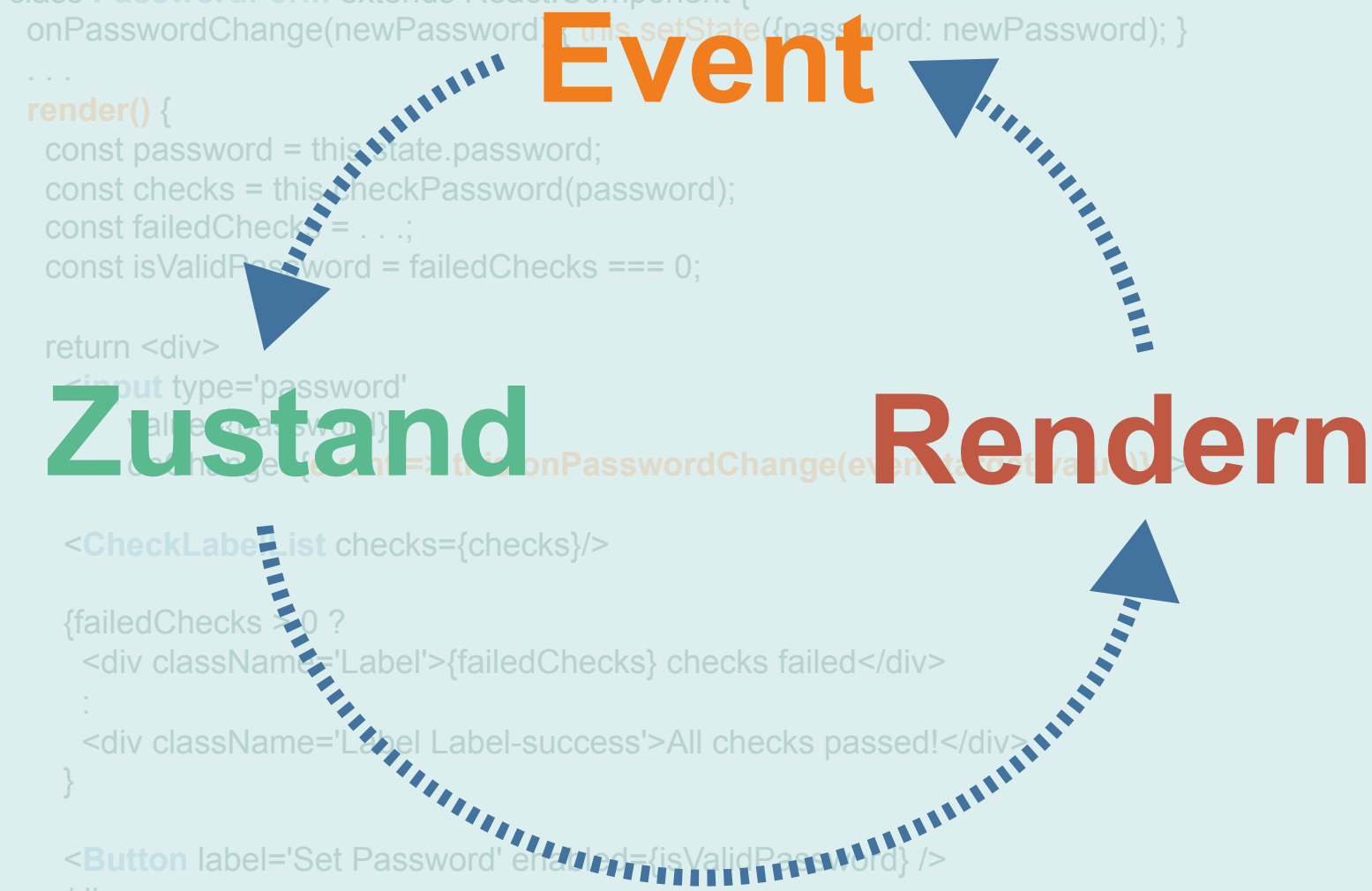
GANZ EINFACH: ALLES RENDERN



BEISPIEL 1: PASSWORD FORMULAR

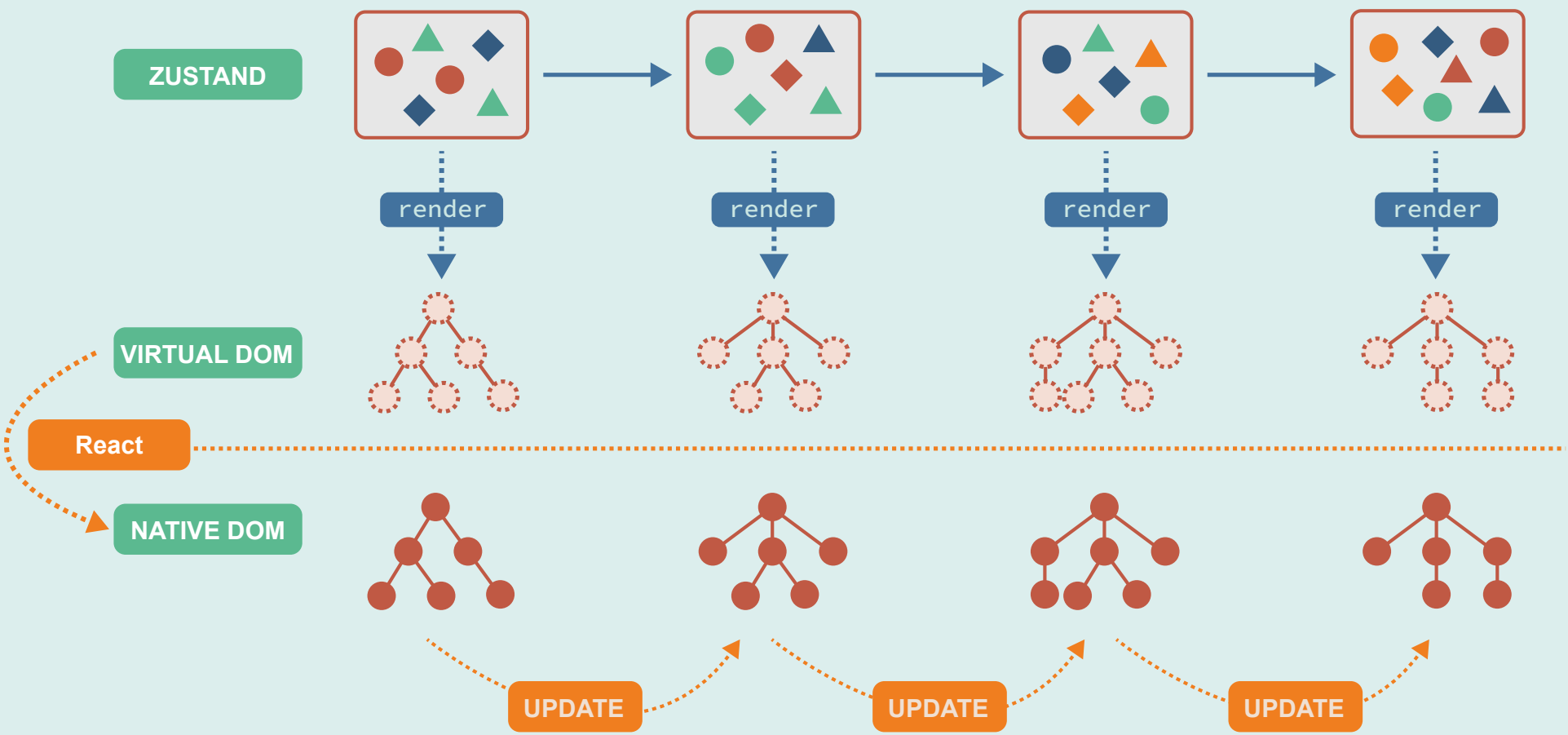
```
class PasswordForm extends React.Component {
  onPasswordChange(newPassword) { this.setState({password: newPassword); }
  . . .
  render() {
    const password = this.state.password;
    const checks = this.checkPassword(password);
    const isValidPassword = checks.failedChecks;

    return <div>
      <input type='password'
            value={password}
            onChange={event => this.onPasswordChange(event.target.value)} />

      <CheckLabelList checks={checks}/>

      . . .

      <Button label='Set Password' enabled={isValidPassword} />
    </div>;
  }
}
```

REACT: UNI DIRECTIONAL DATAFLOW



RESPOND TO EVENTS & RENDER UI

HINTERGRUND: VIRTUAL DOM



HINTERGRUND: VIRTUAL DOM

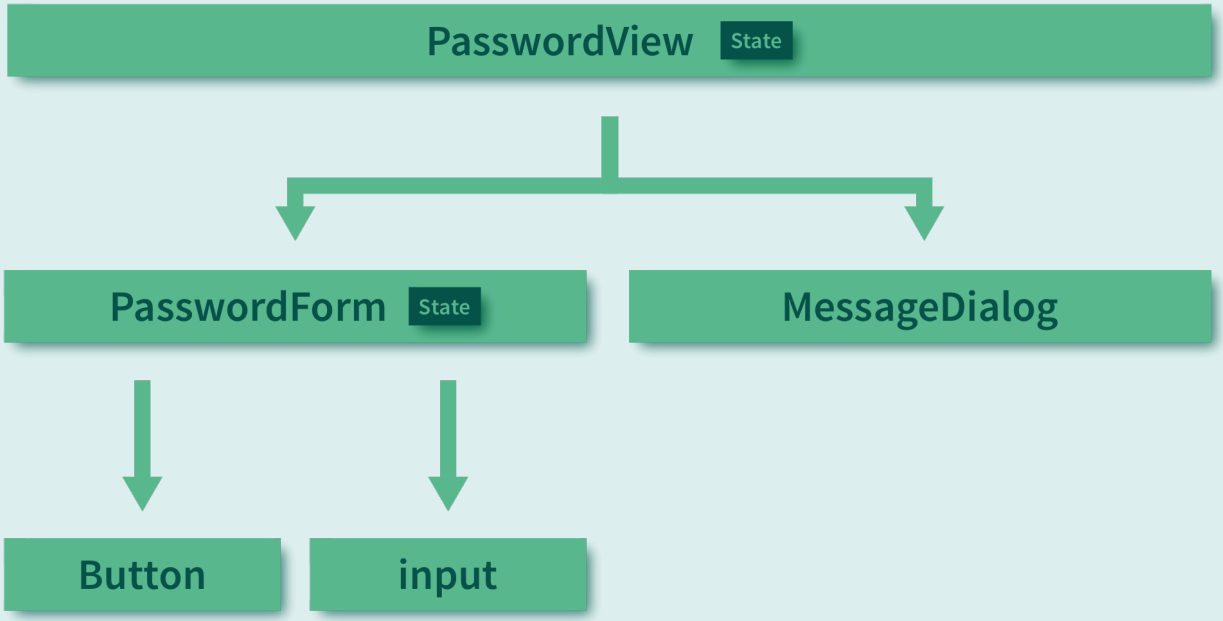
Virtual DOM

- React.createElement() liefert ein virtuelles DOM-Objekt zurück
- DOM Events sind gewrappt
- Trennung von Darstellung und Repräsentation

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

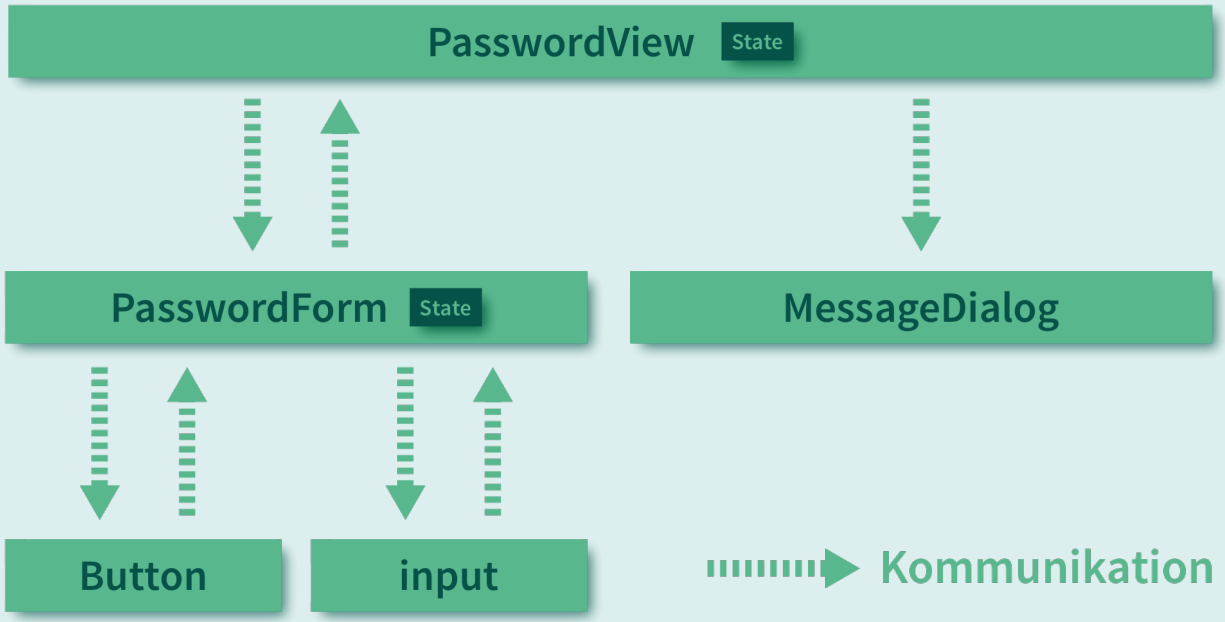
KOMPONENTENHIERARCHIEN



Typische React Anwendungen: Hierarchisch aufgebaut

- State möglichst weit oben („Container Komponenten“)
- Mehrere Komponenten mit State möglich
  - Beim neu rendern bleibt State erhalten

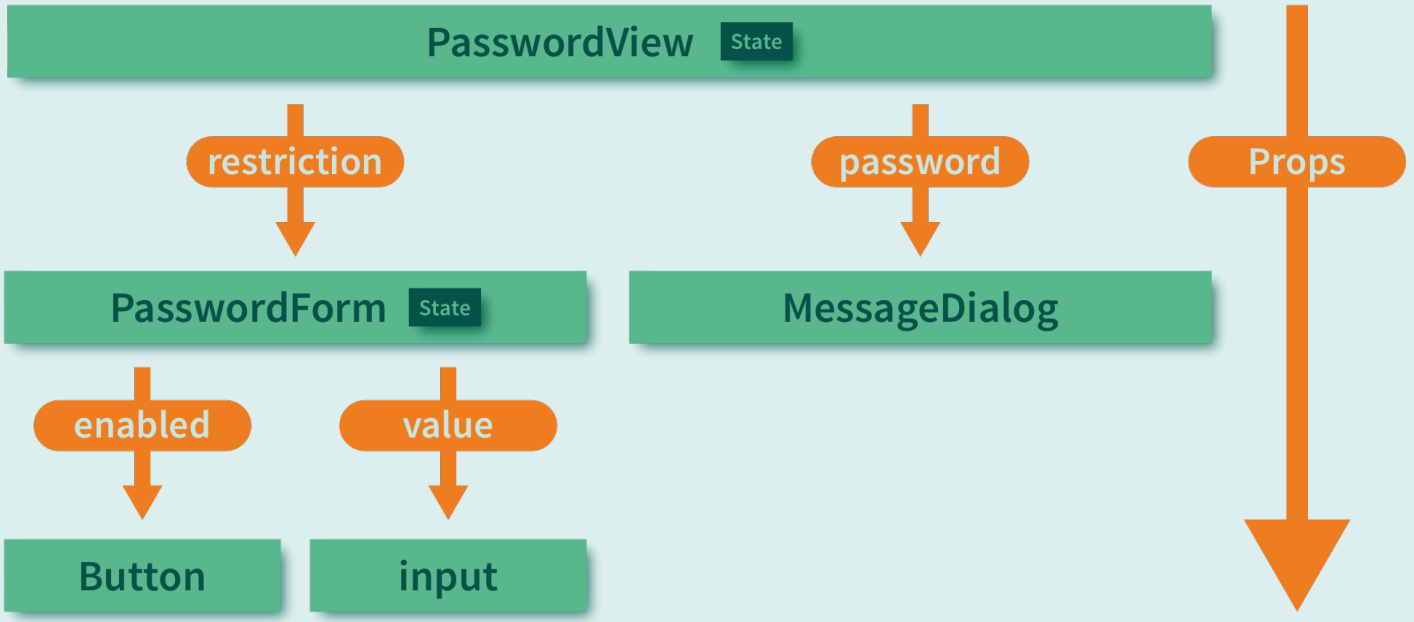
KOMMUNIKATION ZWISCHEN KOMPONENTEN



Typische React Anwendungen: Hierarchisch aufgebaut

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

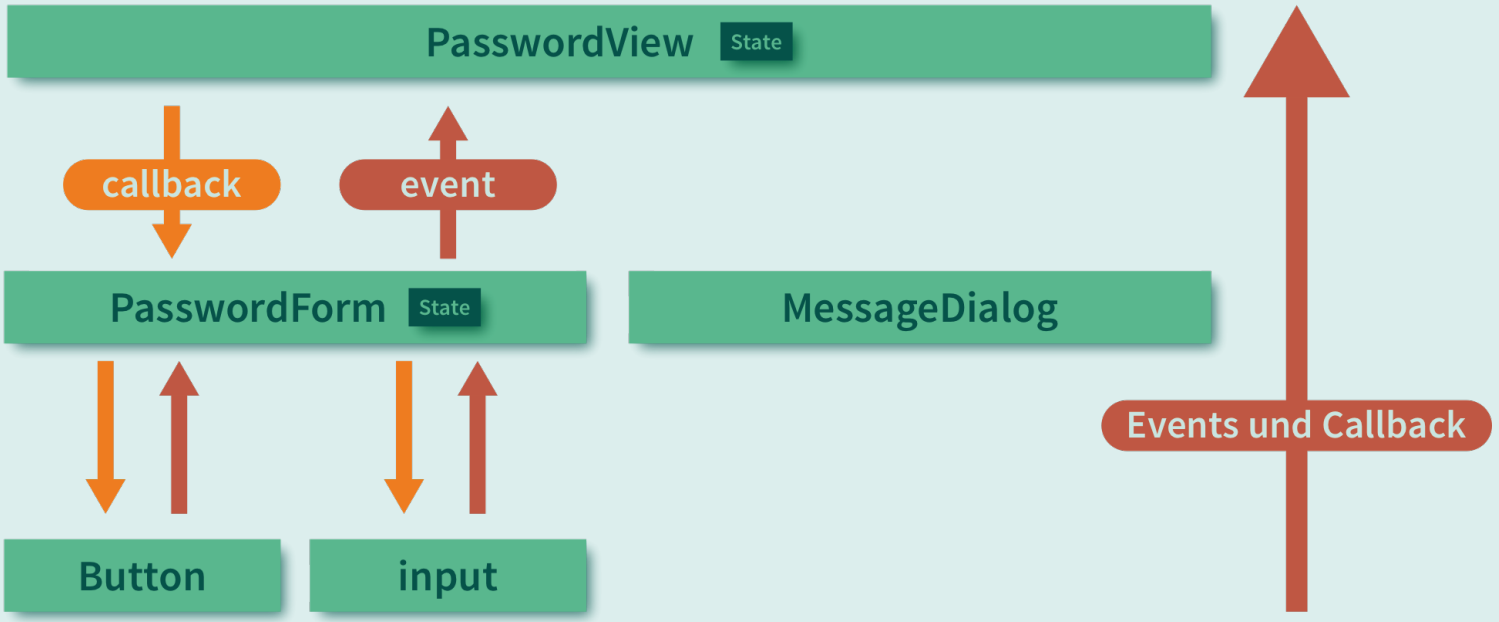
KOMMUNIKATION: PROPERTIES



Von oben nach unten: Properties

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

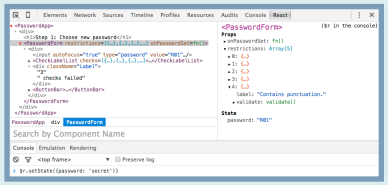
KOMMUNIKATION: EVENTS



Von unten nach oben: Events und Callbacks

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

ÖKOSYSTEM

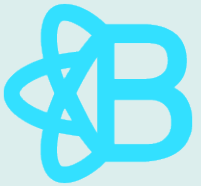


Developer Tools

material-design

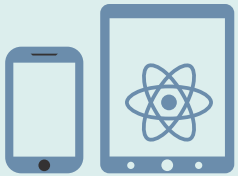
Flux Architekturpattern

Bootstrap



GraphQL & Relay

React Router



React Native

Fertige Komponenten

ZUSAMMENFASSUNG

React

- Nur View-Schicht (Komponenten)
  - Gut integrierbar mit anderen Frameworks
  - Einfache Migrationspfade möglich
- JSX statt Templatesprache („HTML in JavaScript“)
- Deklarative UI
  - Komponenten werden immer komplett gerendert
  - Kein 2-Wege-Databinding
  - Komponenten typischerweise organisiert in Hierarchien

Vielen Dank!

Fragen?

@DJCORDHOSE