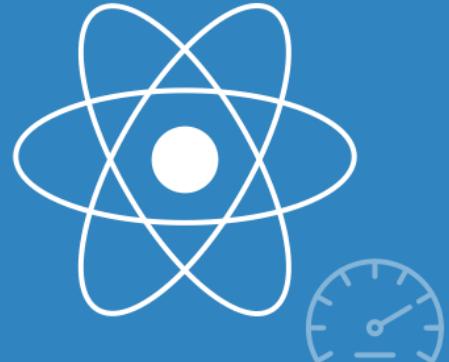


# Einführung in React

Reto Weber & Luca Ingold



# Agenda



Überblick



Features, Patterns, Übung 1 & 2



Redux & Übung 3



Weiterführendes

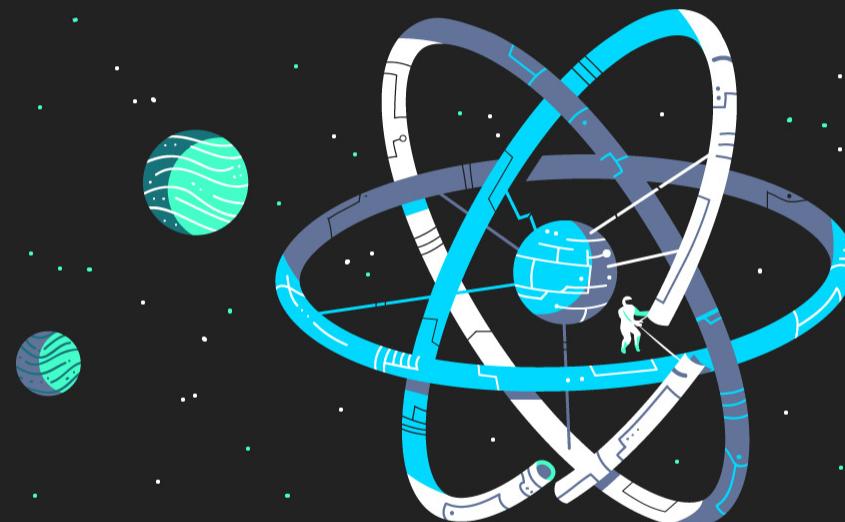


Zusammenfassung & Fragen

\*Bei Fragen & Unklarheiten gerne  
einfach unterbrechen

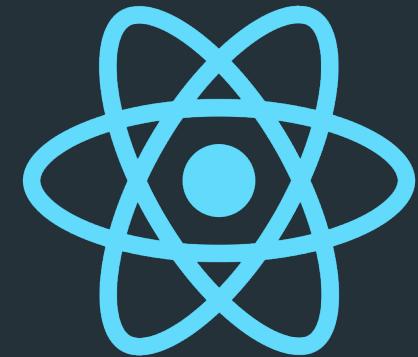
\*\* Zielgruppe: React-Neuanfänger

# Überblick



# Was ist React?

React ist eine deklarative, effiziente und flexible JavaScript-Library um User-Interfaces zu bauen.



# Traditionelle MV\* Frameworks

## Separation of Concerns

Model

Controller

ModelView

View

Data

Display logic (JavaScript)

Tailored Data

Template (HTML + custom extensions)

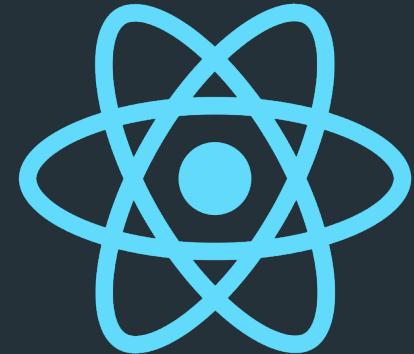
Controller, ModelView und View sind gekoppelt:  
Wird eine Komponente geändert, müssen oftmals die anderen mitangepasst werden

## Traditionelle MV\* Frameworks

	Templates	React Components
Separation of Concerns	Technology (JS, HTML)	Responsibility *Display UI*
Semantic	HTML & JavaScript	New concepts and micro-languages
Expressiveness	Underpowered	Full power of JavaScript

## Build components, not templates

- UI Components sind die zusammenhängenden Einheiten.
- UI-Gestaltung & UI-Logik sind eng gekoppelt und können zusammengeführt werden.
- Volle Leistung von JavaScript zur Darstellung des UI



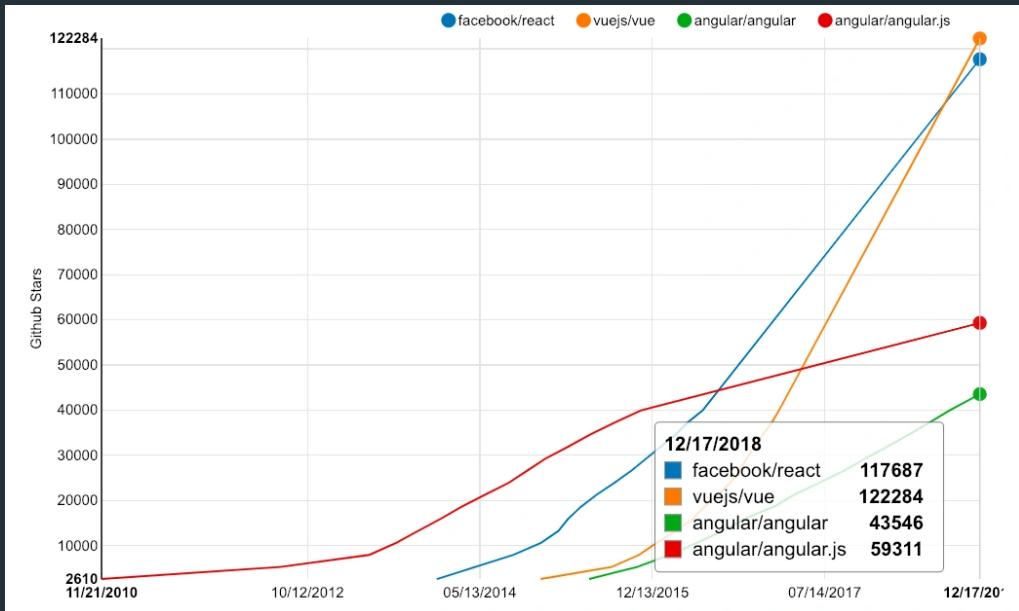
React challenges established best practices  
in traditional MVC framework

- Vielseitigkeit:
  - React kann auch via *Node* auf dem Server gerendert werden
  - Native Apps für iOS & Android mittels *React Native*
  - Native Windows Apps via *React Native for Windows* (Microsoft Build 2019)
- React implementiert einen one-way “reactive” Datenfluss, der Boilerplate reduziert und einfacher zu verstehen ist als herkömmliche Datenbindung.

# React vs Angular vs Vue – Überblick

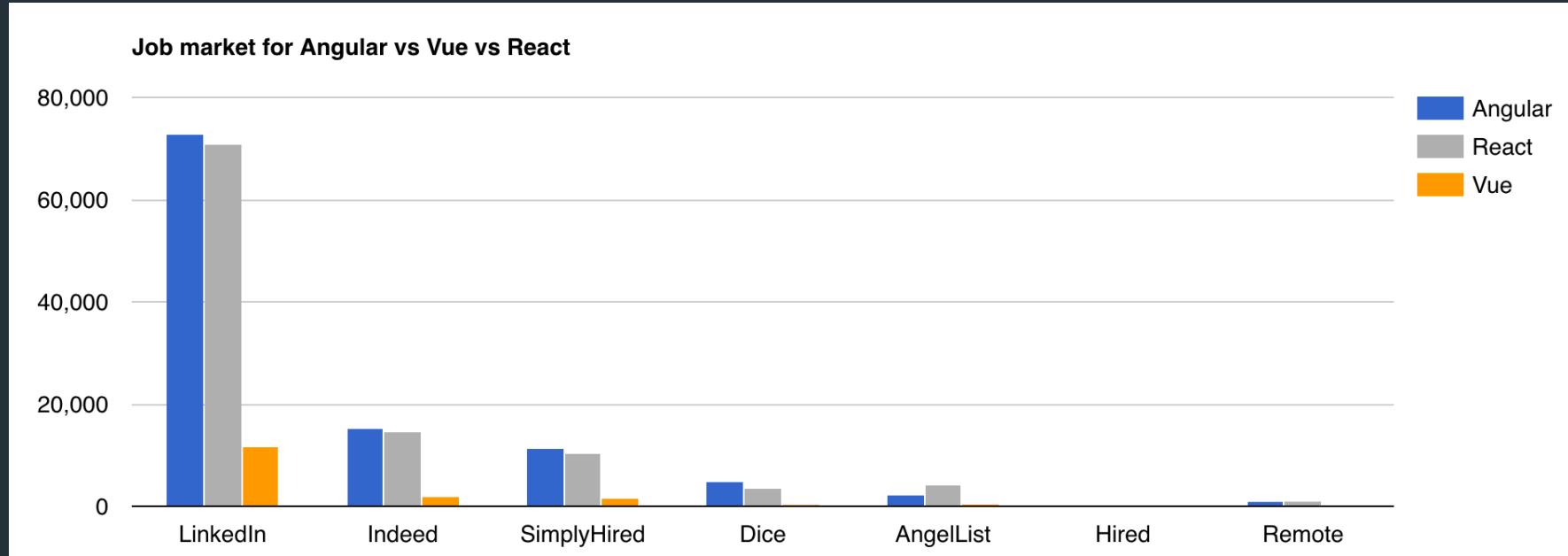
	Angular	React	Vue
Initial release	2010	2013	2014
Official site	<a href="https://angular.io">angular.io</a>	<a href="https://reactjs.org">reactjs.org</a>	<a href="https://vuejs.org">vuejs.org</a>
Approx. size (KB)	500	100	80
Current version	8	16.8.6	2.6
Maintainer	Google,	Facebook	Evan You
Typ	Framework	Library	Library
Datenfluss	Two-Way-Data-Binding	One-Way Data-Flow	Beides
TypeScript-Support	Nur TS	Ja	Ja
Mobile	Ionic (Web-Container)	React Native	Vue-Native

# React vs Angular vs Vue – Popularität / Community



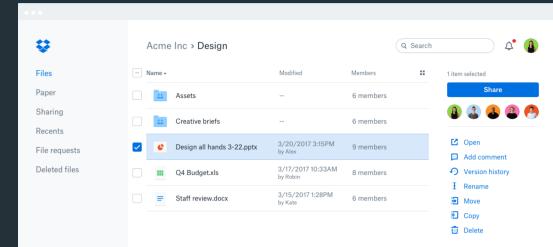
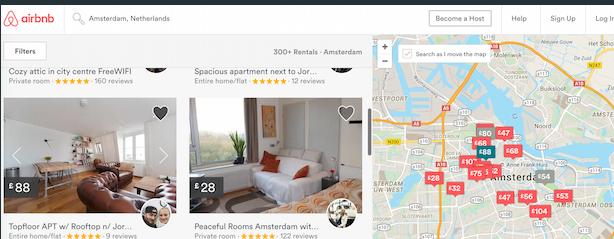
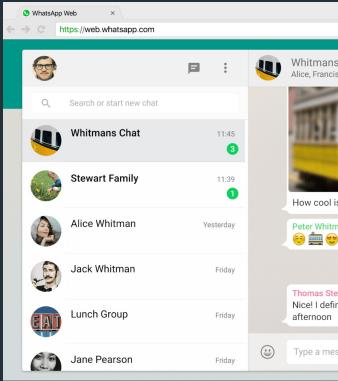
	Angular	React	Vue
# Watchers	3.3k	3.7k	5.7k
# Stars	43k	71k	122k
# Forks	11k	16k	17k
# Commits in last month	446	339	81
# Contributors	798	1.8k	240

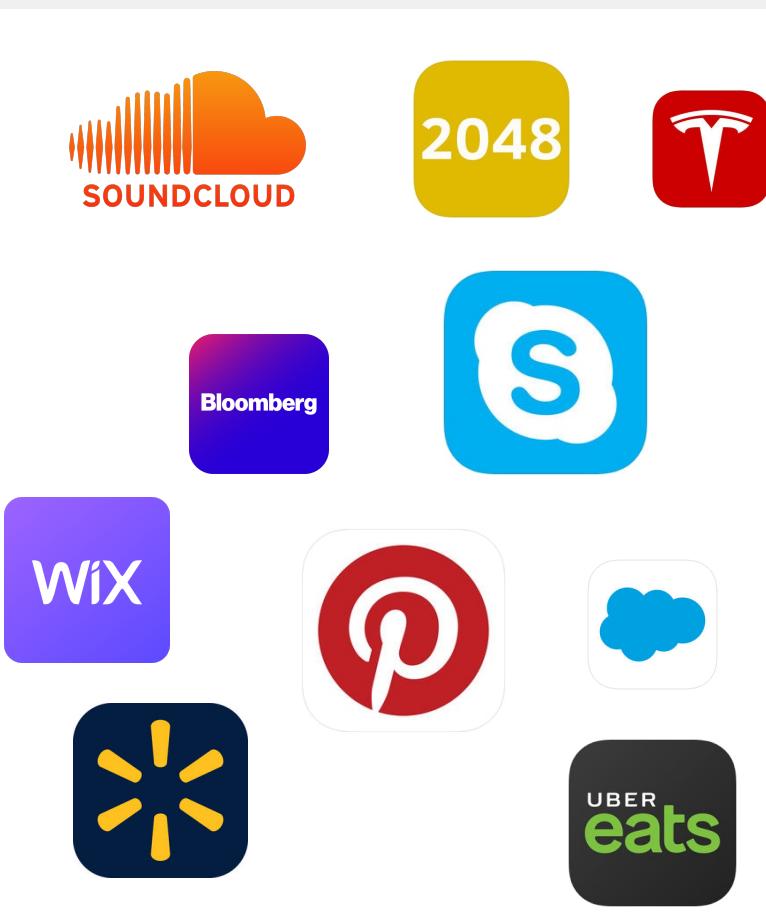
## React vs Angular vs Vue – Job Market 2018



# React Showcase

- Airbnb
- Facebook
- WhatsApp
- Uber
- Atlassian
- Cloudflare
- Dropbox
- BBC
- Instagram
- Netflix
- Paypal
- Salesforce
- Tesla





## React Native

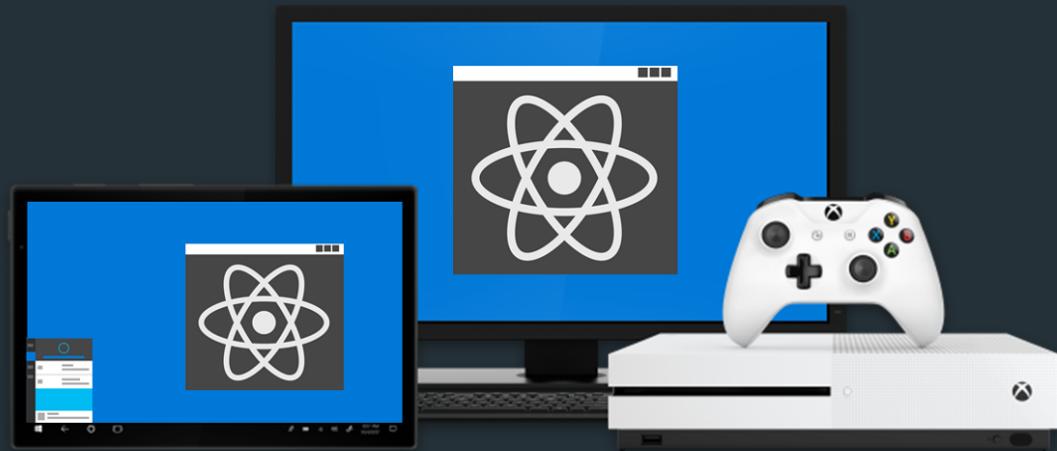
- Code-Basis in JavaScript
- Kompiliert nativ in Java & Swift (+ Cocoa)
- Etablierte native Android- / iOS-Libraries können wiederverwendet werden.

Facebook

Instagram

## Vorgestellt Microsoft Build 2019

- Vorgestellt Microsoft Build 2019
- React Native mit nativer C++-Unterstützung



## Vorteile von React

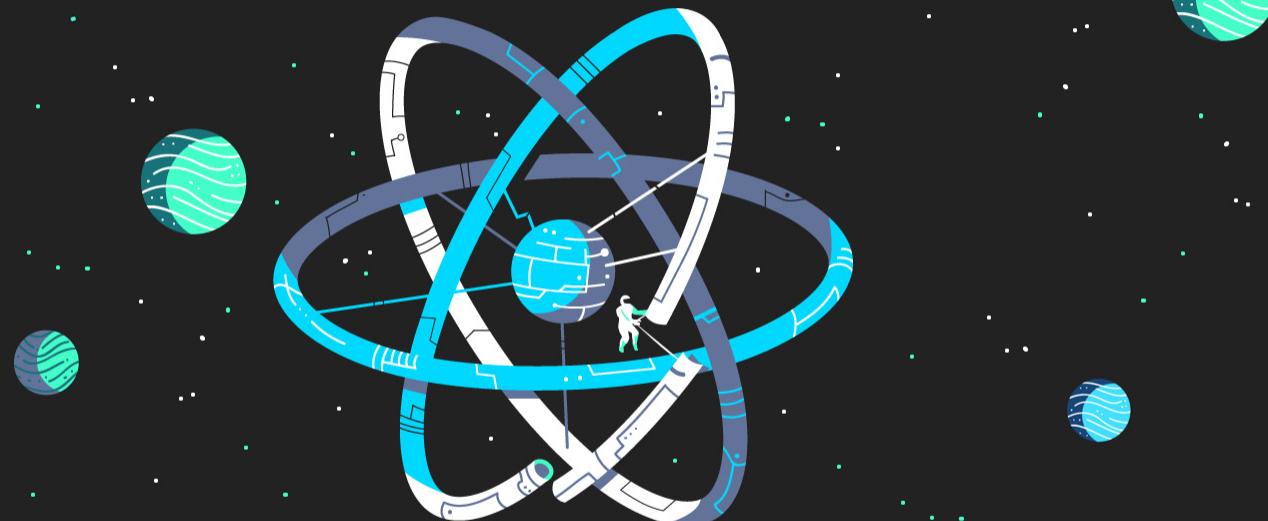
- Hohe Performance dank Leichti & virtuellem DOM
- Kann auf Client & Server-Seite verwendet werden.
- Component- & Data-Pattern verbessern die Lesbarkeit und Wartbarkeit des Codes
- Library-Gedanke (kein Framework) erhöht Flexibilität & individuelle Erweiterbarkeit
- Animiert zu funktionalem Programmieren (Flux, Hooks, etc.)
- React kann mit anderen Frameworks verwendet werden

## Lizenz

Zwar Open-Source aber früher von Facebook BSD+-Lizenziert  
-> Im Fall von Rechtsstreit Verlust der Lizenz...

Neu seit Ende 2017 unter MIT-Lizenz:  
React, Jest, Flow & Immutable.js

# Environment Setup



# Überblick

- **Package manager – Yarn oder npm**

Installieren von Libraries aus einem riesigen Ökosystem von Drittanbietern. Einfaches verwalten und aktualisieren von Versionen

- **Bundler – webpack oder Browserify**

Kreieren von js-Bundles aus einem modularen js-Code für die Verwendung im Browser. Optimierte Ladezeiten dank kleinen Packeten.

- **Compiler – Babel**

Ermöglicht das Schreiben von modernen JavaScript-Code, der trotzdem weiterhin auf älteren Browsern läuft.



```
λ npx create-react-app my-app Überblick  
npx: installed 114 in 4.308s
```

```
Creating a new React app in ~/my-app.
```

```
Installing packages. This might take a couple of minutes.  
Installing react, react-dom, and react-scripts...
```

```
yarn add v1.2.1  
info No lockfile found.  
[1/4] ⚡ Resolving packages...  
[2/4] 🚀 Fetching packages...  
[3/4] ⚙ Linking dependencies...  
[4/4] 🏺 Building fresh packages...
```

```
cd my-app  
yarn start
```

```
Happy hacking!  
λ cd my-app  
λ npm start
```

```
> my-app@0.1.0 start ~/my-app  
> react-scripts start
```

```
Starting the development server...
```

```
Compiled successfully!
```

```
You can now view my-app in the browser.
```

```
Local:          http://localhost:3000/  
On Your Network: http://192.168.37.106:3000/
```

```
Note that the development build is not optimized.  
To create a production build, use yarn build.
```

```
Compiling...  
Compiled successfully!
```

## create-react-app

- Einfachster Weg um ein neues React-Projekt zu starten
- Integriert bereits webpack, Babel & ESLint, sowie bspw. auch PWA-Support!

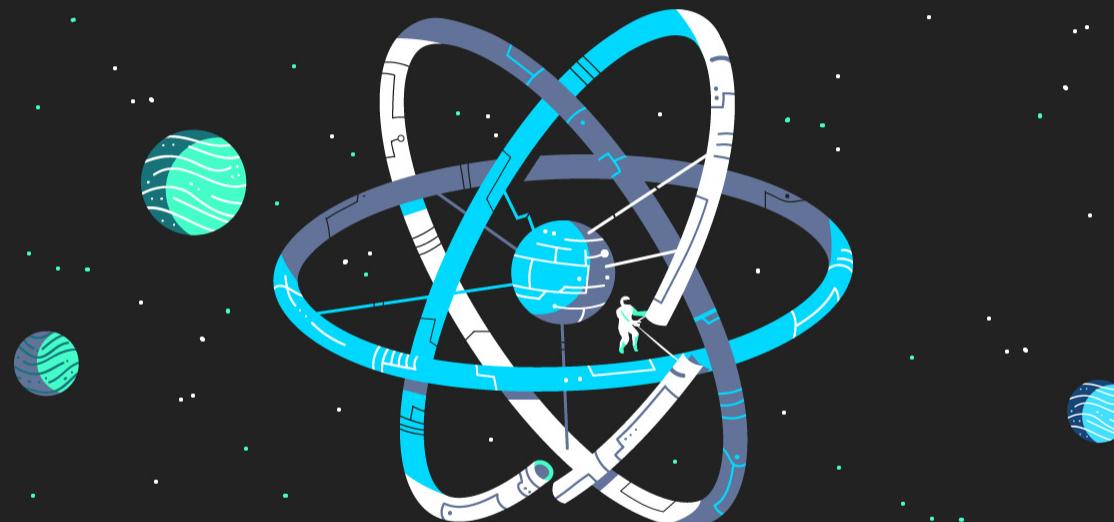
```
npx create-react-app my-app  
cd my-app  
npm start
```

<https://github.com/facebook/create-react-app>

## Tooling

- Visual Studio Code (oder bevorzugte IDE)
- Moderner Webbrowser
- Redux DevTools (Browser-Erweiterung für Debugging)

# Features



## JSX

- JSX ist eine Erweiterung des JavaScript-Syntaxes
- Es ist kein Muss JSX zu nutzen aber empfohlen.
- JSX ist schneller weil es beim Kompilieren zu JavaScript Optimierungen durchführt.
- type-safe: die meisten Fehler können dadurch bereits beim Kompilieren abgefangen werden.

## JSX

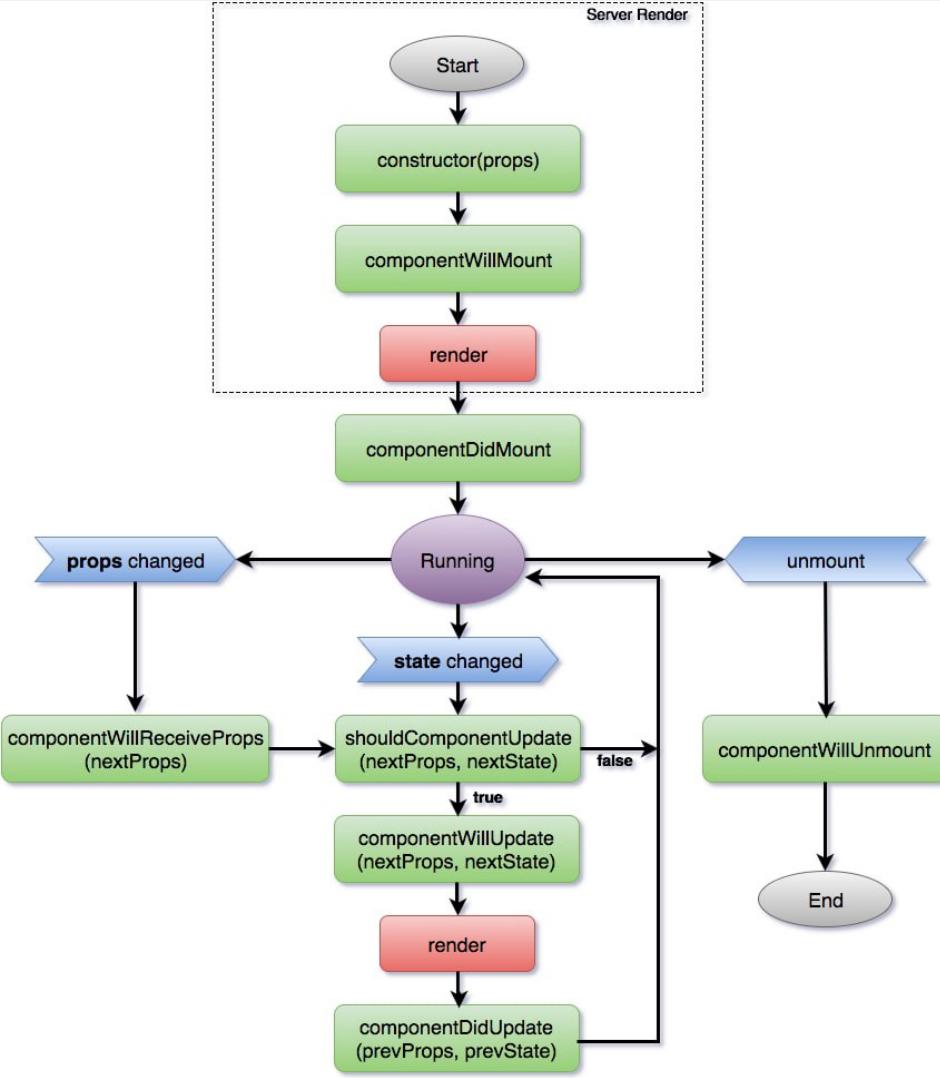
## Im Gegensatz zu klassischem HTML

- style enthält ein JS-Objekt (Effizienz, Vermeidung von XSS)
- camelCase: onClick statt onclick
- Weitere wichtige Attribute
  - class: className
  - for: htmlFor
- Object-Spread-Operator <Component {...props} />

JSX

```
var Header = React.createClass({  
  render: function () {  
    return (  
      <h1>Employee Directory</h1>  
    );  
  }  
});
```

# Lifecycle-Methoden



**Ist der Entwickler verantwortlich für Low-Level-Optimierungen?**

## DOM (Document Object Model)

### Wieso ist DOM-Manipulation langsam?

- DOM repräsentiert Baum-Datenstruktur > Änderungen & Updates somit schnell
- Browser sucht jedes Mal nach grosser Menge an Attributen oder Klassennamen, Kindern, etc. und muss diese einzeln updaten.  
*(getElementById, removeChild)*
- Nach jeder Manipulation muss das Element und die Kinder neu gerendert werden
- Rerendering macht die Applikation langsam
- Je mehr UI-Komponenten, desto teurer

**React: Re-render everything on every update**

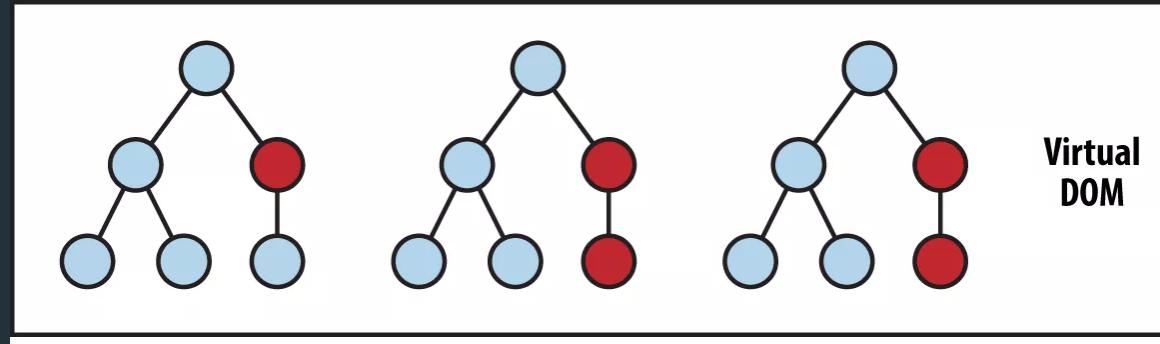
**Klingt teuer und ineffizient?**

## Virtual DOM (Document Object Model)

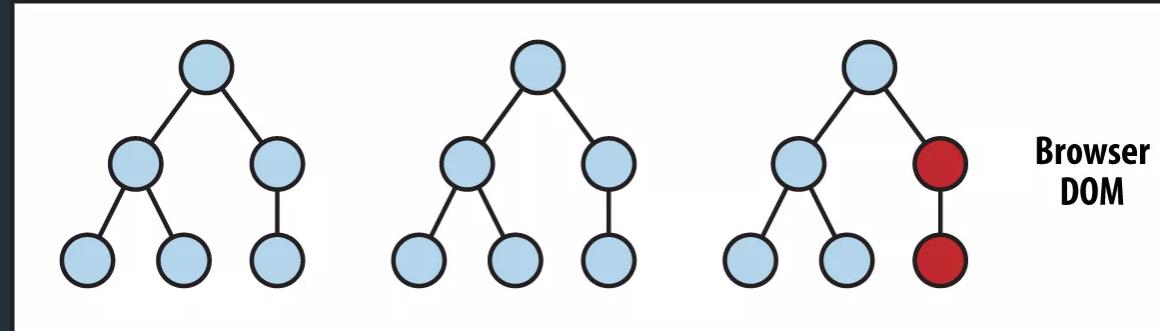
Re-render everything on every update

### Virtual DOM

- Virtuelle Repräsentation des DOMs mit signifikant besserer Performance (Javascript-Objekte)
- Jedes State-Update führt zu V-Dom Update anstatt DOM-Update
- Optimierte Performance und ausgeklügelter Diff-Algorithmus



State Change → Compute Diff → Re-render



## Nested Elements

```
class App extends React.Component {  
  render() {  
    return (  
      <div>  
        <h1>Header</h1>  
        <h2>Content</h2>  
        <p>This is the content!!!</p>  
      </div>  
    );  
  }  
}
```

## JavaScript Expressions

```
class App extends React.Component {  
  render() {  
    return (  
      <div>  
        <h1>{1+1}</h1>  
      </div>  
    );  
  }  
}
```

```
class App extends React.Component {  
  render() {  
  
    var myStyle = {  
      fontSize: 100,  
      color: '#FF0000'  
    }  
  
    return (  
      <div>  
        <h1 style = {myStyle}>Header</h1>  
      </div>  
    );  
  }  
}
```

## Styling

Style enthält ein JS-Objekt  
(Effizienz, Vermeidung von XSS)

## Components & Props

- Mit Komponenten kann GUI in unabhängige, wiederverwendbare Teile aufgeteilt werden.
- Jedes Teil kann einzeln betrachtet, verwendet & getestet werden.
- Komponenten akzeptieren beliebige Input-Parameter (“props” genannt),  
  > Beschreiben, was auf dem Bildschirm erscheinen soll.
- Props sind Read-Only / immutable!  
(Spread-Operator (...), Hilfs-Libraries verwenden [Immutabe.js, lodash, ramda, etc.])

## Komponenten erstellen

- Komponente erstellen
- render()-Methode implementieren
- Return UI-description (JSX)

```
import * as React from "react";
class Message extends React.Component {
  render() {
    return (
      <h1>Hallo Welt!</h1>
    );
  }
}
export default Message;
```

## Komponenten Rendering

- Der Return-Wert von render() ist kein DOM-Node
- Es ist eine leichtgewichtige Beschreibung des Uis (> JSX)
- Es wird mit dem aktuellen Wert im V-DOM verglichen, sodass nur ein Minimum an Änderungen an den DOM gesendet werden muss.

## Komponentenbasierte Komposition

```
render () {  
    return (  
        <div>  
            <Header />  
            <SearchBar />  
            <EmployeeList />  
        </div>  
    );  
}
```

## «Wrapper-Div»

```
class App extends React.Component {  
  render() {  
    return (  
      <p>I would</p>  
      <p>really like</p>  
      <p>to render</p>  
      <p>an array</p>  
    );  
  }  
}
```

Syntax error: Adjacent JSX elements must be wrapped in an enclosing tag (6:8)

## «Wrapper-Div»

```
class App extends React.Component {  
  render() {  
    return (  
      <div>  
        <p>I would</p>  
        <p>really like</p>  
        <p>to render</p>  
        <p>an array</p>  
      </div>  
    );  
  }  
}
```

## React.Fragment

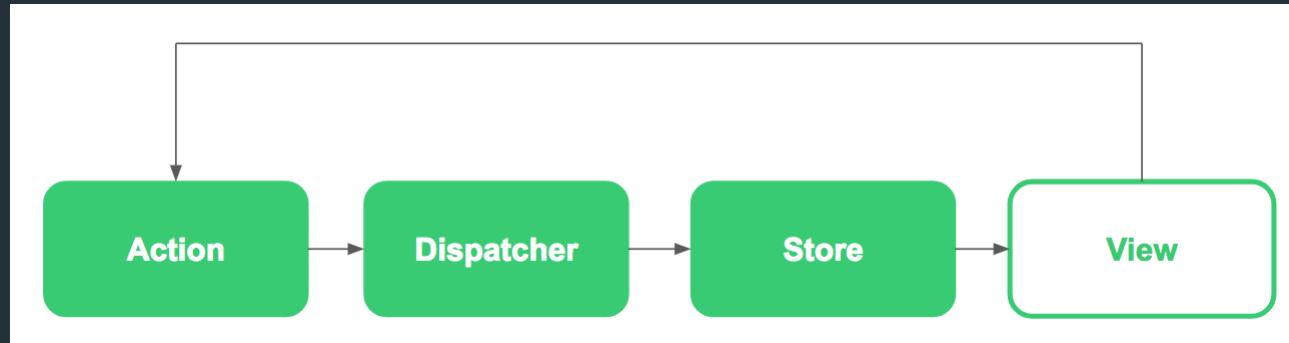
```
class App extends React.Component {  
  render() {  
    return (  
      <React.Fragment>  
        <p>I would</p>  
        <p>really like</p>  
        <p>to render</p>  
        <p>an array</p>  
      </React.Fragment>  
    );  
  }  
}
```

## React.Fragment - Kurzschreibweise

```
class App extends React.Component {  
  render() {  
    return (  
      <>  
        <p>I would</p>  
        <p>really like</p>  
        <p>to render</p>  
        <p>an array</p>  
      </>  
    );  
  }  
}
```

## Unidirektionaler Datenfluss & Flux

React implementiert einen einfachen One-Way-Datenfluss. Flux ist ein Pattern, das hilft, die Daten unidirektional zu halten.



Viele Implementierungen von Flux existieren. In dieser Einführung betrachten wir *Redux*.

## Data-Flow – Parent-to-Child

- Properties sind immutabel
- Properties können an Child-Components via Parameter weitergereicht werden.
- Auf Properties, die vom Parent weitergereicht wurden, können im Child via `this.props` zugegriffen werden.

## Data-Flow – Parent-to-Child

Parent:

```
var HomePage = React.createClass({
  render: function () {
    return (
      <div>
        <Header title="Employee Directory"/>
        <SearchBar/>
        <EmployeeList/>
      </div>
    );
  }
});
```

## Data-Flow – Child

Child:

```
var Header = React.createClass({
  render: function () {
    return (
      <h1>{this.props.title}</h1>
    );
  }
});
```

## Inverse Data Flow – Child to Parent (via Callback-Function)

Parent:

```
var HomePage = React.createClass({
  searchHandler:function(key) {
    alert('Search key: ' + key); ← Callback-Function
  },
  render: function () {
    return (
      <div>
        <Header text="Employee Directory"/>
        <SearchBar searchHandler={this.searchHandler}>           ↑
        <EmployeeList/>                                         Referenz via props übergeben
      </div>
    );
  }
});
```

## Inverse Data Flow – Child

Child:

```
var SearchBar = React.createClass({
  keyChange: function(event) {
    var searchKey = event.target.value;
    this.props.searchHandler(searchKey);
  },
  render: function () {
    return (
      <input type="search" onChange={this.keyChange}>
    );
  }
});
```

Aufruf Callback-Function-Referenz aus Parent

```
import { PropTypes } from "prop-types";
class Hello extends React.Component {
```

```
  static propTypes = {
    name: PropTypes.string,
    myFunction: PropTypes.func,
    customObject: PropTypes.object,
    isActive: PropTypes.bool,
    sum: PropTypes.number,
    numbers: PropTypes.array,
  };
}
```

```
render() {
  const { name } = this.props;
  return (
    <h1>{name}</h1> );
}
export default Hello;
```

## Props – Typechecking

- React ist kompatibel mit TypeScript
- Vanilla-Javascript > Proptypes verwenden:  
*npm install --save prop-types*

oder eben TypeScript...

```
interface Props {
  userName: string;
}

export const HelloComponent = (props: Props) => (
  <h2>Hello user: {props.userName} !</h2>
);
```

## State

- Props sind Read-Only / Immutable!
- State erlaubt React-Komponenten auf z.B. User- oder Netzwerk-Aktionen zu reagieren und ihren Output entsprechend zu ändern.
- State nie direkt manipulieren, sondern immer über setState()

```
constructor(props) {  
  super(props);  
  this.setState({  
    username: this.props.username  
  });  
}
```

Zugriff(z.B. render): {this.state.username}



```
// Wrong  
this.state.fullName = 'eduardo';  
  
// Correct  
this.setState({  
  fullName: 'eduardo',  
});
```

```
import React, { Component } from 'react';
import 'bootstrap/dist/css/bootstrap.css';

class Home extends Component {
  state = {
    firstName: '', // Initial-State
  }

  handleFormChange = (e) => {
    this.setState({
      firstName: e.target.value,
    });
  }

  render() {
    return (
      <div className="Home mt-3 text-center">
        <h2>How to setState in React</h2>
        <h5>by eduardo.vedes</h5>
        <hr />
        <div className="my-3">
          <h3>User Info</h3>
          <div>First Name: {this.state.firstName}</div>
          <hr />
        </div>
        <div className="my-3">
          <h5>Form</h5>
          <div className="form-group" // Trigger State-Änderung>
            <label for="firstName">First Name: </label>
            <input className="ml-2" type="text" name="firstName" onChange={this.handleFormChange} />
          </div>
        </div>
      </div>
    );
  }
}

export default Home;
```

## State-Updates werden 'gemerged'

```
● ● ●  
  
state = {  
  dogId: 991,  
  dogName: 'Pluto',  
  dogAge: 8,  
  dogIsMicroChipped: true,  
  dogLastVaccineDate: 1538784000,  
  dogNeedsVaccination: false,  
}  
  
this.setState({  
  dogNeedsVaccination: true,  
});
```

## Auf Props-Updates reagieren

```
state = {  
  magicNumber: 0,  
}  
  
componentDidMount() {  
  this.setState({  
    magicNumber: this.props.magicNumber,  
  });  
}  
  
componentDidUpdate(prevProps){  
  if (prevProps !== this.props) {  
    this.setState({  
      magicNumber: this.props.magicNumber,  
    });  
  }  
}
```

## State

### Anti-Pattern!

- ComponentDidUpdate für render() > Bad performance

- Bricht den props cascade-flow.

- Nicht machen, wenn vermeidbar. / wenn props nicht geändert werden müssen!

> Stateless Components (Nur props, kein state) sind zu bevorzugen.  
State erhöht Komplexität

## Props vs. State

### Props

- immutable
- ermöglicht React schnelle Referenz-Checks
- um Daten nach unten zu geben (Parent > Child)
- Bessere Performance

### State

- wird isoliert im Controller gemanaged
- mutable
- schlechtere Performance
- sollte nicht von child-components aus zugegriffen werden (props benutzen)

## TypeError

Cannot read property 'setState' of undefined

```
class MyButton extends React.Component {
  constructor(props) {
    super(props);
    this.state = { toggle: false };
  }

  toggleButton() {
    this.setState({ toggle: !this.state.toggle });
  }

  render() {
    return (
      <div>
        <button onClick={this.toggleButton}>Toggle</button>
        <p>{this.state.toggle}</p>
      </div>
    );
  }
}

export default MyButton;
```

## React Binding Patterns - Handling 'this'

1. **Bind in Render** (minimal schlechtere Performance)

```
onChange={this.toggleButton.bind(this)}
```

2. **Arrow Function in Render** (minimal schlechtere Performance)

```
onChange={e => this.toggleButton()}
```

3. **Bind in Constructor**

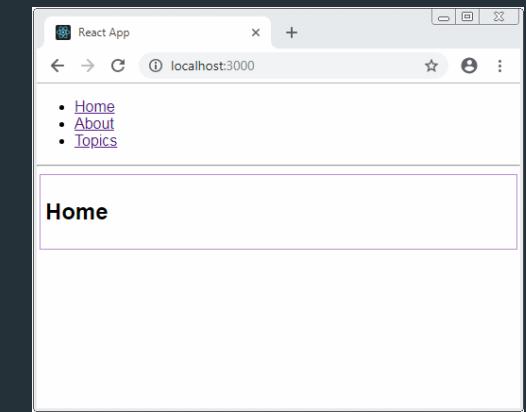
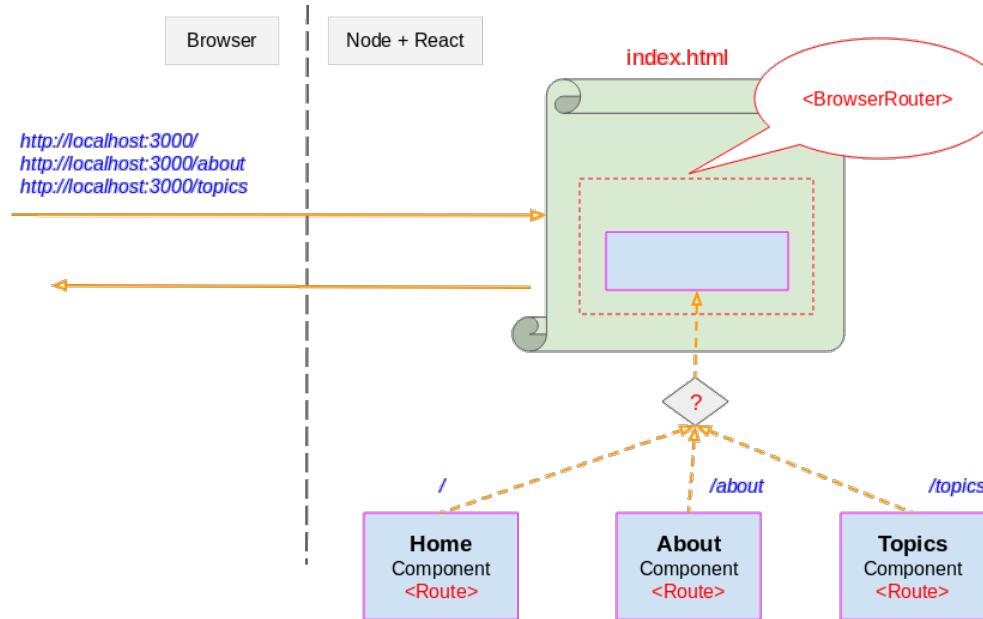
```
constructor(props) {  
  super(props);  
  this.toggleButton = this.toggleButton.bind(this);  
}
```

4. **Arrow Function in Class Property**

```
toggleButton = () => { ... }
```

=> #4 ist die bevorzugte Variante. Performant und keine Repetition wie bei Konstruktor-Variante (#3)

# React Router



## React Router

<BrowserRouter> vs <HashRouter>  
// <BrowserRouter>  
http://example.com/about  
// <HashRouter> --> support auch für alte Browser  
http://example.com/#/about

```
<BrowserRouter>
  <Route exact path="/" component={Home}/>
  <Route path="/about" component={About}/>
</BrowserRouter>
```

-> Siehe im Bsp.-Projekt die Klassen im Ordner “Navigation”

## Kaffe-Pause

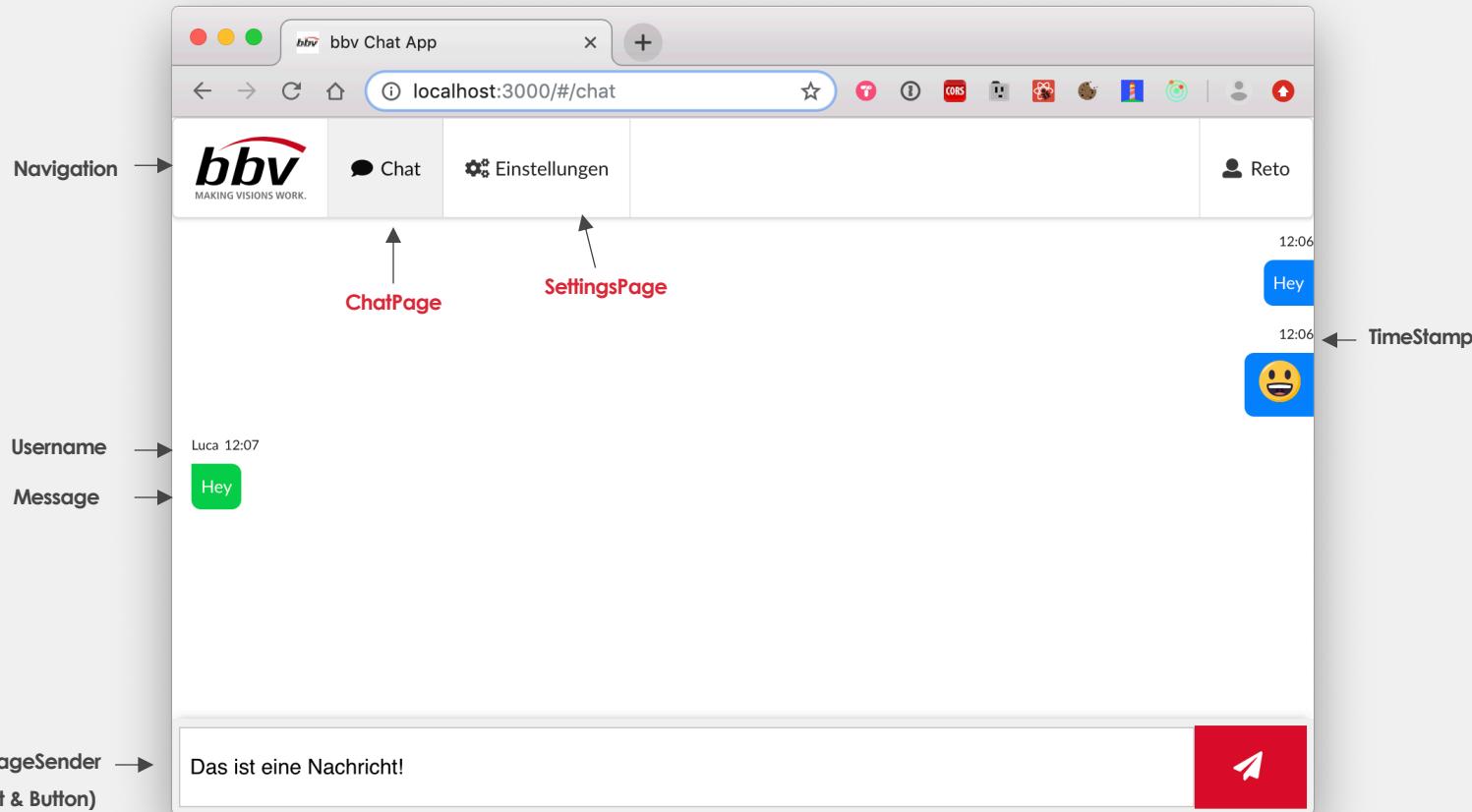
15 Minuten



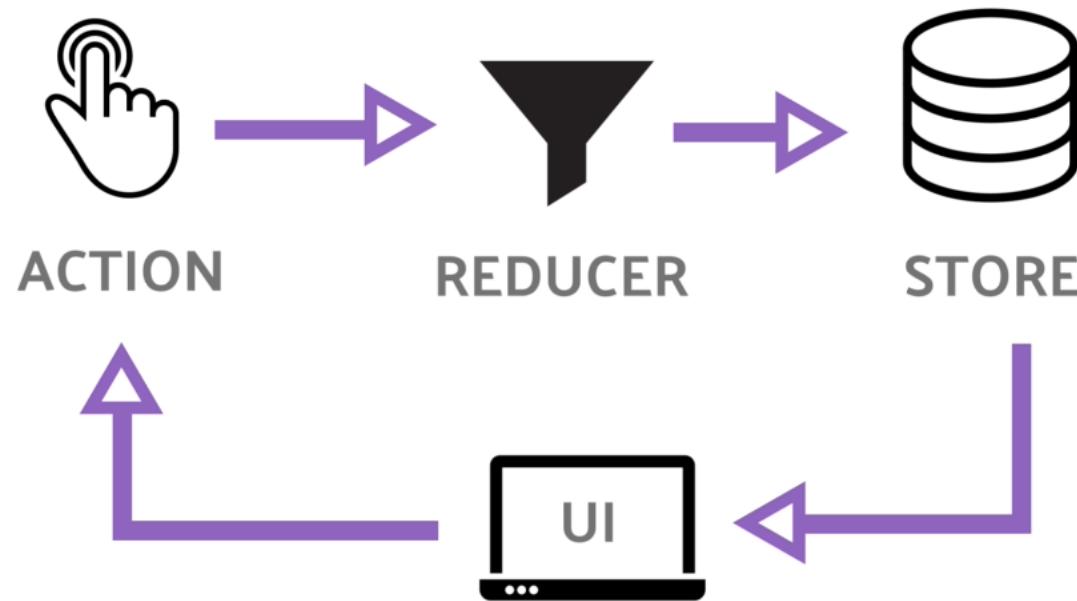
# Übung 1

Siehe uebungen/Uebung1.ms

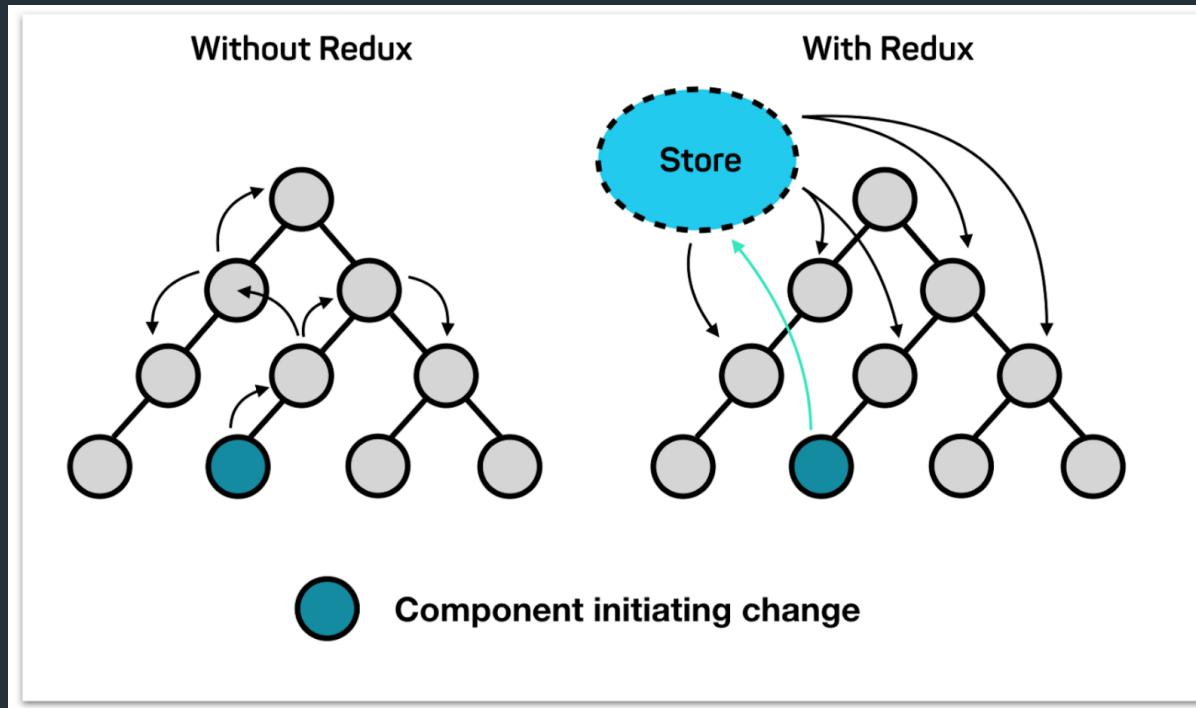




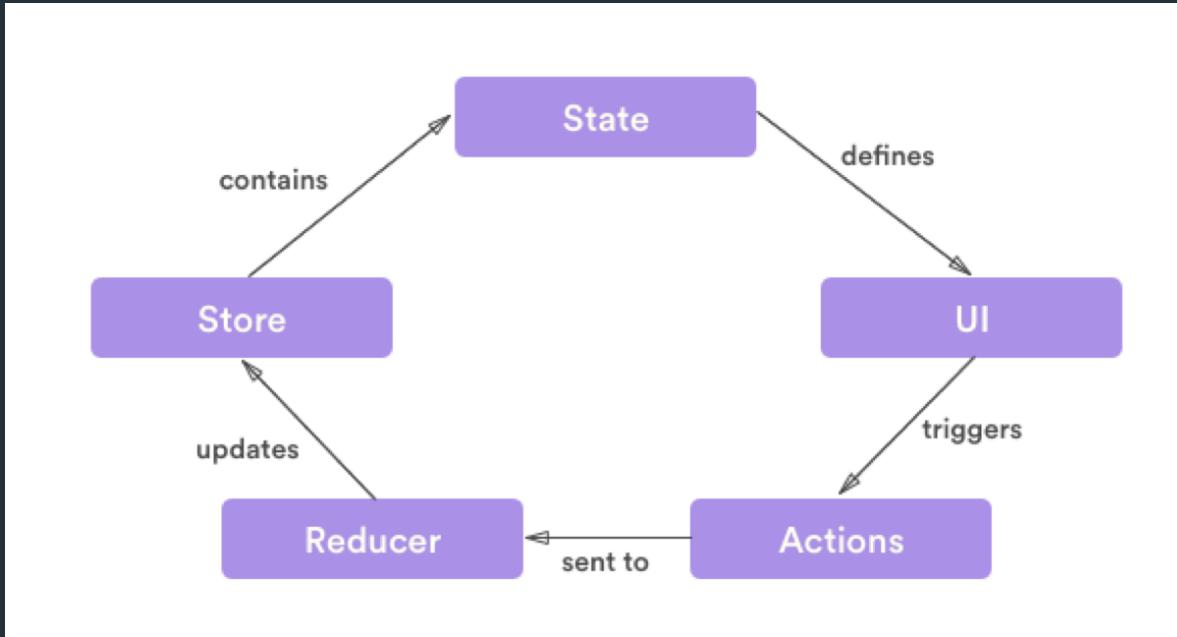
# Redux



# Wieso Redux?



## Redux-Flow



# Actions

- Actions are payloads of information that send data from your application to your store.
- They are the only source of information for the store.
- Send them to the store using `store.dispatch()`.

*actions.js*

## Action-Type

```
export const ADD_TODO = 'ADD_TODO'
```

## Action-Creator

```
export function addTodo(text) {
  return {
    type: ADD_TODO,
    text           // z.B. text="Äpfel kaufen"
  }
}
```

```
reducers.js
import { combineReducers } from 'redux'
import { ADD_TODO } from './actions'

const INITIAL_STATE = { todos: [] }

function todos (state = initialState, action) {
  switch (action.type) {
    case ADD_TODO: return [
      ...state,
      {
        text: action.text,
        completed: false
      }
    ]
    //...weitere action-types
    default: return state
  }
}
const todoApp = combineReducers({todos})
export default todoApp
```

## Reducers

Reducers specify how the application's state changes in response to actions sent to the store.

The **Store** is the object that brings all together.

- Holds application state;
- Allows access to state via getState();
- Allows state to be updated via dispatch(action);
- Registers listeners via subscribe(listener);
- Handles unregistering of listeners

**Important:** only one store per Redux application.

## Store

*store.js (im Bsp.-Projekt store/index.js)*

```
import { createStore } from 'redux'  
import todoApp from './reducers'
```

```
const store = createStore(todoApp)
```

---

*index.js (Einstiegspunkt der React App)*

```
import { Provider } from "react-redux";  
import store from "./store";
```

```
ReactDOM.render(  
  <Provider store={store}>  
    <App />  
  </Provider>,  
  document.getElementById("root") );
```

# Store

## Verwendung (pur, ohne Hilfmethoden):

```
// Log the initial state
```

```
console.log(store.getState())
```

```
// Every time the state changes, log it
```

```
// Note that subscribe() returns a function for unregistering the listener
```

```
const unsubscribe = store.subscribe(() => console.log(store.getState()))
```

```
// Dispatch some actions
```

```
store.dispatch(addTodo('Learn about actions'))
```

```
// Stop listening to state updates
```

```
unsubscribe()
```

```
import { connect } from "react-redux";
import { changeUsername } from "../store/message/actions";
```

```
export class UserSettings extends React.Component {
  handleUserNameChange = (e) => { this.props.changeUsername(e.currentTarget.value); }
  render() {
    return (<input value={this.props.username} onChange={this.handleUserNameChange}>)
  }
}
```

```
const mapStateToProps = state => {
  return { username: state.username };
};
```

```
// Map dispatch function into props
const mapDispatchToProps = dispatch => ({
  changeUsername: username => dispatch(changeUsername(username))
});
```

```
export default connect(mapStateToProps, mapDispatchToProps)(UserSettings);
```

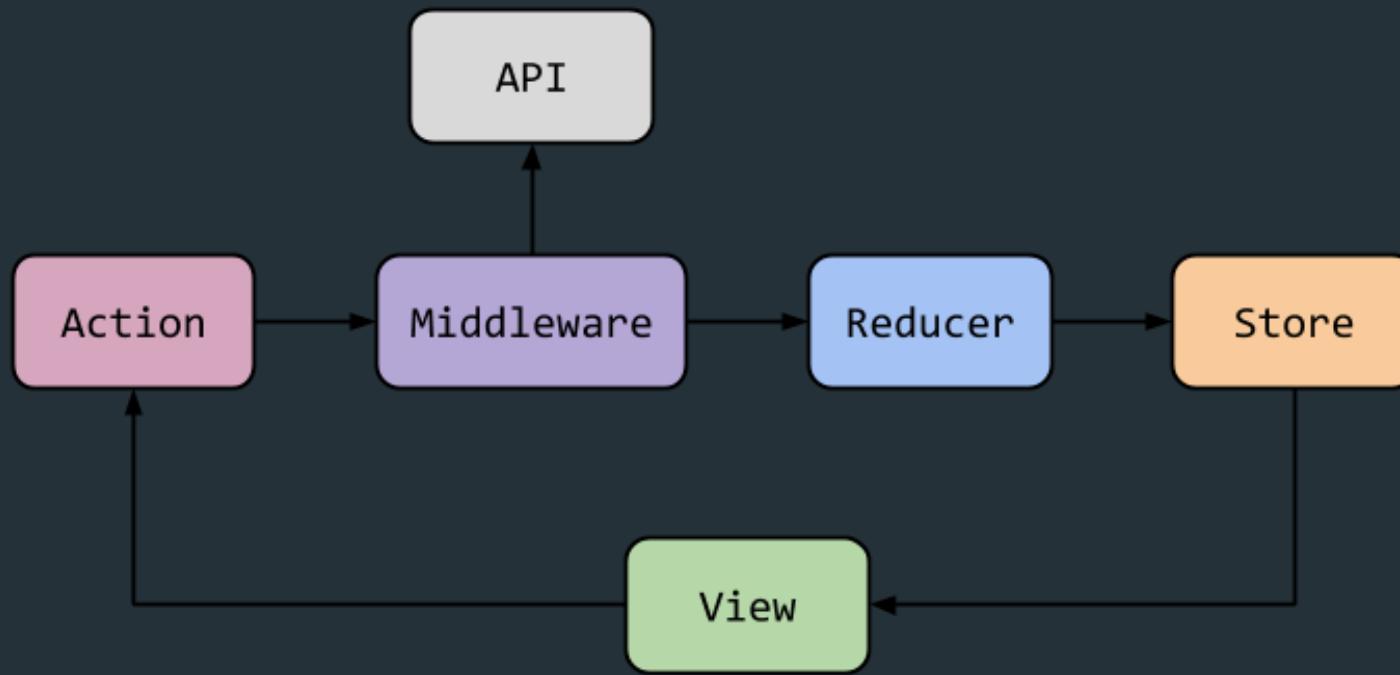
## Redux connect

# Store

## DEMO Redux mit DevTools

**Empfehlenswert zu lesen:** <https://levelup.gitconnected.com/componentdidmakesense-react-lifecycle-explanation-393dcb19e459>

## Middleware



## Middleware

```
const logger = store => next => action => {
  console.log('dispatching', action) let result = next(action)
  console.log('next state', store.getState()) return result
}
```

```
import { createStore, combineReducers, applyMiddleware } from 'redux'

const todoApp = combineReducers(reducers)
const store = createStore(
  todoApp,
  // applyMiddleware() tells createStore() how to handle middleware
  applyMiddleware(logger, crashReporter)
)
```

## Kaffe-Pause

15 Minuten



## Demo & Übung 2

Siehe uebungen/Uebung2.md

## Übung 3 (für die Schnellen)

Siehe uebungen/Uebung3.md



## Zusammenfassung

- React ist eine Library zum Aufbau
- Es animiert um Verwenden wiederverenbarer UI-Komponenten.
- Viele benutzen es als das V in MVC
- React abstrahiert den DOM vom Entwickler. Ermöglicht somit ein einfacheres Entwickeln und bessere Performance.
- React kann auch server-seitig gerendert werden und sogar native Apps für iOS, Android, Windows, Mac, Linux unterstützen.
- One-Way-Data-Flow. Reduziert Boilerplate und ist einfacher zu verstehen als traditionelles Data-Binding.

## Weiterführende Themen

- TypeScript
- React Hooks!
- Higher-Order Components (HOC), Higher-Order Functions
- Testing (enzyme, jest)
- React Native
- Store persistieren
- PWA
- redux-observable / redux sagas / (redux-thunk)
- Refs (React.createRef();)
- ... ;)



Vielen Dank  
Für Eure Aufmerksamkeit

Reto Weber & Luca Ingold