

NILS HARTMANN

Entwicklung von

Single-Page-Anwendungen

am Beispiel von React

Slides: <https://nils.buzz/spas-mit-react.pdf>

WORKSHOP | 8. MAI 2020 | @NILSHARTMANN

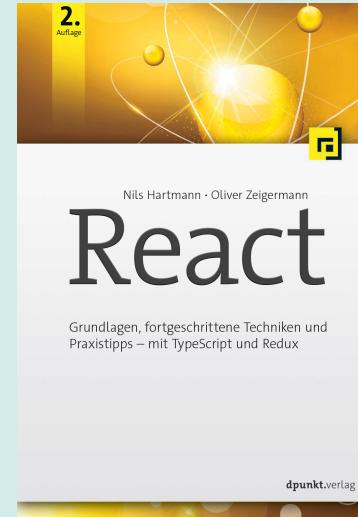
NILS HARTMANN

nils@nilshartmann.net

Freiberuflicher Entwickler, Architekt, Trainer aus Hamburg

Java
JavaScript, TypeScript
React
GraphQL

Trainings & Workshops



<https://reactbuch.de>

HTTPS://NILSHARTMANN.NET

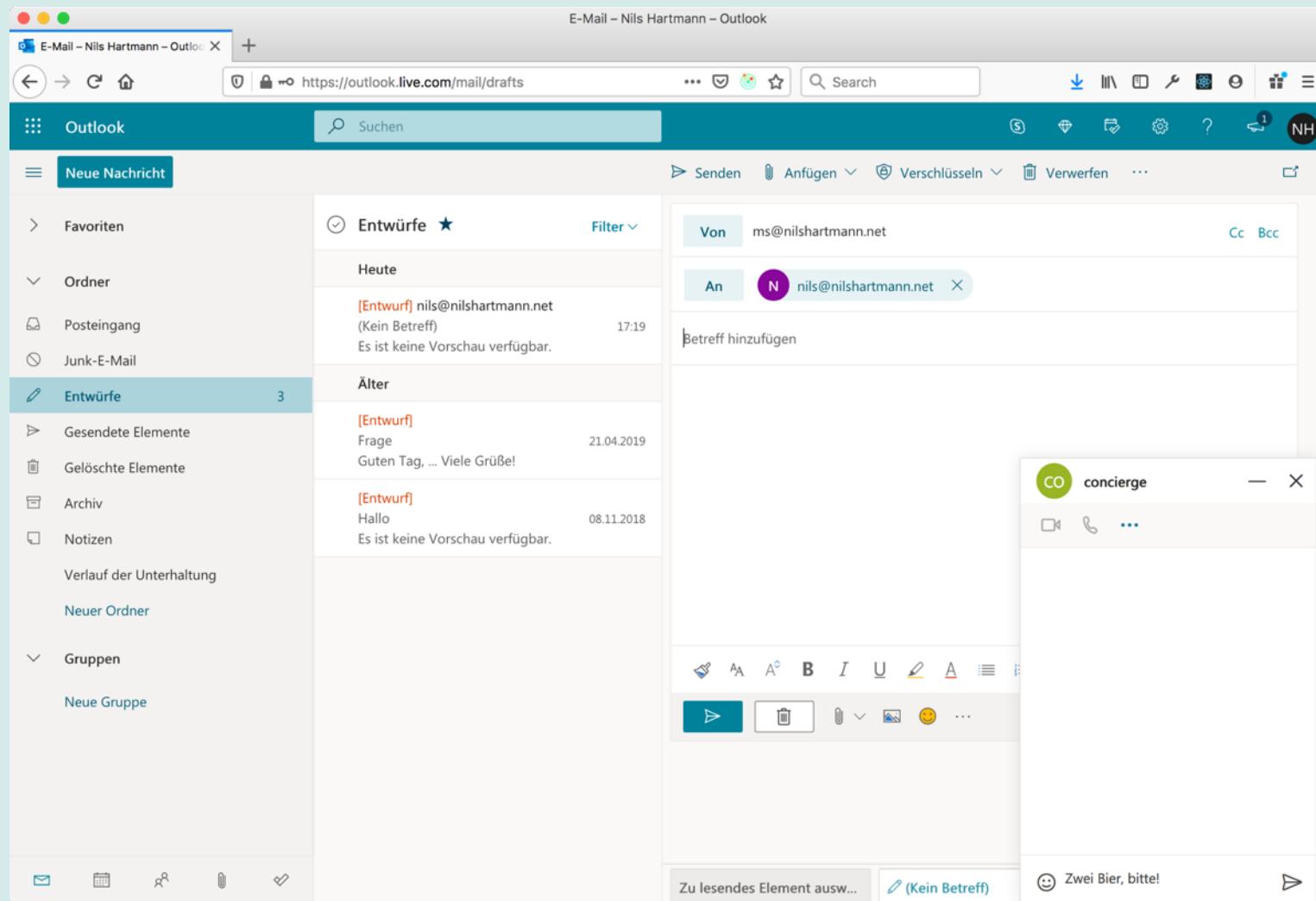
AGENDA

Mein Vorschlag

- Single-Page-Anwendungen (Hintergrund und Definition)
- Web-Anwendungen mit JavaScript (DOM API)
- Single-Page-Anwendungen am Beispiel React
- Frontend-Architekturen
- Umsetzung (Toolstack, Sprache)

Single- Page- Anwendungen

BEISPIELE



[HTTPS://OUTLOOK.LIVE.COM](https://outlook.live.com)

EXKURS: PROGRESSIVE WEB APP

Progressive Web Apps: Trennung zwischen Web und nativer App aufheben

- Responsive Design, um wie native Apps auszusehen
- **Offline-fähig durch Daten Caching**
- Können (auf dem Home-Screen) lokal installiert werden
- Erlauben den Empfang von Push-Notifications
- Können als SPA entwickelt werden, ist aber kein Muss
- "Progressive Enhancement": Features werden je nach Browser zur Verfügung gestellt

EXKURS: PROGRESSIVE WEB APP

Beispiel: <https://mobile.twitter.com>



„MODERNE“ WEB-ANWENDUNGEN

Aus Benutzersicht: bestes UI/UX

- Einheitliches Layout und Design
- Konsistentes Verhalten in der ganzen Anwendung
- Konsistente Darstellung der Daten
- **Gewohntes Verhalten von Desktop Anwendungen**
- **Kurze Reaktionszeiten**
- **Eventuell Offline-fähig**

Für Entwicklung

- Einfach und schnell
- Saubere und verständliche Architektur
- Wartbar auch bei großer und langlebiger Code-Basis

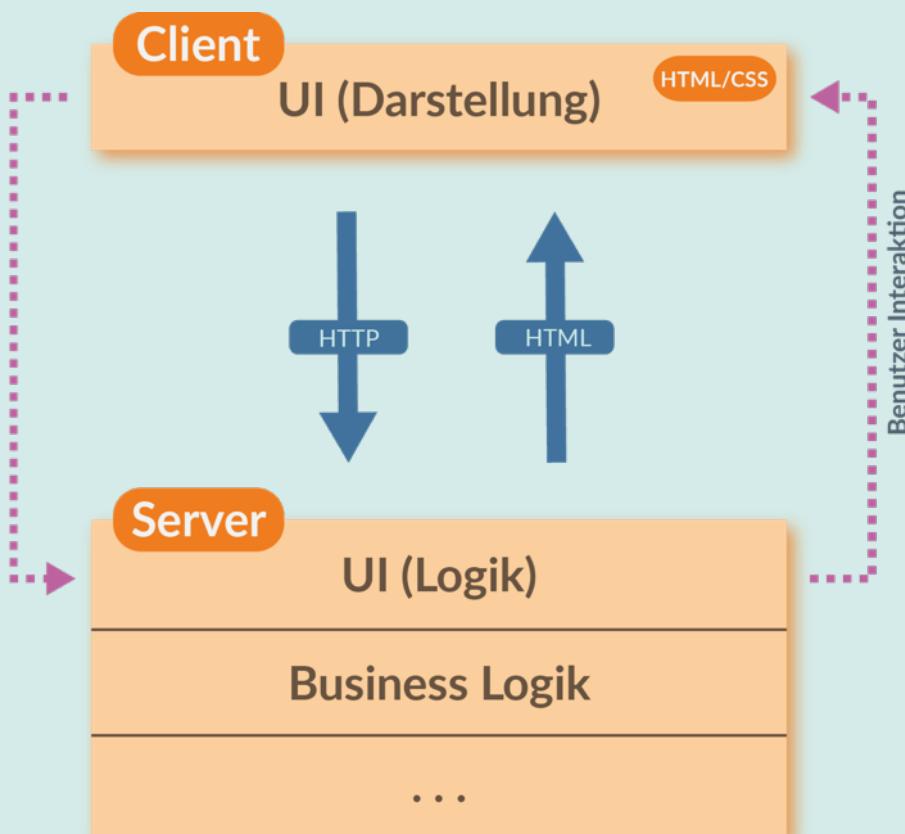
KLASSISCHE WEB-ANWENDUNGEN

RÜCKBLICK

RÜCKBLICK: KLASSISCHE WEB-ANWENDUNG

Klassische Web-Anwendung: Bekannte Technologie und Sprache

Zum Beispiel: Spring MVC, Java EE

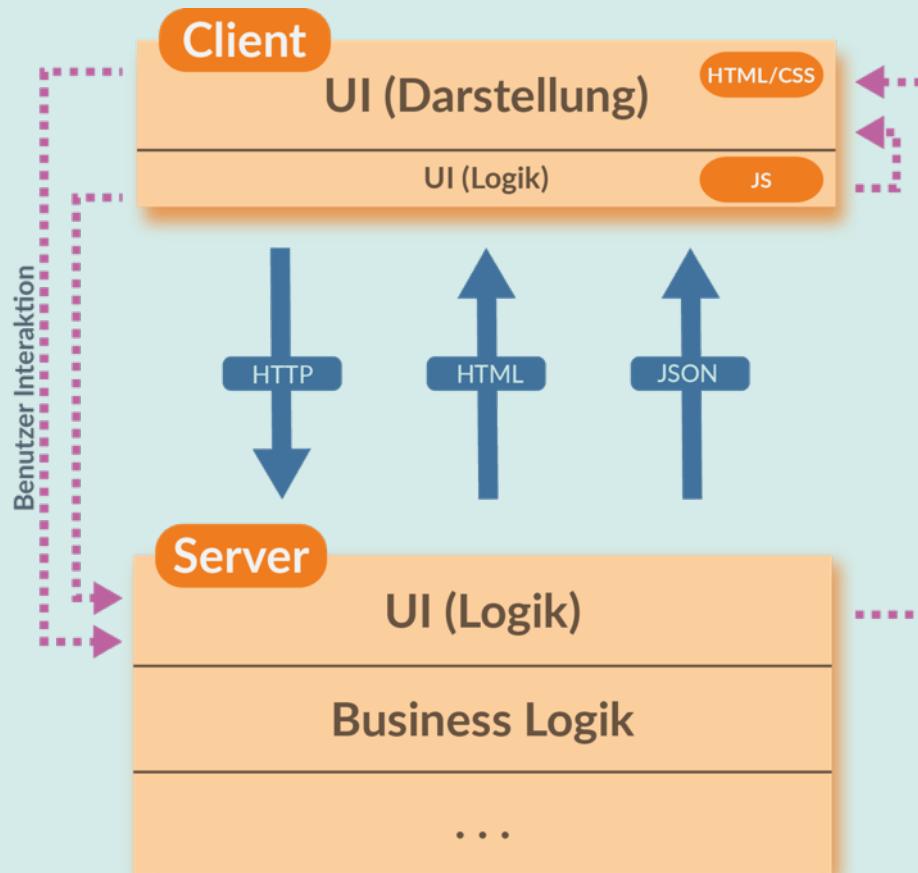


Konsequenzen: Server-Roundtrips für jede Kleinigkeit

RÜCKBLICK: KLASSISCHE WEB-ANWENDUNG

Klassische Web-Anwendung: ...mit ein „bisschen“ JavaScript

Zum Beispiel mit jQuery



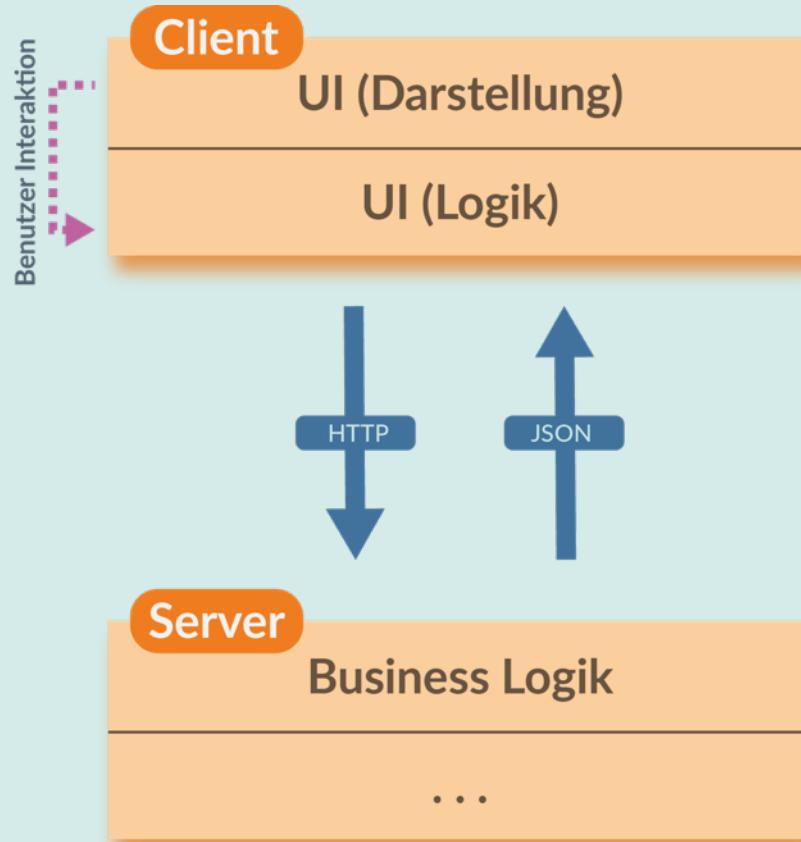
Konsequenzen: unsaubere Architektur, Wartungshölle

SINGLE-PAGE-ANWENDUNG

Moderne Single-Page-Anwendung: Saubere Architektur

Zum Beispiel: React

Schnelles Feedback



Konsequenzen: Kompletter Client in JavaScript...

SINGLE-PAGE-ANWENDUNGEN

Konsequenzen Single-Page-Anwendungen

- Die ganze Anwendung liegt im Client und ist in JavaScript geschrieben
- Der Client tauscht mit dem Server **Daten** aus (keine „fertige“ UI)
 - In der Regel JSON über REST- oder GraphQL APIs
- Der Client muss die Daten effizient an die Stellen in der UI geben
 - (im Gegensatz zur „fertigen“ Website)
- Sicherheit findet weiterhin auf dem Server statt!

SINGLE-PAGE-ANWENDUNGEN

Single Page?

- SPAs werden üblicherweise über eine HTML-Seite ausgeliefert
- Den Inhalt auf der Seite legen sie mit JavaScript fest
- Natürlich kann es für den Benutzer so „aussehen“, als ob es mehrere Seiten wären
- Auch der Back-Button funktioniert, wenn man möchte 😊

DOM API

WEB-ENTWICKLUNG MIT JAVASCRIPT

Web-Entwicklung mit JavaScript: Die DOM API

- Aus dem HTML-Code erzeugt der Browser das **Document Object Model (DOM)**
- Der Browser rendert und stellt den DOM dar
- Über JavaScript kann mit dem DOM interagiert werden:
 - es können Elemente gesucht werden (`getElementById`, ...)
 - man kann auf Veränderungen bzw. Ereignisse reagieren
 - Elemente können hinzugefügt, verändert, gelöscht werden (gilt auch für CSS)
- ➡ Wenn sich der DOM ändert, rendert der Browser neu und die Änderungen werden für den Benutzer sichtbar

Mozilla Developer Network (MDN)

- <https://developer.mozilla.org>
- Zentrales Dokumentationsportal rund um JavaScript und DOM APIs
 - Unterstützt auch von Google, Microsoft, W3C, ...



Grenzen der DOM API

- DOM API sehr mächtig, aber auch sehr "sperrig" und low-level
- 👉 <https://nilshartmann.github.io/react-talk/examples/dom-api.html>

Grenzen der DOM API

```
const title = document.createElement("h1");
const titleText = document.createTextNode("Hello World");
title.appendChild(titleText);

const greetButton = document.createElement("button");
greetButton.innerHTML = "Greet!";
greetButton.disabled = true;
greetButton.addEventListener("click", event => { ...

const greetInput = document.createElement("input");
greetInput.addEventListener("input", event => { ...

document.body.appendChild(title);
document.body.appendChild(greetInput);
document.body.appendChild(greetButton);
```

Code, der nur für die **Darstellung** der Hello-World-App benötigt wird
(Event-Listener ausgelassen)



SPA

Frameworks

am Beispiel

React

FRAMEWORKS

SPA-Frameworks

- spezialisiert auf das Bauen ganzer Anwendungen für den Browser
- Höhere Abstraktion als die DOM API
- Vier prominente Vertreter
 - React
 - Angular (vormals AngularJS)
 - Vue
 - Web Components (Standard)
- Frameworks und Tool-Stack mittlerweile relativ stabil und "Standard"

NO FRAMEWORKS

"Use the platform"

- Mit Standard JavaScript und Standard Brower API heute viel möglich
 - "Modernes" JavaScript mit Modulsystem etc
 - Sehr guter Browsersupport
 - WebComponents für Komponenten

NO FRAMEWORKS

"Use the platform"

- Mit Standard JavaScript und Standard Brower API heute viel möglich
 - "Modernes" JavaScript mit Modulsystem etc
 - Sehr guter Browsersupport
 - WebComponents für Komponenten

Platform (Standard) oder SPA-Bibliothek?

🤔 Was spricht für das eine, was für das andere?

NO FRAMEWORKS

Platform...

- Standard
- Eher langsame Weiterentwicklung
- Sehr zukunftssicher (es gibt praktisch keine API die aus dem DOM entfernt wurde)
- Gefällt mir das Programmiermodell?

SPA-Framework...

- unterschiedliche Programmiermodelle "für jeden Geschmack"
- Eher schnellere Weiterentwicklung
 - Variiert je nach Framework
 - Insgesamt aber recht stabil
- Zukunftssicher?

React

React

- <https://reactjs.org>
- Minimales API (?)
- Minimales Feature Set
 - 👉 Ihr könnt/müsst viele Entscheidungen selbst treffen
 - 👉 **Sehen wir später, auch in Bezug auf Eure Fragen**
- Bewusste Verstöße gegen „Best Practices“
- eher **funktionale Programmierung/API**

React: Aufteilung nach Komponenten

Klassische Aufteilung

Logik, Model
(Java, JS)



View
(HTML, Template)



Gestaltung
(CSS)



Aufteilung in fachliche Komponenten

Button



Eingabefeld



Label



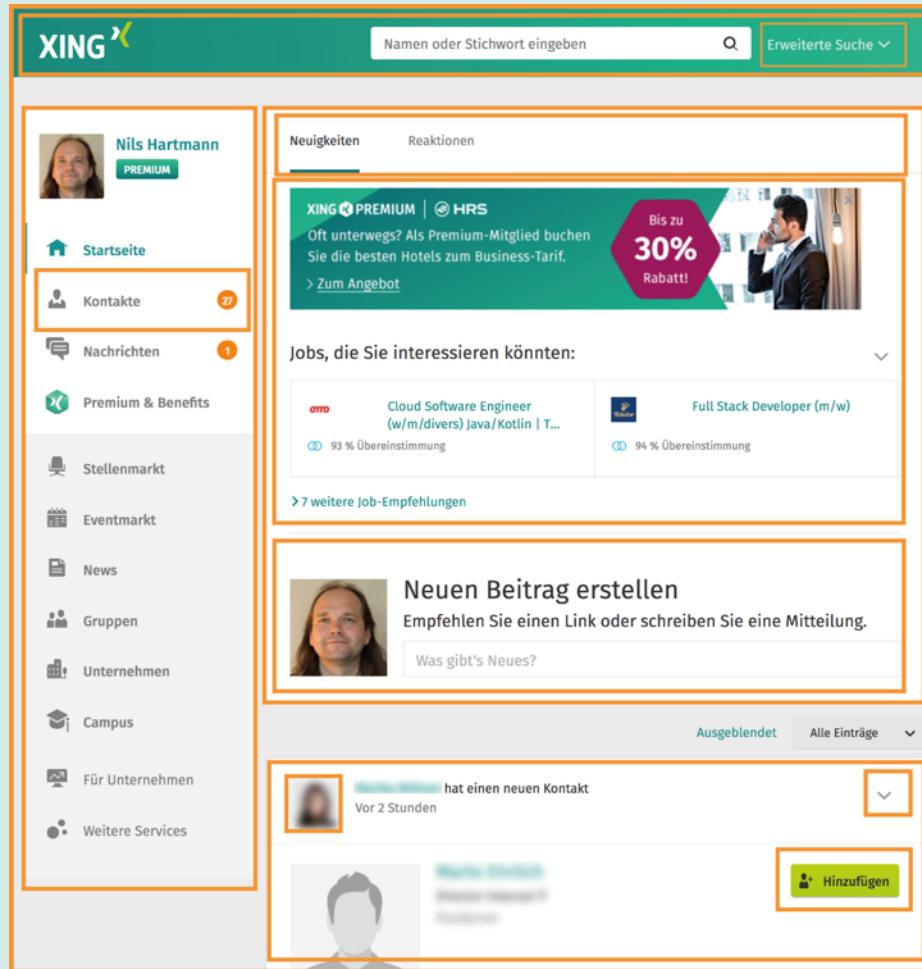
React: Komponenten

- bestehen aus **Logik und UI**
- **keine Templatesprache (?)**
- werden **deklarativ** beschrieben
- werden immer komplett gerendert (kein 2-Wege-Databinding)
- werden zu ganzen Anwendungen aggregiert

REACT

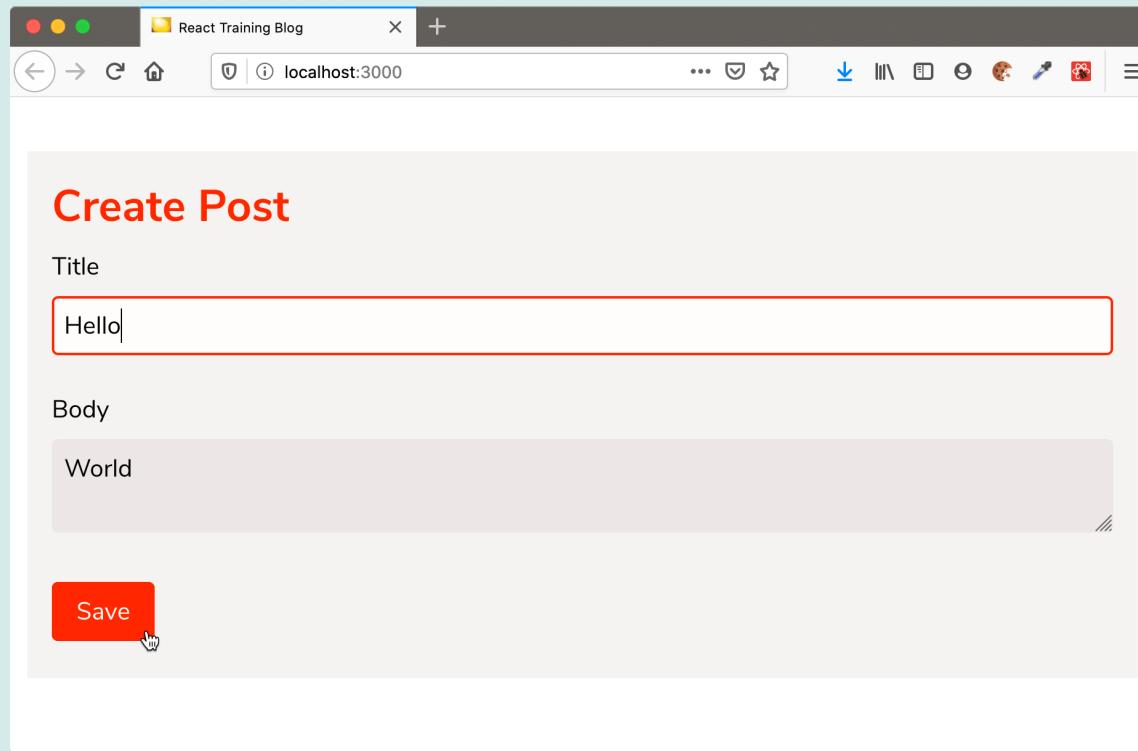
Komponenten werden zu Anwendungen aggregiert

Beispiel: XING



BEISPIEL: REACT

Live:
eine Mini-"Anwendung" vom Editor zum Browser



(blog-example/steps/1-hello-world)

BEISPIEL: REACT

React - Eine Komponente

```
import React from "react";

export default function Greeter(props) {
  const [greeting, setGreeting] = React.useState(props.initialGreeting);

  const buttonDisabled = greeting.length === 0;

  return (
    <div>
      <input
        value={greeting}
        onChange={e => setGreeting(e.target.value)}
      />

      <button disabled={buttonDisabled}
              onClick={() => props.onGreet(greeting)}>
        Greet!
      </button>
    </div>
  );
}
```

BEISPIEL: REACT

React - Eine Anwendung

```
import React from "react";

import {Header, Main, Footer} from "./Layout";
import LoginForm from "./forms/LoginForm";
import Greeter from "./forms/Greeter";

export default function App() {
  return (<>
    <Header><LoginForm /></Header>
    <Main>
      <Greeter initialGreeting ="Hello!" onGreet={greeting => ... } />
    </Main>
    <Footer>© Greeting Inc.</Footer>
  </>
);
}
```

BEISPIEL: REACT

React - Eine Komponente mit TypeScript

```
import React from "react";

type GreeterProps = {
    initialGreeting: string
    onGreet(greeting: string): void
}

export default function Greeter(props: GreeterProps) {
    // Rest unverändert,
    // TypeScript kann alle Typen herleiten
}
```

BEISPIEL: REACT

React – Mit TypeScript

Fehlermeldungen bei inkorrektener Verwendung

```
(alias) function Greeter(props: GreeterProps): JSX.Element
import Greeter

export
return
<He
<Ma
<Footer>© Greeting Inc.</Footer>
</>
);
}
```

Property 'onGreet' is missing in type '{ initialGreeting: string; }' but required in type 'GreeterProps'. ts(2741)

Greeter.tsx(5, 3): 'onGreet' is declared here.

Peek Problem No quick fixes available

<Greeter initialGreeting = "Hello!" />

BEISPIEL: REACT

React – Clientseitiges Routing

Eine Router-Komponente synchronisiert URL mit der Anwendung
"Deep-Links" funktionieren und gewohnte Navigation über Back-Button

```
import React from "react";
import { BrowserRouter, Router, Redirect } from "react-router";
import LoginForm from "./forms/Greeter";
import Greeter from "./forms/Greeter";

function App() {
  return ...
  <Main>
    <BrowserRouter>
      <Route path="/greet"> <Greet ... /> </Route>
      <Route path="/login"> <LoginForm /> </Route>
      <Route path="/"> <Redirect to="/greet"/> </Route>
      <Route> <NotFound /> </Route>
    </BrowserRouter>
  </Main>
  ...
}
```

BEISPIEL: REACT

Serverseitiges Rendering (SSR) – Anwendung auf dem Server vorrendern

- Schnelle First-Page-Impression
- Verbessertes SEO
- Vorschauen, z.B. in Social Media

```
import React from "react";
import ReactDOMServer from "react-dom/server";
import App from "./App";

ReactDOMServer.renderToString(<App />);
```

Code-Beispiel: Unittest mit Jest

```
sum.js    export function sum(a,b) {  
            return a+b  
        };  
  
sum.test.js import {sum} from '../sum.js';  
  
        test('sum of 2 and 2 is 4', function() {  
            expect(sum(2, 2)).toBe(4);  
        });  
  
        test('sum of 2 and 2 is not 3', function() {  
            expect(sum(2, 2)).not.toBe(3);  
        });
```

BEISPIEL: REACT

React – Testen einer Komponente

Testen ohne Browser möglich, headless im CI-Build

```
import React from "react";
import { render, fireEvent } from "@testing-library/react";
import BlogPost from "./BlogPost";

test("it should behave fine", () => {

  // Mock für Event-Handler
  const onNewPostHandler = jest.fn();

  // Komponente Rendern
  const { getByText, container } = render(
    <BlogPost post="..." onNewPost={onNewPostHandler} />
  );

  // Ereignis simulieren
  fireEvent.click(getByText("Add new Post"));

  // Ergebnis überprüfen
  expect(onNewPostHandler).toHaveBeenCalledTimes(1);

});
```

BEISPIEL: REACT

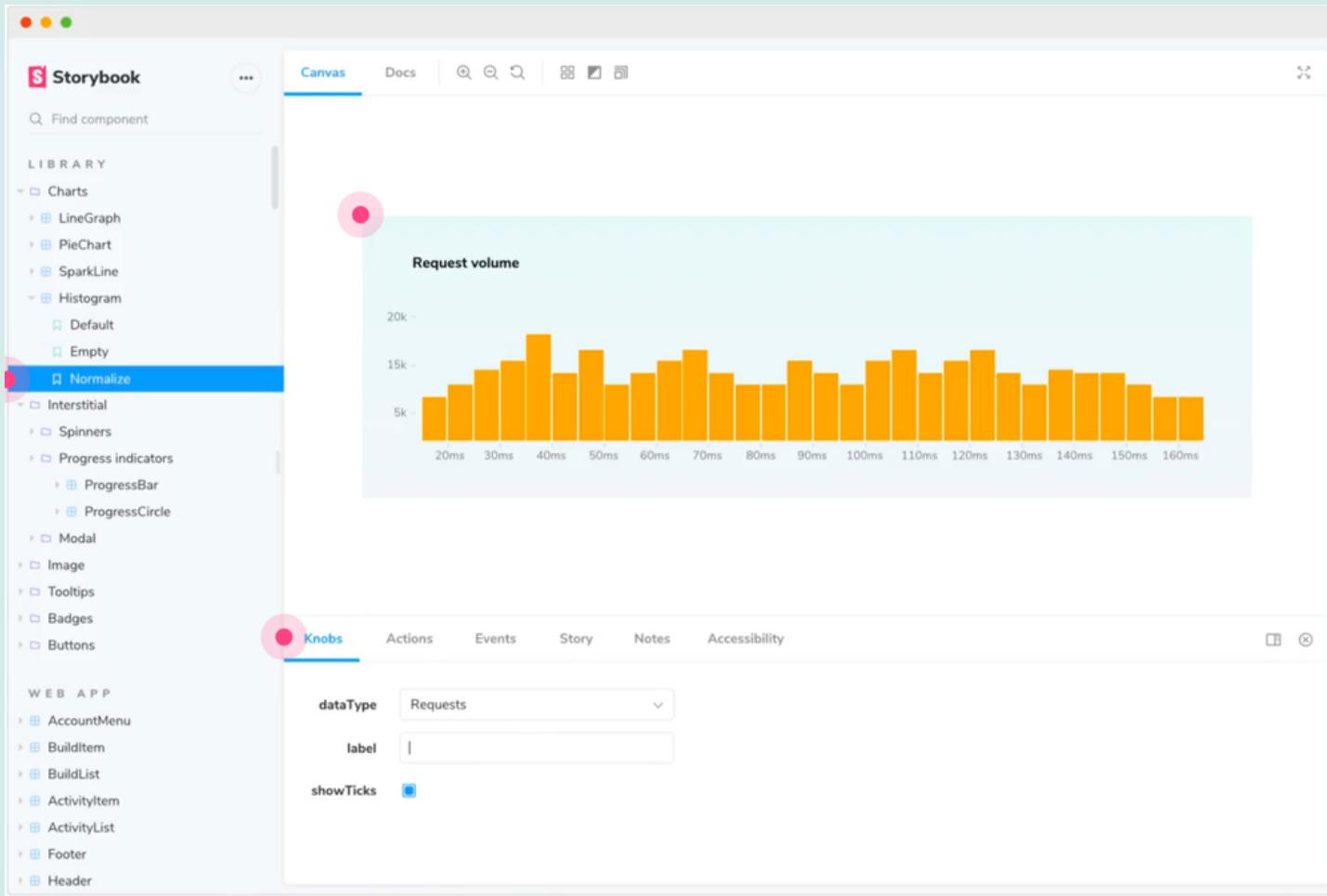
End-2-End-Tests im Browser

- TestCafe (<https://DevExpress.github.io/testcafe/>)
 - Cypress (<https://www.cypress.io/>)
 - Selenium natürlich weiterhin möglich
-
- Tests werden in JavaScript/TypeScript geschrieben
 - Guter Browser-Support (insb. TestCafe)
 - Gutes Debuggen und Fehleranalyse von Tests
 - Ausführung headless im CI-Build möglich

BEISPIEL: REACT

Storybook (<https://storybook.js.org/>)

- Testen, dokumentieren und präsentieren von Komponenten



ÜBER REACT HINAUS...

Styling

- Ihr müsst „ganz normales“ CSS schreiben
- Übliche Tools:
 - BEM (eher Methode): <https://getbem.com>
 - SASS/LESS (CSS Präprozessoren)
 - CSS Modules
- SASS + CSS Modules funktionieren mit create-react-app out-of-the-box

ÜBER REACT HINAUS...

Styling: CSS-in-JavaScript

- CSS wird direkt in JavaScript geschrieben
- Prominente Vertreter (mit React Support):
 - Styled Components (<https://styled-components.com/>)
 - CssInJs (<https://cssinjs.org>)
- Auch bei diesen Ansätzen wird „normales“ CSS für den Browser erzeugt

ÜBER REACT HINAUS...

Barrierefreiheit: Grundsätzlich möglich mit React

- Ihr seid verantwortlich für das HTML (aria-* -Elemente, Markup etc.)
- Vieles ist React-unabhängig
 - semantisches Markup
 - alt-Attribute für Bilder etc.
 - Browser Titel setzen
- ESLint Plug-in für React kann helfen
 - Findet klassische Fehler (vergessene aria-Attribute etc.)
 - <https://www.npmjs.com/package/eslint-plugin-jsx-a11y>
- react-testing-library versucht Entwicklung barrierefreier Apps zu forcieren
- Mehr zu React und Barrierefreiheit:
<https://reactjs.org/docs/accessibility.html>

ÜBER REACT HINAUS...

Barrierefreiheit: Grundsätzlich möglich mit React

- Ihr seid verantwortlich für das HTML (aria-* -Elemente, Markup etc.)
- Vieles ist React-unabhängig
 - semantisches Markup
 - alt-Attribute für Bilder etc.
 - Browser Titel setzen

ÜBER REACT HINAUS...

Barrierefreiheit: Grundsätzlich möglich mit React

- Ihr seid verantwortlich für das HTML (aria-* -Elemente, Markup etc.)
- Vieles ist React-unabhängig
 - semantisches Markup
 - alt-Attribute für Bilder etc.
 - Browser Titel setzen
- ESLint Plug-in für React kann helfen
 - Findet klassische Fehler (vergessene aria-Attribute etc.)
 - <https://www.npmjs.com/package/eslint-plugin-jsx-a11y>

ÜBER REACT HINAUS...

Barrierefreiheit: Grundsätzlich möglich mit React

- Ihr seid verantwortlich für das HTML (aria-* -Elemente, Markup etc.)
- Vieles ist React-unabhängig
 - semantisches Markup
 - alt-Attribute für Bilder etc.
 - Browser Titel setzen
- ESLint Plug-in für React kann helfen
 - Findet klassische Fehler (vergessene aria-Attribute etc.)
 - <https://www.npmjs.com/package/eslint-plugin-jsx-a11y>
- react-testing-library versucht Entwicklung barrierefreier Apps zu forcieren
- Mehr zu React und Barrierefreiheit:
<https://reactjs.org/docs/accessibility.html>

BEISPIEL: REACT

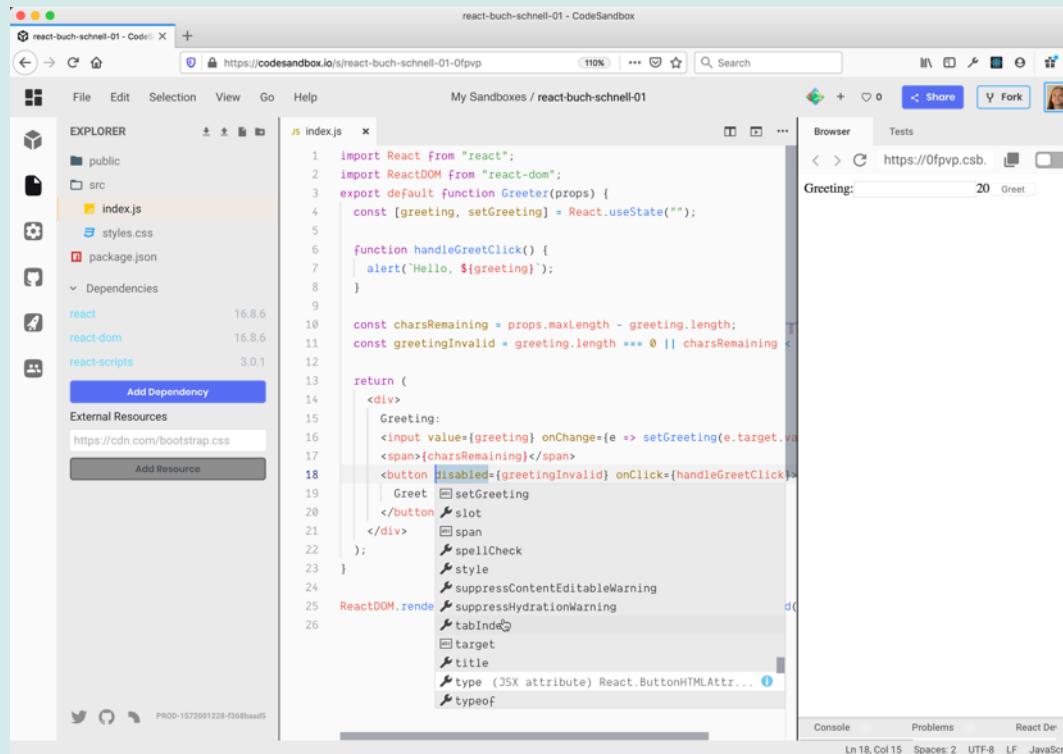
Fertiges Setup neuer Projekte: `create-react-app`

- <https://create-react-app.dev/>
- Erzeugt neues React Projekt
- Fertige Konfiguration für Build-Prozess, Testing, TypeScript, ...
 - Jest, react-testing-library
 - ESLint (inklusive a11y-Plug-in)
 - Produktionsbuild mit Minifizierung und diversen Optimierungen

MAL KURZ AUSPROBIEREN...

Online-Editor: <https://codesandbox.io/>

- Vollständiger JavaScript/TypeScript-Editor im Web
- Vorkonfigurierte Projekte für Angular, React, Vue
- Zum ausprobieren, ohne was installieren müssen



The screenshot shows the CodeSandbox web interface. On the left, the Explorer sidebar displays a project structure with files like index.js, styles.css, and package.json. The central area is a code editor showing index.js:

```
import React from "react";
import ReactDOM from "react-dom";
export default function Greeter(props) {
  const [greeting, setGreeting] = React.useState("");
  function handleGreetClick() {
    alert(`Hello, ${greeting}`);
  }
  const charsRemaining = props.maxLength - greeting.length;
  const greetingInvalid = greeting.length === 0 || charsRemaining < 0;
  return (
    <div>
      Greeting:
      <input value={greeting} onChange={e => setGreeting(e.target.value)}>
      <span>{charsRemaining}</span>
      <button disabled={greetingInvalid} onClick={handleGreetClick}>Greet</button>
    </div>
  );
}
ReactDOM.render(<Greeter maxLength={20} />, document.getElementById("root"));
```

To the right, there's a preview window titled "Browser" showing a simple form with an input field and a button. The input field has the placeholder "Greeting:" and contains "20". The button is labeled "Greet". Below the preview are tabs for "Console", "Problems", and "React Dev".

Deployment

- create-react-app erzeugt statische Assets (JavaScript, CSS, HTML) im Build, die auf einen regulären Web-Server (z.B. nginx) kopiert werden können
- Build-Prozess (inklusive Testen, Quality-Checks) etc. kann über gewohnte Tools erfolgen (GitLab, Jenkins, ...)

ÜBER REACT HINAUS...

React ist eine Bibliothek, kein Framework!

- zu vielen Dingen macht React keine Aussage, wie man es machen soll
- es gibt aber oftmals de-facto-Standard Wege
- React hat sehr wenig Anforderungen, die meisten existieren Browser APIs und JavaScript Bibliotheken lassen sich mit React verwenden

ÜBER REACT HINAUS...

Laden von Daten

Oft verwendet: **fetch** API (ersetzt Ajax-Requests mit XMLHttpRequest)

Browser Standard API

```
const response = await fetch("http://my-api.com/users");

if (response.ok) {
  const users = await response.body();
  // users ist hier ein JavaScript-Objekt mit den geladenen Daten
} else {
  // ...
}
```

ÜBER REACT HINAUS...

Speichern von Daten im Browser

- **Cache Storage API:** Cached den Netzwerkverkehr
 - z.B. für Offline-fähigkeit
- <https://developer.mozilla.org/en-US/docs/Web/API/Cache>

ÜBER REACT HINAUS...

Speichern von Daten im Browser

- **Cache Storage API:** Cached den Netzwerkverkehr
 - z.B. für Offline-fähigkeit
<https://developer.mozilla.org/en-US/docs/Web/API/Cache>
- **Local Storage / Session Storage:** Key-Value-Store (Strings) im Browser
 - z.B. User-Token
 - Session: gültig für einen Browser-Tab
 - Local Storage über mehrere Tabs und Fenster, auch über Browser Neustart
 - <https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage>

ÜBER REACT HINAUS...

Speichern von Daten im Browser

- **Cache Storage API:** Cached den Netzwerkverkehr
 - z.B. für Offline-fähigkeit
<https://developer.mozilla.org/en-US/docs/Web/API/Cache>
- **Local Storage / Session Storage:** Key-Value-Store (Strings) im Browser
 - z.B. User-Token
 - Session: gültig für einen Browser-Tab
 - Local Storage über mehrere Tabs und Fenster, auch über Browser Neustart
 - <https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage>
- **IndexedDB:** SQL-artiger Speicher im Browser
 - für größere strukturierte Daten, z.B. Formulare
 - große Datenmengen (variiert je Browser, genau wie Vorhaltezeit)
 - https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

👉 Daten bleiben in jedem Fall in **einem** Browser; kein Weiterverarbeiten in anderem Browser möglich

Toolchain (meine persönliche Empfehlung)

- create-react-app: Erzeugen Eures Projektes
- Editoren: IntelliJ IDEA (Ultimate) oder VS Code
- Sprache: TypeScript
- Zum Testen:
 - Unit und Integrationtests: Jest und react-testing-library (defacto Standard)
 - End2End-Tests: TestCafe oder Cypress
 - Code Analyse: ESLint, ggf. SonarCube
 - Obfuscation: ?
- Kontinuierlicher Build wie gewohnt (Jenkins etc)

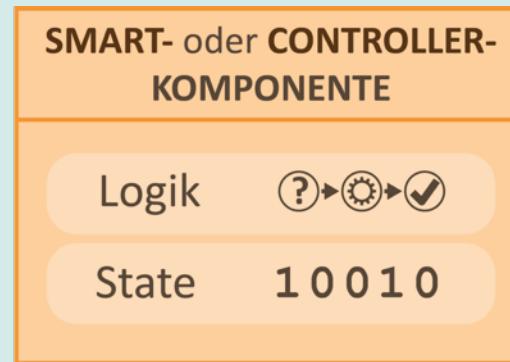
Frontend- Architektur

Frameworks: wir bauen Komponenten



ARCHITEKTUR

Architektur-Muster: Smart- und Dumb-Komponenten



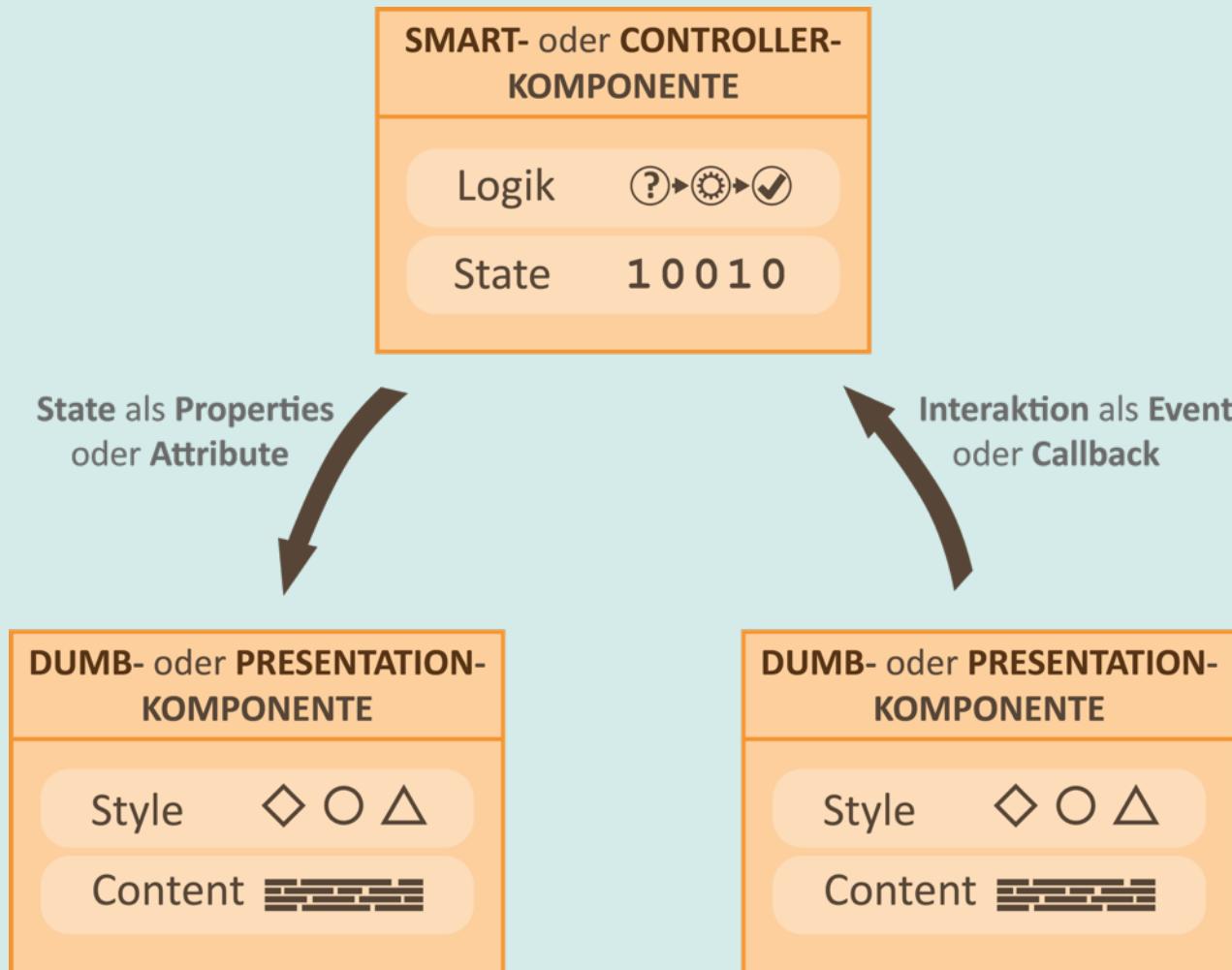
ARCHITEKTUR

Architektur-Muster: Smart- und Dumb-Komponenten



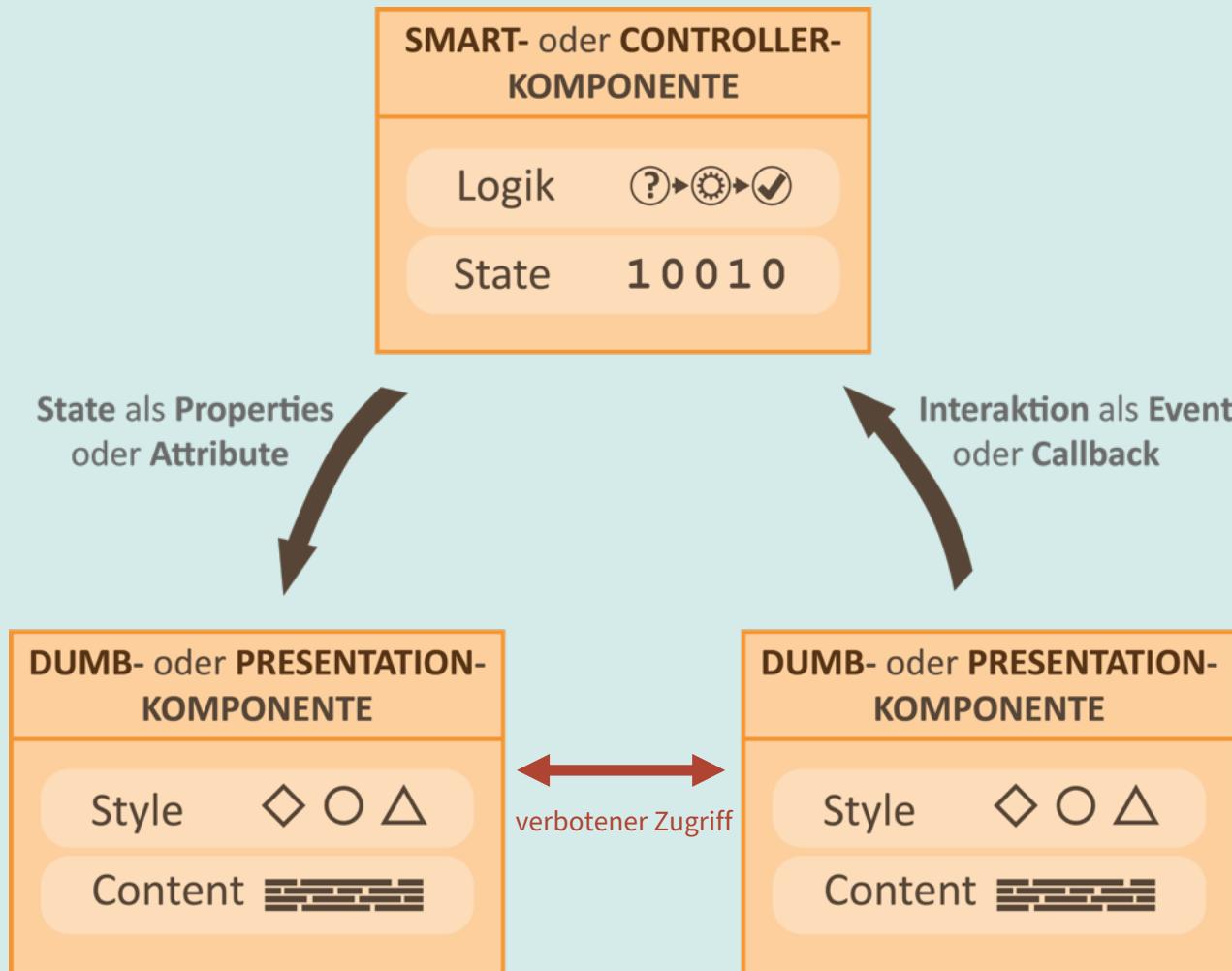
KOMPOSITION VON KOMPONENTEN

Architektur-Muster: Smart- und Dumb-Komponenten



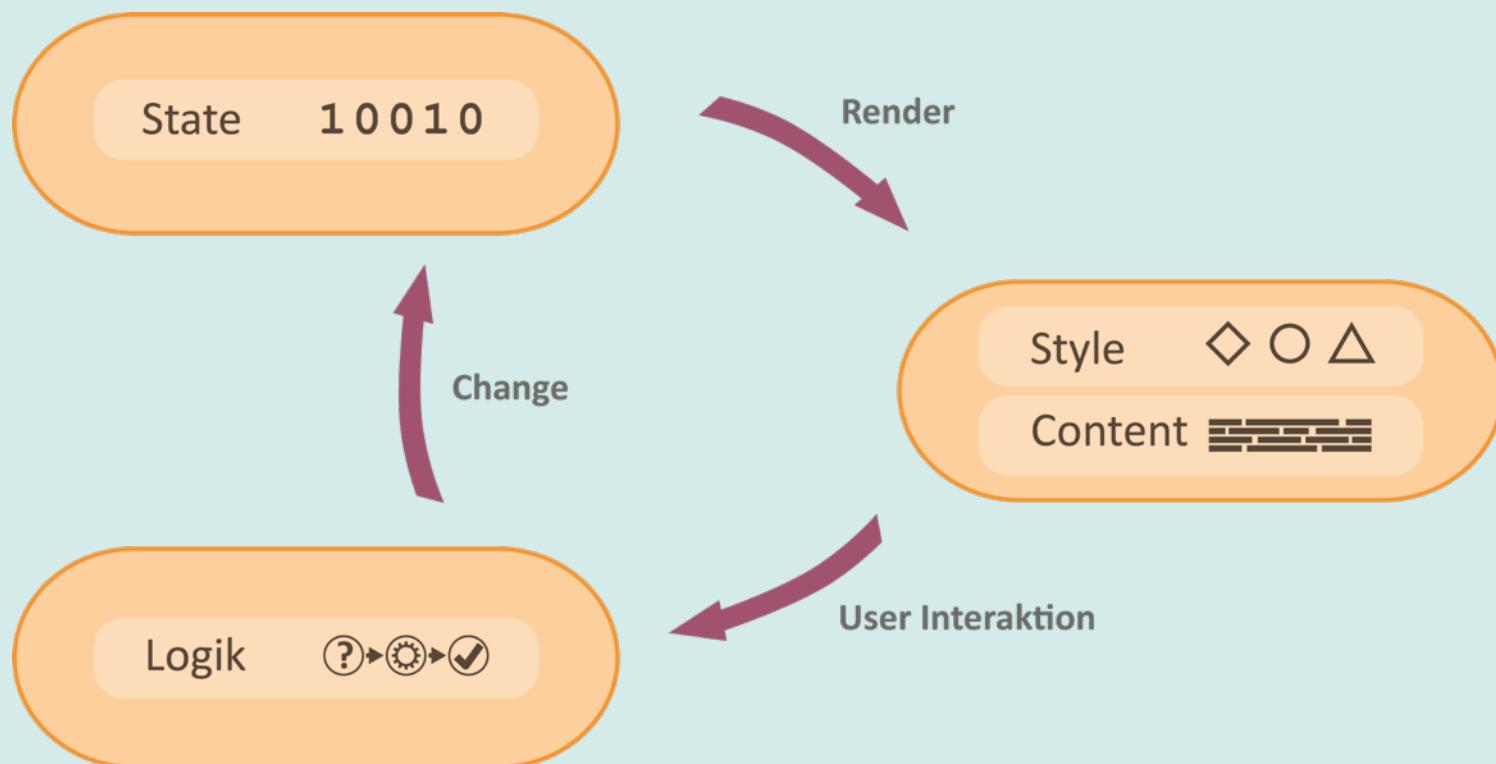
KOMPOSITION VON KOMPONENTEN

Architektur-Muster: Smart- und Dumb-Komponenten



ARCHITEKTUR MUSTER

Reaktives Verhalten: Nachvollziehbarer Renderzyklus



KOMPONENTEN

Grenzen dieser Architektur

- Große Komponenten entstehen
- Viele Controller-Komponenten => wieder Chaos
- Starke Abhängigkeit vom gewählten UI-Framework

KOMPONENTEN

Grenzen dieser Architektur

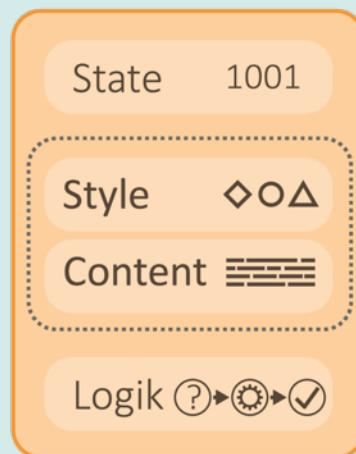
- Große Komponenten entstehen
- Viele Controller-Komponenten => wieder Chaos
- Starke Abhängigkeit vom gewählten UI-Framework

Externes Statemangement

- Verschiebt Logik und Zustand aus der Komponente
- Prominente Vertreter: Redux, MobX

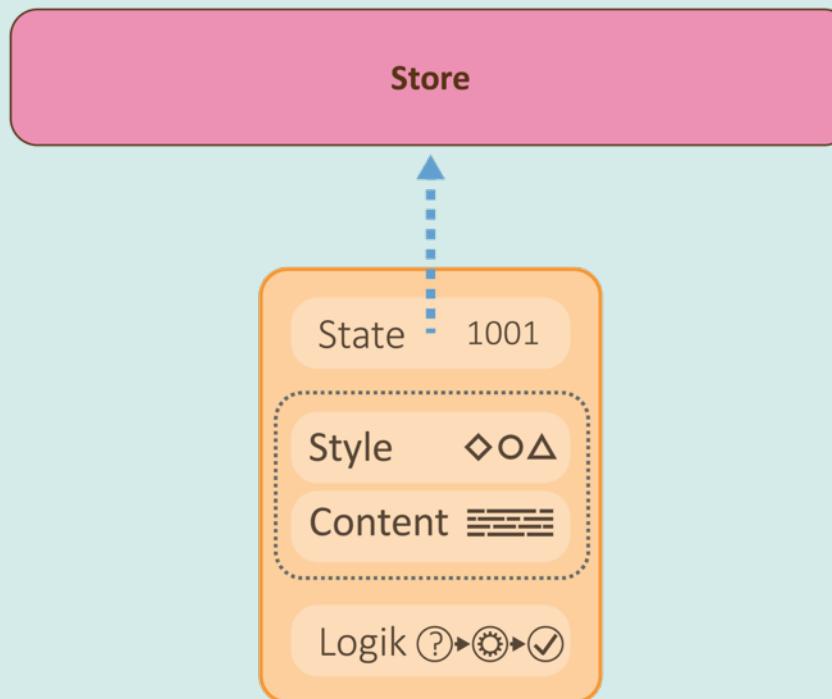
KOMPONENTEN

Externes Statemanagement



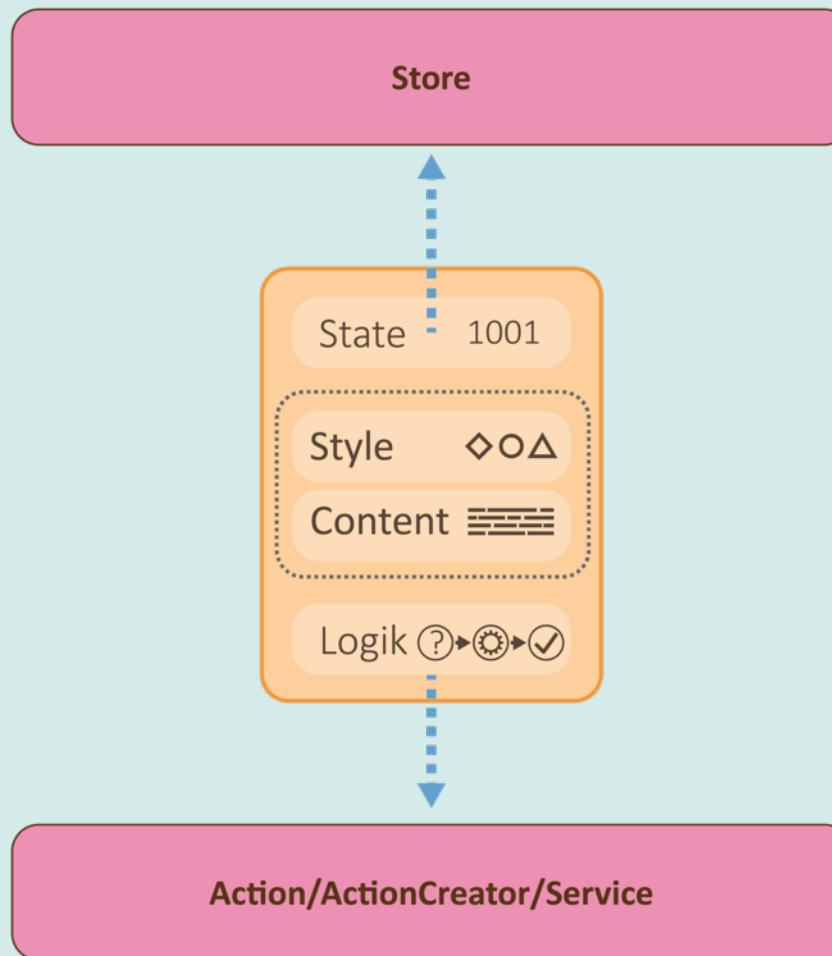
KOMPONENTEN

Externes Statemanagement



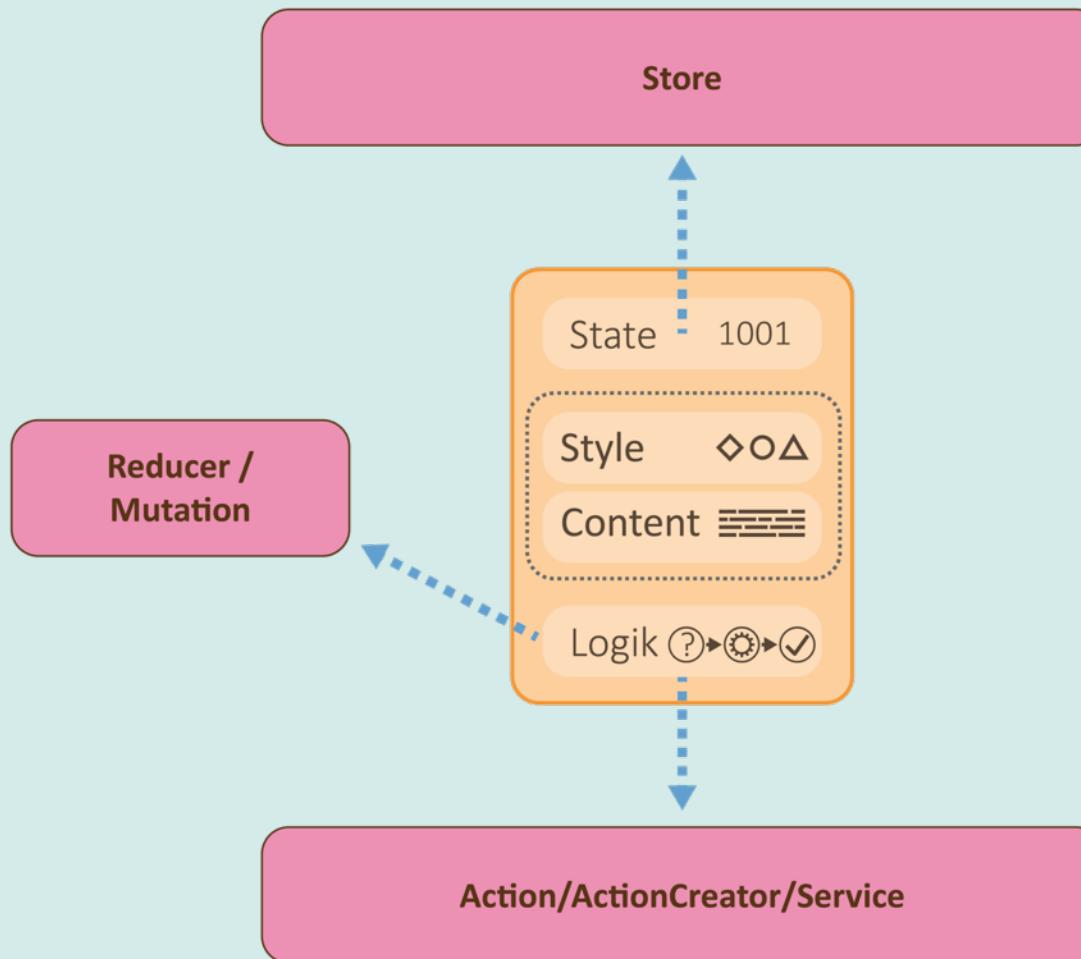
KOMPONENTEN

Externes Statemanagement



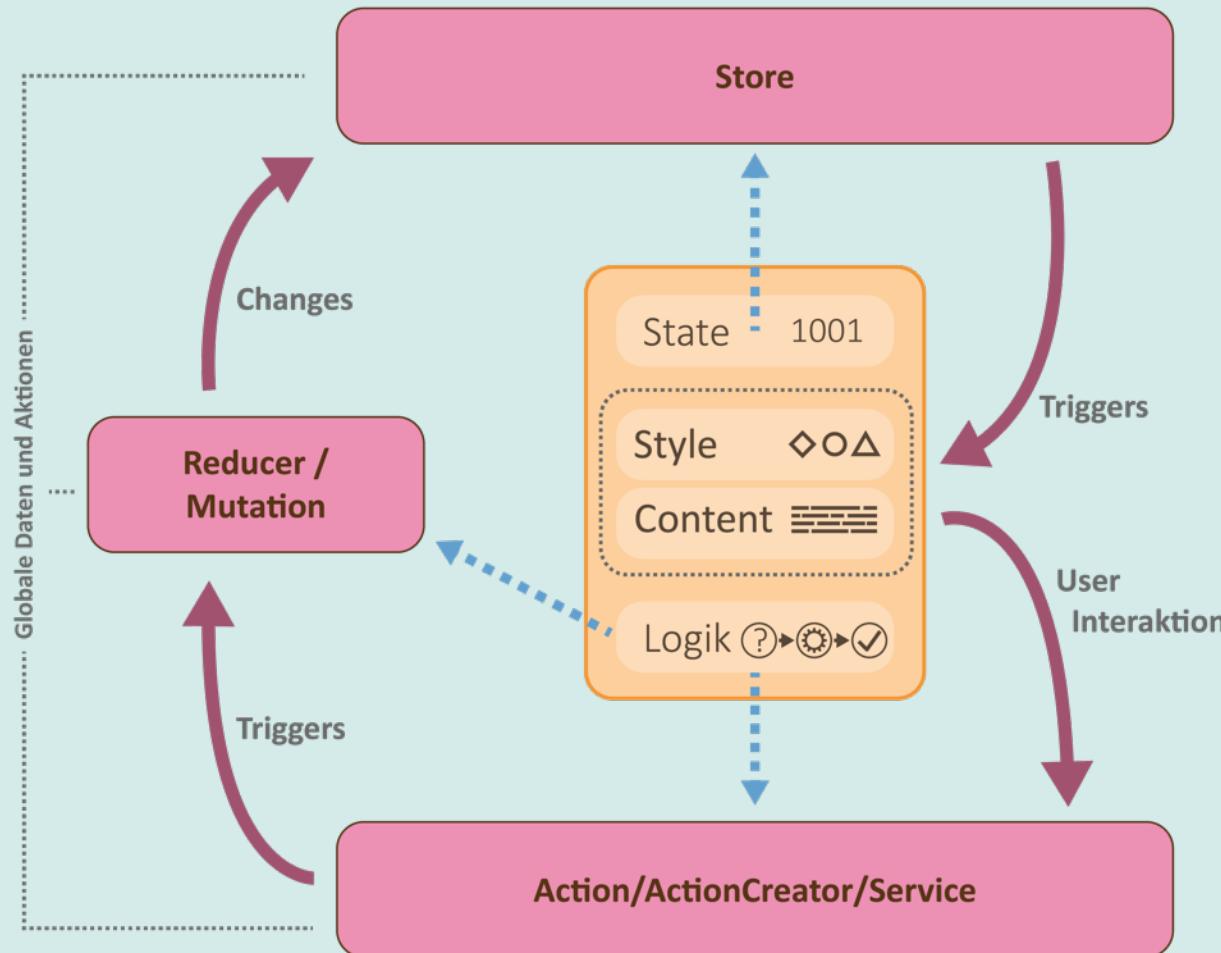
KOMPONENTEN

Externes Statemanagement



KOMPONENTEN

Externes Statemanagement



KOMPONENTEN

Grenzen dieser Architektur

- Sehr große Anwendungen
- Mehrere Teams
- Unabhängige Anwendungsteile, unabhängige Deployments
erwünscht

Toolstack:

Sprache

&

Typische Werkzeuge

JavaScript: Nur Sprache

- Keine zentrale Organisation (abgesehen vom Sprachstandard)
- Keine Standard Bibliothek (nur minimal)
- Kein Typ-System
- Kein Compiler
- Keine einheitliche Laufzeitumgebung
- ~~Kein Modul-System~~
- Kein Threading

- Wenig Konventionen (keine klassische Maven Projektstruktur z.B.)

ZUM VERGLEICH: JAVA

Java: Bringt alles mit was wir zum Entwicklung brauchen

Tools, Bibliotheken, Laufzeitumgebung, ...

Alles aus einer Hand

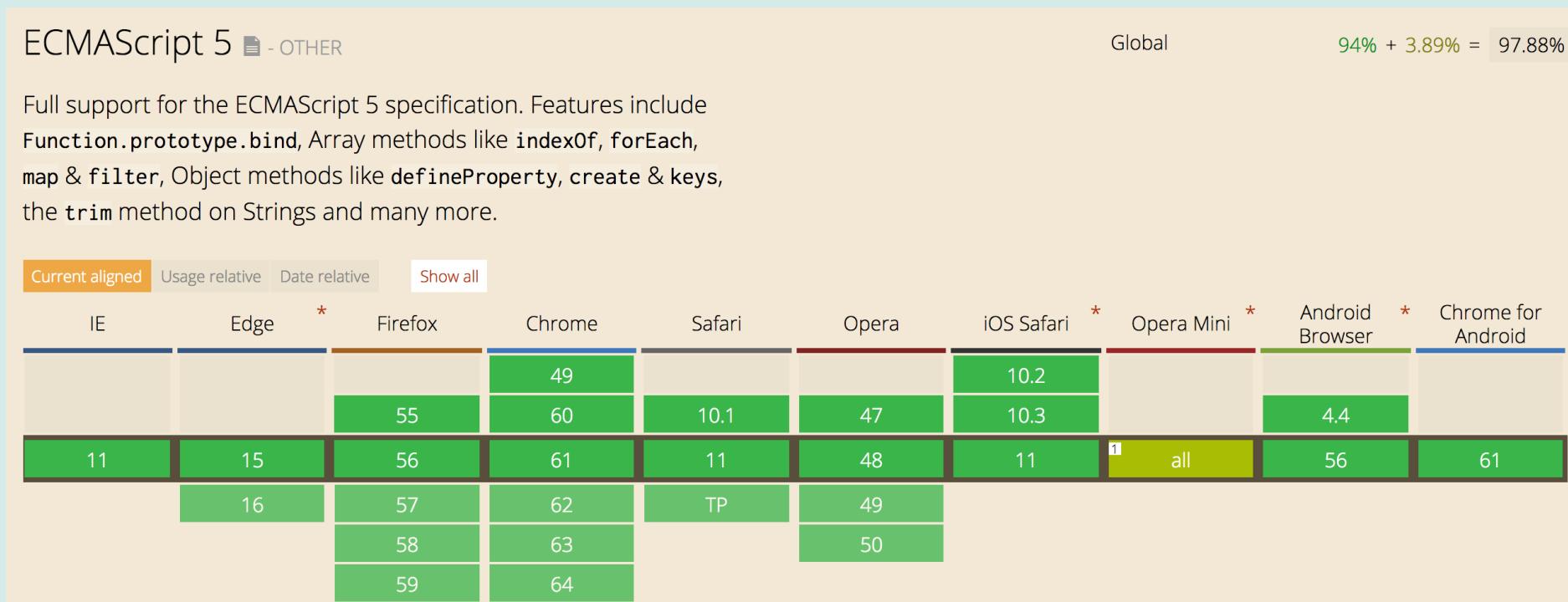
Java Language											
Java Language	java	javac	javadoc	jar	javap	jdeps	Scripting				
Tools & Tool APIs	Security	Monitoring	JConsole	VisualVM	JMC	JFR					
Deployment	JPDA	JVM TI	IDL	RMI	Java DB	Deployment					
User Interface Toolkits	Internationalization		Web Services		Troubleshooting						
	Java Web Start			Applet / Java Plug-in							
Integration Libraries	JavaFX										
	Swing		Java 2D		AWT	Accessibility					
Other Base Libraries	Drag and Drop		Input Methods		Image I/O	Print Service	Sound				
	IDL	JDBC	JNDI	RMI	RMI-IIOP		Scripting				
lang and util	Beans	Security		Serialization		Extension Mechanism					
	JMX	XML JAXP		Networking		Override Mechanism					
Base Libraries	JNI	Date and Time		Input/Output		Internationalization					
	lang and util										
Java Virtual Machine	Math	Collections		Ref Objects		Regular Expressions					
	Logging	Management		Instrumentation		Concurrency Utilities					
Java HotSpot Client and Server VM	Reflection	Versioning		Preferences API		JAR	Zip				
	Java HotSpot Client and Server VM										

Grafik: <https://docs.oracle.com/javase/8/docs/index.html>

DIE SPRACHE

ECMAScript 5: Veröffentlicht 2009

- Unterstützung von praktisch allen Browsern
- Die "Referenz"-Version



- **JavaScript: Implementierung | ECMAScript: Spezifikation**

JavaScript: Versionsübersicht

- **ES5** erschienen 2009, "Referenzversion", die von allen Browsern (inkl. IE) unterstützt wird
- **ES6, ECMAScript 2015**, großes Release von 2015, sehr viele Neuerungen
 - "modernes JavaScript"
- **ES7, ES8, ES9 / ES2016, ES2017, ES2018, ...**: jährliche Releases mit weniger Neuerungen
- **ES.Next**: "symbolischer" Name für das jeweils nächste Release

Browsersupport für neue JavaScript-Versionen:

<http://kangax.github.io/compat-table/es6/>

Modernes JavaScript: Releases ab 2015

```
class Person {  
  
  #name;  
  
  constructor(name) {  
    this.#name = name;  
  }  
  
  sayHi() { return `Hello, ${this.#name}`  
}  
export default Person;  
  
const person = new Person("Christian");  
const greeting = person.sayHi();
```

ES6 SUPPORT

Feature name	Current browser	iOS 11	iOS >=10.3	iOS <11	SF 11	SF 10.1	Node >=8.7	Node <9 [5]	CH 62*	CH 61*	FF 56	Node >=6.5	Edge 16	Edge 15	XS6	FF 52 ESR	Babel + core-js ^[2]	JXA	Type-Script + core-js	Traceur	Node 4 ^[5]	Closure	DUK 2.2	es6-shim	IE 11
Syntax																									
default function parameters	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	6/7	4/7	0/7	5/7	4/7	0/7	5/7	0/7	0/7	0/7	
rest parameters	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	3/5	0/5	4/5	4/5	0/5	2/5	0/5	0/5	0/5	
spread(...)	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	13/15	11/15	4/15	15/15	0/15	12/15	0/15	0/5	0/15	
object literal extensions	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	5/6	6/6	6/6	6/6	6/6	4/6	4/6	0/5	0/6	
for...of loops	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	7/9	9/9	8/9	3/9	9/9	7/9	6/9	0/9	0/9	
octal and binary literals	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	2/4	4/4	4/4	4/4	2/4	0/4
template literals	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	4/5	5/5	5/5	3/5	4/5	5/5	5/5	0/5	0/5	0/5
RegExp "y" and "u" flags	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	3/5	0/5	0/5	3/5	0/5	0/5	0/5	0/5	0/5	0/5
destructuring declarations	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	21/22	21/22	21/22	19/22	15/22	20/22	0/22	20/22	0/22	0/2	0/22	
destructuring assignment	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	23/24	24/24	21/24	19/24	23/24	0/24	21/24	0/24	0/4	0/24	0/24	
destructuring parameters	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	23/24	23/24	21/24	21/24	18/24	16/24	19/24	0/24	18/24	0/24	0/4	0/24	
Unicode code point escapes	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	1/2	1/2	1/2	1/2	2/2	1/2	2/2	0/2	0/2	
new.target	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	
Bindings																									
const	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	14/16	10/16	14/16	14/16	9/16	14/16	2/16	0/6	12/16		
let	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	10/12	0/12	10/12	10/12	6/12	10/12	0/12	0/2	10/12		
block-level function declaration ^[14]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	Yes	No	Yes	
Functions																									
arrow functions	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	12/13	13/13	9/13	0/13	9/13	11/13	9/13	10/13	0/13	0/3	0/13		
class	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	19/24	18/24	19/24	17/24	0/24	13/24	0/24	0/4	0/24		
super	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	4/8	7/8	7/8	7/8	0/8	6/8	0/8	0/3	0/8	
generators	27/27	27/27	27/27	27/27	27/27	27/27	27/27	27/27	27/27	27/27	27/27	27/27	27/27	25/27	24/27	0/27	0/27	24/27	20/27	16/27	0/27	0/7	0/27		
Built-ins																									
typed arrays	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	45/46	46/46	45/46	0/46	43/46	0/46	20/46	0/6	16/46		
Map	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	18/19	19/19	18/19	19/19	14/19	17/19	14/19	0/19	15/19	8/19	
Set	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	18/19	19/19	18/19	19/19	14/19	17/19	14/19	0/19	15/19	8/19	
WeakMap	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	11/12	11/12	12/12	11/12	6/12	11/12	9/12	0/12	0/2	6/12		
WeakSet	11/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11	10/11	10/11	11/11	10/11	5/11	10/11	8/11	0/11	0/1	0/11		
Proxy ^[19]	34/34	34/34	34/34	34/34	34/34	34/34	34/34	34/34	34/34	34/34	34/34	34/34	34/34	34/34	0/34	0/34	0/34	0/34	0/34	0/34	15/34	0/4	0/34		
Reflect ^[20]	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20	16/20	20/20	15/20	0/20	15/20	0/20	0/20	14/20	14/20	0/20	
Promise	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	0/8	8/8	8/8	4/8	7/8	7/8	0/8	7/3	0/8	
Symbol	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	9/12	12/12	8/12	4/12	10/12	2/12	12/12	2/2	0/12	0/12	
well-known symbols ^[21]	26/26	26/26	26/26	26/26	26/26	26/26	26/26	26/26	26/26	26/26	26/26	26/26	26/26	26/26	15/26	22/26	15/26	1/26	3/26	1/26	1/26	0/6	0/26		
Built-in extensions																									
Object static methods	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	3/4	4/4	3/4	3/4	4/4	2/4	4/4	2/4	1/4		
function "name" property	17/17	17/17	17/17	17/17	17/17	17/17	17/17	17/17	17/17	17/17	17/17	17/17	17/17	17/17	8/17	14/17	3/17	0/17	6/17	0/17	0/7	0/17	0/17	0/17	
String static methods	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	1/2	2/2	2/2	0/2	
String.prototype methods	10/10	10/10	10/10	10/10	10/10	10/10	10/10	10/10	10/10	10/10	10/10	10/10	10/10	10/10	9/10	10/10	9/10	8/10	10/10	7/10	7/0	0/10			
RegExp.prototype properties	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	5/6	6/6	0/6	0/6	1/6	1/5	0/6			
Array static methods	11/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11	8/11	9/11	10/11	9/11	5/11	7/1	0/11	0/11	0/11		

COMPILER: WENN DER BROWSER SUPPORT NICHT AUSREICHT

Babel: Der "Standard" Compiler

- <https://babeljs.io/>
- Compiliert ES2015+ nach ES5
- Durch Plug-ins erweiterbar (eigenes Ökosystem ...)
- Unterstützt auch experimentelle Sprachfeatures

TypeScript: Sprache von Microsoft inklusive Compiler

- <http://www.typescriptlang.org/>
- Typ-System für JavaScript
- Bringt Sprach-Erweiterungen mit
 - z.B. private Felder, Enum

HINTERGRUND: POLYFILLS

- Compiler / Transpiler übersetzen "nur" die Sprache
 - Retrotranslator ☺
- Polyfills sind JS Bibliotheken, die fehlende APIs implementieren
 - Stellen Abwärtskompatibilität für ältere Browser her
 - Zum Beispiel LocalStorage oder HTML5 History API

Browser: Der Klassiker

- Nahezu alle Browser implementieren ES5
- JavaScript-Support wird besser und einheitlicher
- Wettbewerb um beste Developer Tools

NodeJS: Serverseitiges JavaScript

- Basiert auf der JS Engine V8 von Chrome
 - Ermöglicht zusätzlich Zugriff auf File-System, Konsole etc
- **Grundlage auch für diverses Tooling**
 - Package Manager, Build, Test, ...

Unterschiedliche Anforderungen für den Build!

JavaScript Modul Systeme

- **CommonJS**: NodeJS Modul-System ("require ..." / "module.exports")
- **AMD**: Asynchrone Module (für Browser)
- **ES2015** spezifiziert natives Modul System

"Module" sind sehr fein-granular (z.B. eine Datei)

- Nicht direkt vergleichbar mit Java9/OSGi Modulen

```
module.exports = function isObject(val) {  
    return val != null && typeof val === 'object' && Array.isArray(val) === false;  
};
```

Beispiel: isobject Modul

Empfehlung: ES2015 Modulsystem

- In neuen Projekten mit import/export beginnen

Editoren / IDEs

- Die meisten Editoren bzw. IDEs haben guten bis sehr guten JavaScript Support
- Beliebt:
 - IntelliJ IDEA (Ultimate) oder Webstorm
 - Visual Studio Code (<https://code.visualstudio.com/>)

Statische Code-Analyse: ESLint (<https://eslint.org/>)

- Findet typische JavaScript Programmierfehler
- Achtet auf Einhaltung von Konventionen (z.B. Semikolon ja/nein)
- Kann in den CI-Build eingebunden werden

The screenshot shows a code editor window with a tab labeled "example.js". The code contains a function "identical" that checks if two variables are equal. ESLint has flagged several issues:

```
1
2  function identical(a, b) {
3    if (a == b) {
4      return
5      true;
6    }
7 }
```

The code editor highlights errors with red squiggly lines under "==" and "true;". The ESLint panel at the bottom shows three errors:

- [eslint] Expected '===' and instead saw '=='. (eqeqeq) (3, 8)
- [eslint] Expected an assignment or function call and instead saw 'true;'. (eqeqeq) (5, 3)
- [eslint] Unreachable code. (no-unreachable) (5, 3)

TypeScript (<https://typescriptlang.org>)

- Mächtiges Typ-System für JavaScript von Microsoft
- Baut auf JavaScript auf, jeder JavaScript-Code auch TypeScript-Code
- Compiler für ES6 Code nach ES5

TYPESCRIPT

TypeScript (<https://typescriptlang.org>)

```
// Person.ts
export default class Person {
    private name: string;           // Typ-Angaben und Sichtbarkeiten

    constructor(name: string) {
        this.name = name;
    }

    sayHi() { return `Hello, ${this.#name}` } // Return-Typ wird abgeleitet
}
```

TypeScript (<https://typescriptlang.org>)

```
// Person.ts
export default class Person {
    private name: string;           // Typ-Angaben und Sichtbarkeiten

    constructor(name: string) {
        this.name = name;
    }

    sayHi() { return `Hello, ${this.#name}` } // Return-Typ wird abgeleitet
}

// main.ts
import Person from "./Person";

const p = new Person("Klaus"); // OK
const h = new Person(123);    // ERROR: Number is not a string

const hi = p.sayHi();
hi.toUpperCase(); // OK - hi ist ein String (abgeleitet)

hi.sayHello(); // ERROR: sayHello gibt's nicht auf String
```

TypeScript (<https://typescriptlang.org>)

- Sehr guter IDE Support (Visual Studio Code, IDEA)
- Refaktorings, Find Usages etc. zuverlässig möglich
- Funktioniert mit allen SPA-Frameworks (Vue, Angular, React)
- M.E. „alternativlos“ im großen Projekt

The screenshot shows a code editor window for a file named `Greeting.ts`. The code defines a class `Greeting` with a private property `phrase` and a constructor that takes a `phrase` string. It has a method `greet` that returns a template string with the phrase and a name. A variable `g` is created and initialized with a new `Greeting` instance with the value "Hello". The code then attempts to call `g.greet(123)`, which triggers a TypeScript error: "Argument of type '123' is not assignable to parameter of type 'string'. ts(2345)". A tooltip for this error provides "Quick Fix..." and "Peek Problem" options. The code also contains a comment line and a final line where `g.phrase` is set to "goodbye".

```
1  class Greeting {
2      private phrase: string;
3
4      constructor(phrase: string) {
5          this.phrase = phrase;
6      }
7
8      greet(name: string) {
9          return `${this.phrase}, ${name}`;
10     }
11 }
12
13 const g = new Greeting("Hello");
14
15 // Argument of type '123' is not assignable to parameter of
16 // type 'string'. ts(2345)
17 // Quick Fix... Peek Problem
18 g.greet(123);
19
20 g.phrase = "goodbye";
21
22
23 const result:number = g.greet("Klaus");
```

Below the code editor, the VS Code interface shows the `PROBLEMS` tab selected, displaying three errors:

- Argument of type '123' is not assignable to parameter of type 'string'. ts(2345) [18, 9]
- Property 'phrase' is private and only accessible within class 'Greeting'. ts(2341) [20, 3]
- Type 'string' is not assignable to type 'number'. ts(2322) [23, 7]

At the bottom, the status bar shows: Go Live, Ln 16, Col 1, Spaces: 2, UTF-8, LF, TypeScript 3.4.3.

PACKAGE MANAGER / EXTERNE ABHÄNGIGKEITEN

npm: node package manager (<https://npmjs.com>)

- Standard Package Manager für Node-Entwicklung
- Beschreibung externer Abhängigkeiten
- Eigene Packages können publiziert werden
 - In zentrale Registry (analog maven-central)
 - Private Registry (z.B. Nexus) möglich
- Über npm werden üblicherweise auch notwendige **Tools** deklariert und installiert
 - Zum Beispiel für Compiler und Build-Tools
 - Jeder im Team verwendet einheitliche Version

yarn: npm Alternative (<https://yarnpkg.com/>)

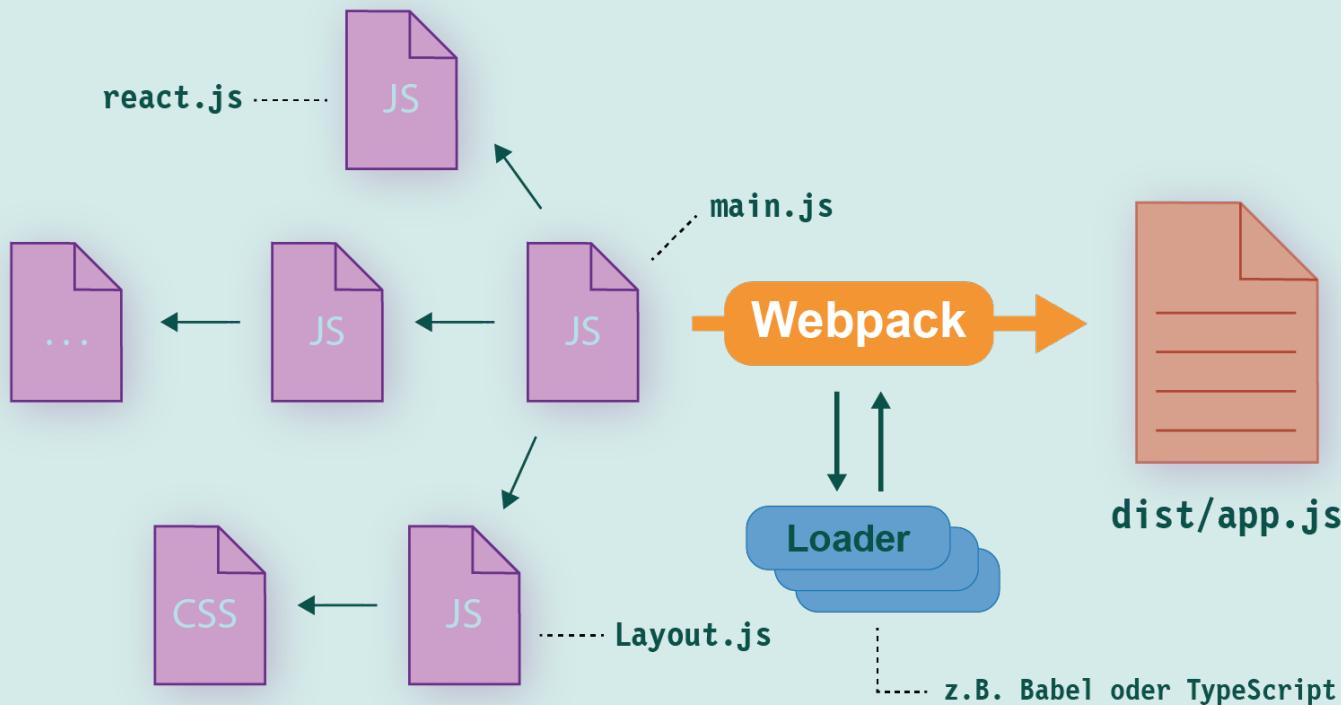
- Populäre Alternative zu npm
- Verwendet selbe Registries und gleiche Abhängigkeitsbeschreibung
- Etwas andere Kommandozeilen-Syntax
- Besonderes Feature
 - Plug‘n‘Play: Abhängigkeiten werden nur einmal pro Computer installiert
 - Workspaces: Arbeiten mit Monorepos

Webpack: Zentrales Build-Werkzeug

- <https://webpack.github.io/>
- Erstellt lauffähiges JavaScript-Modul ("Bundle"), quasi ein "Fat Jar" für den Browser
- Unterstützung für **alle** Modul-Systeme (CommonJS, Native, AMD)
- Wird von allen Frameworks verwendet, fester Bestandteil im React Tool-Stack

MODULE VERWENDEN

Webpack: Zentrales Build-Werkzeug



NILS HARTMANN

<https://nilshartmann.net>

vielen Dank!

Fragen & Kontakt: nils@nilshartmann.net

NILS@NILSHARTMANN.NET