

# **RICH CLIENT REACT**

# LERNZIELE

- Wie schreiben wir eine SPA in Javascript?
- Wie unterstützt uns React beim Schreiben einer SPA?
- Wie hilft eine Component Architecture beim Schreiben eines Frontends?
- Wie bringe ich Ordnung in eine Component Architectur?
- Was sind Micro Frontends und was bringt das?

# SPA IN PLAIN JAVASCRIPT

## SPA IN PLAIN JAVASCRIPT?

- spricht etwas dagegen?
- wäre quasi nur das exzessive einsetzen von Ajax

# PRAXIS: SPA IN PLAIN JAVASCRIPT

- Todo App als SPA in plain JavaScript
- HTML Seite mit plain Javascript anlegen
- den Web Service habt ihr letzte Woche bereits geschrieben
- falls ihr nicht fertig geworden seid
  - [https://gitlab.com/dhbw\\_webengineering\\_2/rich\\_client\\_server](https://gitlab.com/dhbw_webengineering_2/rich_client_server)

# PRAXIS: ANFORDERUNGEN

- Listenansicht der Todos
  - alle Titel der Todos werden in einer Liste angezeigt
  - es gibt einen Button hinter jedem Todo, um eine Detailansicht zu öffnen
- Detailseite für ein Todo
  - hier soll eine genauere Beschreibung des Todos angezeigt werden
- kein explizites Routing (Anpassung der URL)

**PRAXIS: WAS BRAUCHEN WIR?**

## **PRAXIS: WAS BRAUCHEN WIR?**

- index.html mit Placeholder für den Inhalt



## PRAXIS: WAS BRAUCHEN WIR?

- index.html mit Placeholder für den Inhalt
- HTTP call ans Backend, zum laden der Daten

## PRAXIS: WAS BRAUCHEN WIR?

- index.html mit Placeholder für den Inhalt
- HTTP call ans Backend, zum laden der Daten
- Javascript zum Bauen der Listenansicht

## PRAXIS: WAS BRAUCHEN WIR?

- index.html mit Placeholder für den Inhalt
- HTTP call ans Backend, zum laden der Daten
- Javascript zum Bauen der Listenansicht
- Javascript zum Bauen der Detailansicht

## PRAXIS: WAS BRAUCHEN WIR?

- index.html mit Placeholder für den Inhalt
- HTTP call ans Backend, zum laden der Daten
- Javascript zum Bauen der Listenansicht
- Javascript zum Bauen der Detailansicht
- Initialisierung der Seite

# PRAXIS: UNTERSTÜTZUNG

- ihr könnt natürlich direkt mit einer HTML Seite starten
- wer sich beim Styling ein wenig Zeit sparen möchte:
  - [https://gitlab.com/dhbw\\_webengineering\\_2/spa\\_plain\\_javascript](https://gitlab.com/dhbw_webengineering_2/spa_plain_javascript) (branch: main)
  - enthält funktionen, um einige Components zu bauen

# PRAXIS: LÖSUNG

- Lösung gibt es hier:  
[https://gitlab.com/dhbw\\_webengineering\\_2/spa\\_plain\\_javascript](https://gitlab.com/dhbw_webengineering_2/spa_plain_javascript)  
(branch: solution)
- ihr dürft euch natürlich inspirieren lassen

# **SPA IN PLAIN JAVASCRIPT?**

Was ist aufgefallen bei einer Implementierung in plain JavaScript?

# SPA IN PLAIN JAVASCRIPT?

Was ist aufgefallen bei einer Implementierung in plain JavaScript?

- es ist sehr aufwendig
- viel boilerplate Code
  - XMLHttpRequests sind immer gleich
  - DOM Manipulation
  - Routing



# SPA IN REACT

# WIE HILFT UNS REACT?

- unterstützt
  - DOM Manipulation
  - Routing
- bietet eine Menge Libraries zur Unterstützung
  - axios: für XMLHttpRequests
- bietet Change Detection

# REACT FUNCTIONS

- enthalten Information und Logik zum Rendern der UI
- eine Mischung aus Javascript und HTML (JSX)

```
1 export default function ReactFunction() {  
2     const name = 'World';  
3  
4     return <div>Hello {name}!</div>  
5 }
```

# REACT FUNCTIONS

- Expressions im HTML sind möglich

```
1 export default function ReactFunction() {  
2     const names = ['World', 'Daniel', 'Iven', 'Kai'];  
3  
4     return <div>Hello {names.join(', ')}!</div>  
5 }
```

# REACT FUNCTIONS

- HTML in einer Expression ebenfalls

```
1 export default function ReactFunction() {  
2     const names = ['World', 'Daniel', 'Iven', 'Kai'];  
3  
4     return <div>Hello {names.map(name => <b>{name}</b>)}!</div>  
5 }
```

# REACT FUNCTIONS

```
1 export default function ReactFunction() {  
2     let name;  
3  
4     return <div>{ name ? `Hello ${name}!` : 'Loading' }</div>  
5 }
```

# PRAXIS: REACT FUNCTIONS (LISTVIEW)

- ListView mit React bauen
- ganz simpel, kein Styling, kein Backend!
- [https://gitlab.com/dhbw\\_webengineering\\_2/rich\\_client\\_react](https://gitlab.com/dhbw_webengineering_2/rich_client_react)  
(branch: step\_0-list\_view)

# REACT ROUTING

- bietet uns einfache Navigation
- automatische Anpassung der URL



# REACT ROUTING

```
1 function App() {
2   return (
3     <div className="App">
4       <BrowserRouter>
5         <Routes>
6           <Route path="/" element={<Screen1 />} />
7           <Route path="/screen1" element={<Screen1 />} />
8           <Route path="/screen2/:someParam"
9             element={<Screen2 />} />
10        </Routes>
11      </BrowserRouter>
12    </div>
13  );
14 }
```

# REACT ROUTING

- mit `useNavigate()` können wir Navigationen auslösen.

```
1 export default function SomeScreen() {
2   const navigate = useNavigate();
3   const someParam = 'someParam';
4
5   return <div>
6     <Button label='Screen1' onClick={navigate('/') }
7     <Button
8       label='Screen2 '
9       onClick={navigate(`/screen2/${someParam}`) }
10   </div>
11 }
```

# REACT ROUTING

- useParams() erlaubt es uns auf Pfadparameter zuzugreifen

```
1 export default function Screen2() {  
2     const { someParam } = useParams();  
3  
4     return <div>{someParam}</div>;  
5 }
```

## PRAXIS: REACT ROUTING

- zu unserem bestehenden ListView bauen wir einen DetailView
- Navigation zum DetailView und zurück soll möglich sein
- wer nicht mitgekommen ist:
  - [https://gitlab.com/dhbw\\_webengineering\\_2/rich\\_client\\_react](https://gitlab.com/dhbw_webengineering_2/rich_client_react) (branch: step\_1-routing)

# LADEN DYNAMISCHER DATEN

- mit der Library axios
- GET Request wie folgt:

```
1 async getData() {  
2     return axios.get(  
3         'https://somedomain.de/get/data',  
4     ).then((response) => response.data);  
5 }
```

# LADEN DYNAMISCHER DATEN

- POST Request:

```
1 async saveData(data) {  
2     return axios.post(  
3         'https://somedomain.de/post/data',  
4         data,  
5         {  
6             headers: { 'Content-Type': 'application/json' },  
7         },  
8     ).then((response) => response.data);  
9 }
```

# LADEN DYNAMISCHER DATEN

- Auslagern der Requests in eine eigene Klasse

```
1 export default class DataHttpClient {  
2     async getData() {  
3     }  
4  
5     async saveData(data) {  
6     }  
7 }
```

# LADEN DYNAMISCHER DATEN

- Bereitstellen des DataHttpClient mittels Dependency Injection
- in React nutzt man Context Injection

```
1 export const DataHttpClientContext =  
2     createContext(DataHttpClient);  
3  
4 function App() {  
5     return (  
6         <div classname="App">  
7             <DataHttpClientContext.Provider  
8                 value={new DataHttpClient()}>  
9                 <Screen1 />  
10            </DataHttpClientContext.Provider>  
11        </div>  
12    );  
13 }
```



# LADEN DYNAMISCHER DATEN

- Abrufen des Objekts mit useContext()
- nur möglich, wenn sich die React function im korrekten Kontext befindet

```
1 export default function Screen1() {  
2     const dataHttpClient = useContext(DataHttpClientContext)  
3  
4     return <div></div>;  
5 }
```

## PRAXIS: LADEN DYNAMISCHER DATEN

- schreibt euch einen TodoHttpClient mit dem ihr Todos abrufen könnt
- startet dazu den Web Service vom letzten Mal
- macht den Client per DI verfügbar
- wer nicht mitgekommen ist:
  - [https://gitlab.com/dhbw\\_webengineering\\_2/rich\\_client\\_react](https://gitlab.com/dhbw_webengineering_2/rich_client_react) (branch: step\_2-load\_data)

# REACT HOOKS

- ein Thema für sich
- speichern von State: `useState()`
- Lifecycle: `useEffect()`

# REACT HOOKS

- `useState()`
  - zum Speichern/Ändern von Daten in einer React function

```
1 export default function Screen2() {  
2   const [count, setCount] = useState(0)  
3  
4   return <button onClick={() => setCount(count + 1)}>  
5     {count}  
6   </button>;  
7 }
```

# REACT HOOKS

- `useEffect()`
  - Seiteneffekte für React functions
  - Callback der zu bestimmten Zeitpunkten aufgerufen wird

```
1 export default function Screen2() {  
2     ...  
3  
4     useEffect(() => {  
5         dataHttpClient.getData()  
6             .then((data) => console.log(data));  
7     });  
8  
9     ...  
10 }
```

# REACT HOOKS

- `useEffect()`
  - funktioniert gut in Kombination mit `useState()`

```
1 export default function Screen2() {
2   ...
3   const [data, setData] = useState('')
4
5   useEffect(() => {
6     dataHttpClient.getData(data)
7       .then((data) => setData(data));
8   });
9
10  return <input
11    value={data}
12    onChange={(e) => setData(e.target.value)} />;
13 }
```

# REACT HOOKS

- `useEffect()`
  - muss nicht auf State Änderungen reagieren
  - reagiert auf alle Parameter im Array

```
1 export default function Screen2() {
2   ...
3   const [data, setData] = useState('')
4
5   useEffect(() => {
6     dataHttpClient.getData(data)
7       .then((data) => setData(data));
8   }, []);
9
10  return <input
11    value={data}
12    onChange={(e) => setData(e.target.value)} />;
13 }
```

# REACT HOOKS

- `useEffect()`
  - kann zum Aufräumen verwendet

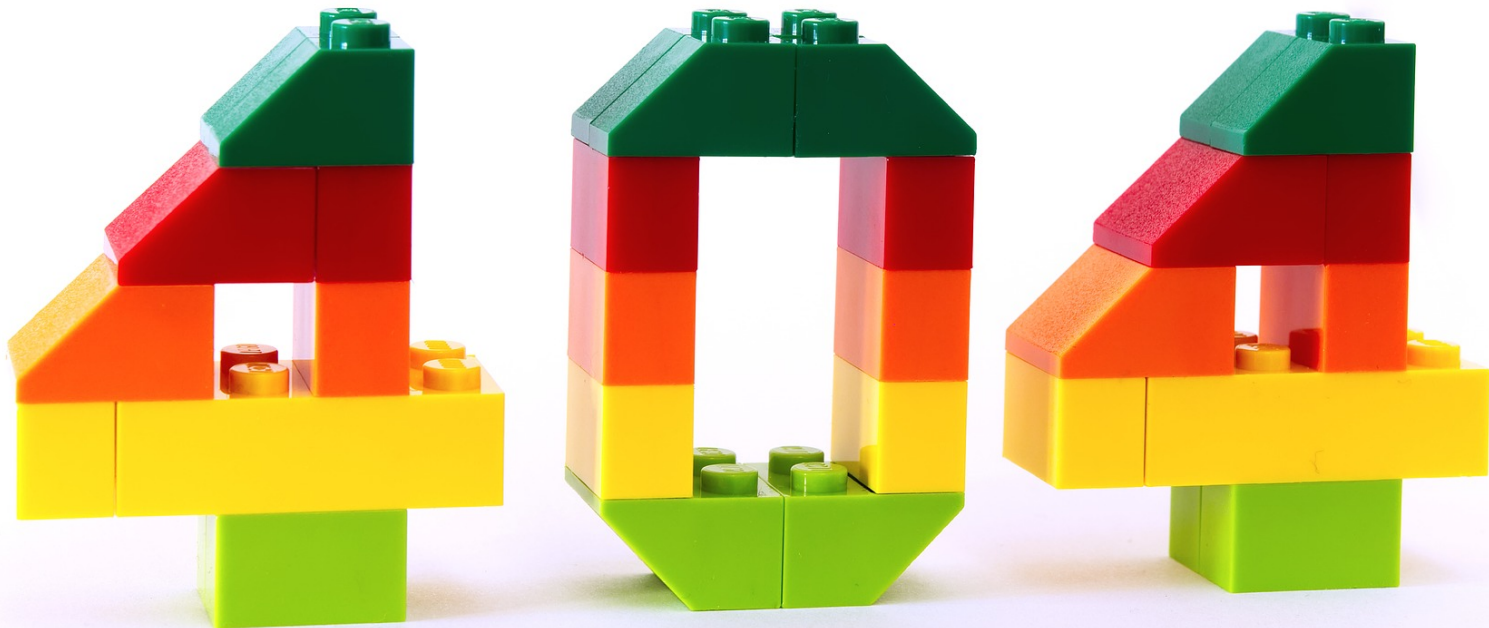
```
1 export default function Screen2() {
2   ...
3   const [data, setData] = useState('')
4
5   useEffect(() => {
6     dataHttpClient.getData(data)
7       .then((data) => setData(data));
8     return () => console.log('teardown');
9   }, []);
10
11   return <input
12     value={data}
13     onChange={(e) => setData(e.target.value)} />;
14 }
```



## PRAXIS: REACT HOOKS

- ladet die Todos über den Client mit dem `useEffect()` Hook
- für den ListView und den DetailView
- macht die Checkbox im Detailview funktionsfähig
- wer nicht mitgekommen ist:
  - [https://gitlab.com/dhbw\\_webengineering\\_2/rich\\_client\\_react](https://gitlab.com/dhbw_webengineering_2/rich_client_react) (branch: `step_3-react_hooks`)

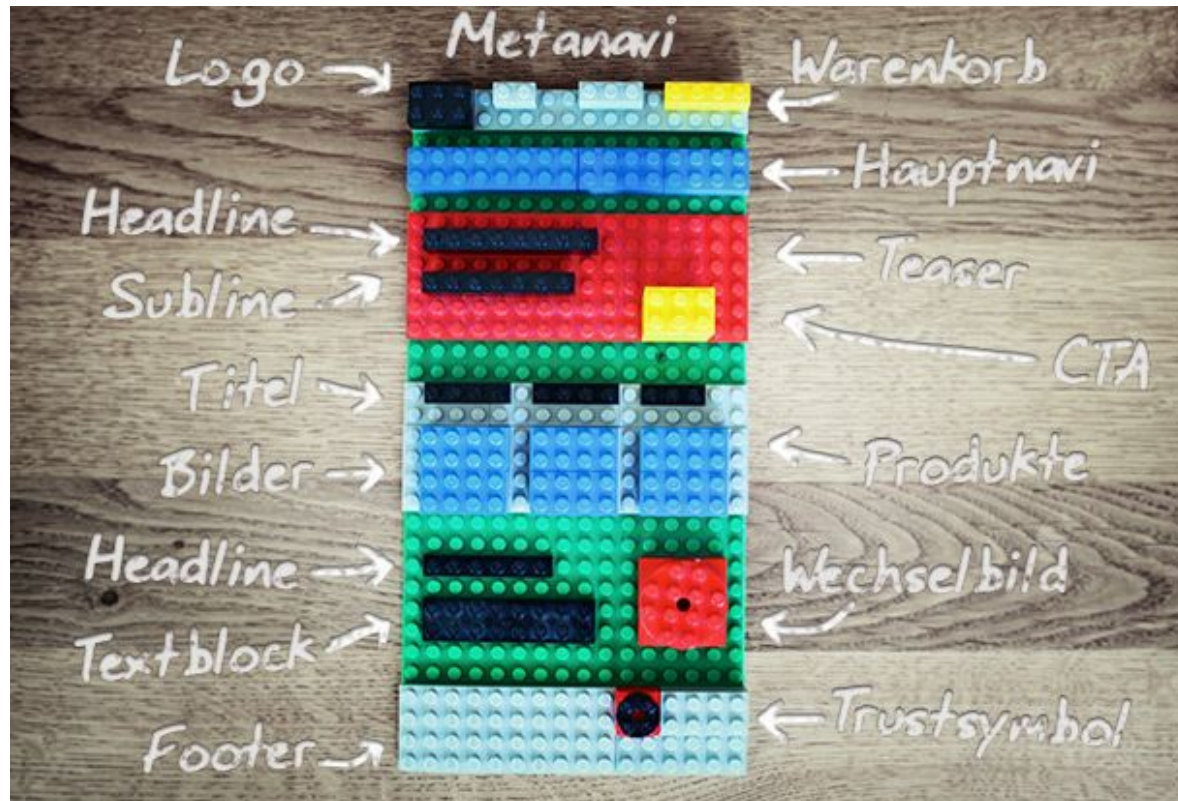
# COMPONENT ARCHITECTURE



# COMPONENT ARCHITECTURE

- divide et impera
  - teilen der Webseite in einzelnen Components
  - Verteilung und Strukturierung der Komplexität
- Components
  - enthalten zusammengehörige Funktionalität
  - haben feste Schnittstellen
    - möglichst lose Kopplung und hohe Kohäsion
    - analog wie Legosteine
  - abstrahieren Struktur und Styling

# COMPONENT ARCHITECTURE

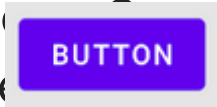


# COMPONENT ARCHITECTURE

- Was kann alles eine Component sein?
  - Buttons, Text Fields, Labels, etc.
  - Search Bar, Form Groups, Cards, etc.
  - Header, Footer, Overlays, etc.
  - Pages

# COMPONENT ARCHITECTURE

- Was kann alles eine Component sein?
  - Buttons, Text Fields, Labels, etc.
  - Search Bar, Form Groups, Cards, etc.
  - Header, Footer, Modals, etc.
  - Pages



# COMPONENT ARCHITECTURE

- Was k... ent sein?
  - Bu
  - Se
  - He
  - Pa



## Cafe Badilico

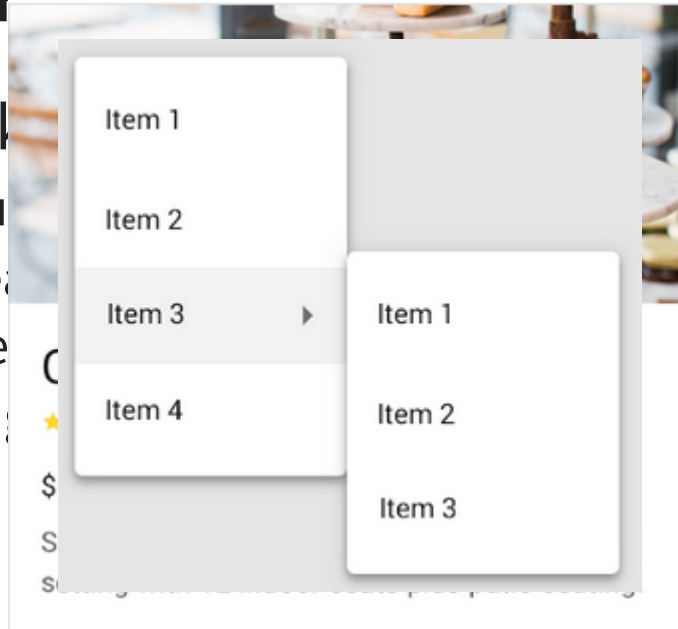
★★★★☆ 4.5 (413)

\$ • Italian, Cafe

Small plates, salads & sandwiches an intimate setting with 12 indoor seats plus patio seating.

# COMPONENT ARCHITECTURE

- Was hier zu sehen ist, ist ein Beispiel für eine Komponente, die in einem Menü ent sein?
  - Bu
  - Se
  - He
  - Pa





## COMPONENT ARCHITECTURE

[Listenansicht](#)

Take the trash out

Done ☐

Details

Feed the cat

Done ☐

Details

Eat grandpa

Done ☒

Details

Clean the bathroom

Done ☐

Details

+

# COMPONENTS

```
1 <button value="Submit" onclick="alert('Button clicked!')"/>
```

- Components haben feste Schnittstellen
- damit können sie modular eingesetzt werden
- es können Parameter rein und rausgegeben werden

# COMPONENTS IN REACT

- React functions sind Components
- eine React functions kann Parameter entgegennehmen
- über eine Callback kann ein Wert zurückgegeben werden

```
1 export default function Button({ primary, label,  
2                                onClick, className }) => {  
3     const mode = primary ?  
4         'button--primary' : 'button--secondary';  
5     return (  
6         <button  
7             type="button"  
8             className={['button', mode, className].join(' ')  
9             onClick={() => onClick()}  
10        />  
11        {label}  
12    </button>  
13    );  
14 };
```

# COMPONENTS IN REACT

- über propTypes können wir eine Schnittstelle definieren
- über defaultProps können wir Defaultwerte hinterlegen

```
1 export default function Button({ ... }) {
2     ...
3 };
4 Button.propTypes = {
5     primary: PropTypes.bool,
6     label: PropTypes.string.isRequired,
7     onClick: PropTypes.func,
8     className: PropTypes.string,
9 };
10 Button.defaultProps = {
11     primary: false,
12     onClick: undefined,
13     className: '',
14 };
```

# COMPONENT ARCHITECTURE

- Vorteile:
  - Konsistenz im Styling
  - Wiederverwendbarkeit
  - schnellere Entwicklung
  - einfachere Instandhaltung
- Nachteile:
  - tiefe Verschachtelungen möglich

# PRAXIS: COMPONENT ARCHITECTURE

- aufteilen der ListView Seite in kleinere Components
- überlegt euch selbst, wie ihr die Seite aufteilen könnt
- Basiskomponenten stehen bereit um Zeit zu sparen
  - [https://gitlab.com/dhbw\\_webengineering\\_2/rich\\_client\\_react](https://gitlab.com/dhbw_webengineering_2/rich_client_react) (branch: step\_3-component\_architecture)

# ATOMIC DESIGN

# ATOMIC DESIGN

große Frontends mit vielen Components werden unübersichtlich

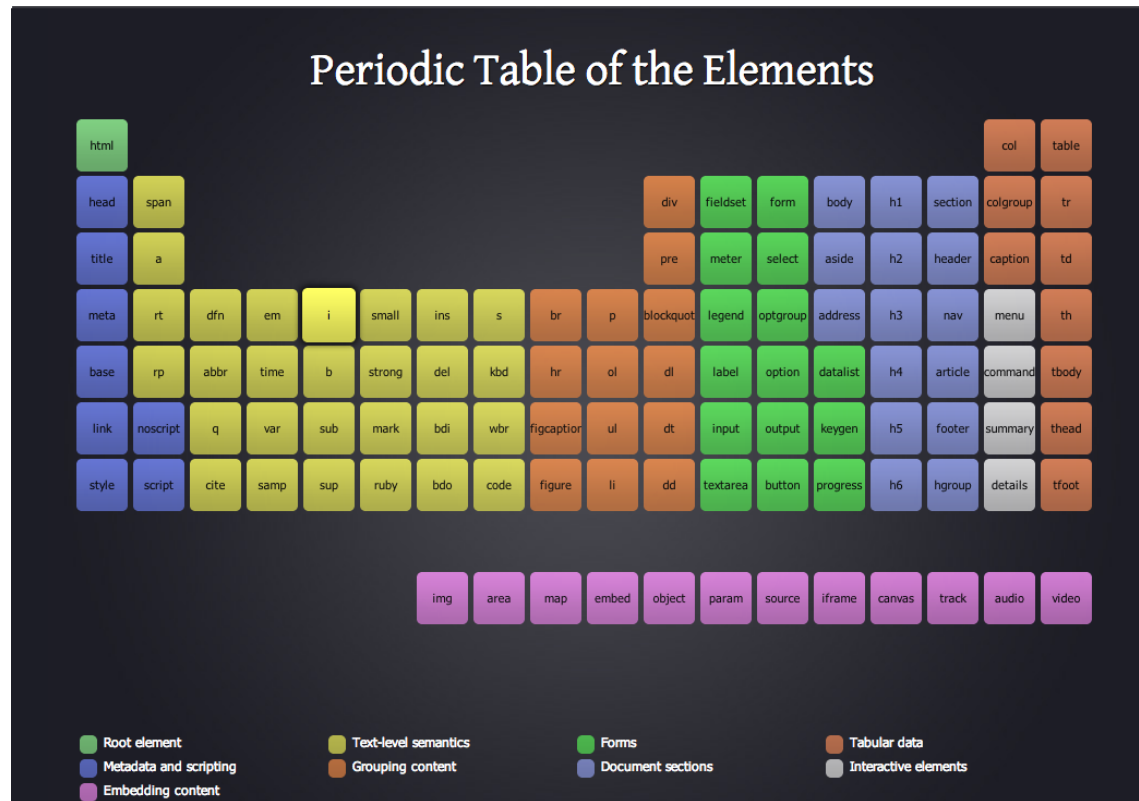




# ATOMIC DESIGN

# Strukturierung und Kategorisierung von Components

## Ziel ist ein ordentlicher Baukasten an Components



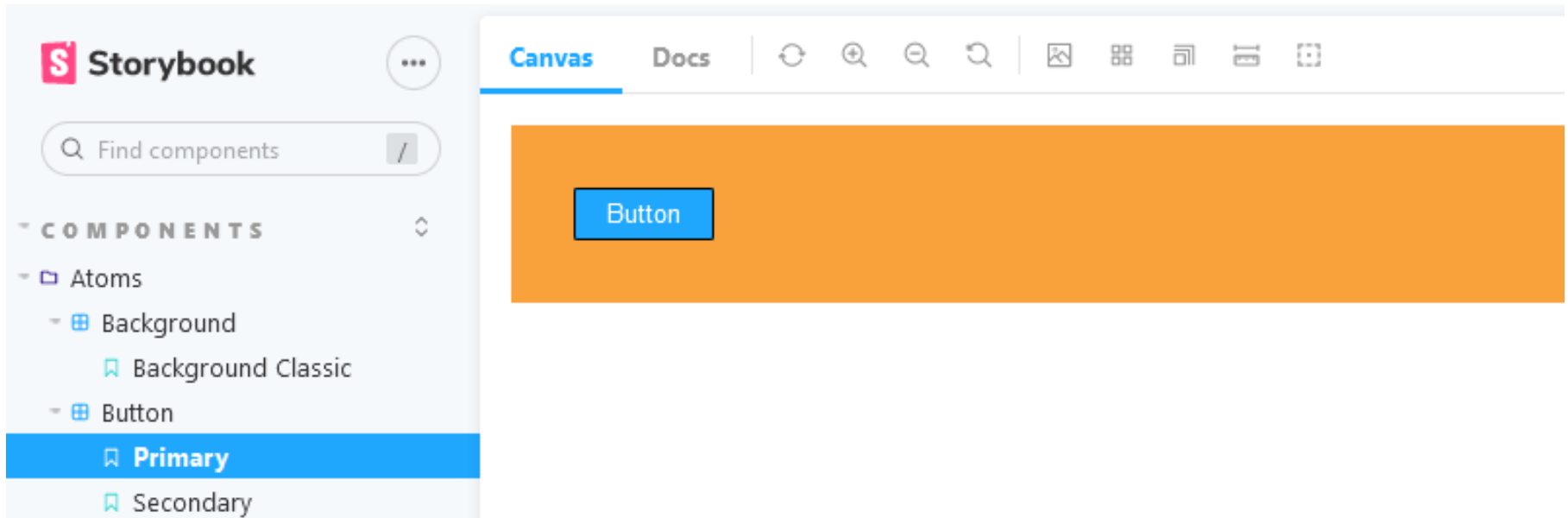
<https://bradfrost.com/blog/post/atomic-web-design/>

# ATOMIC DESIGN

- Atomic Design ordnet Components nach:
  - Atoms - Buttons, Text Fields, etc.
  - Molecules - Search Bar, Form Groups, etc.
  - Organisms - Header, Footer, Overlays, etc.
  - Templates - Schablone
  - Pages - konkrete Seite

# STORYBOOK

- macht Atomic Design noch nützlicher
- Visualisierung einzelner Components in verschiedener Ausprägung



# PRAXIS: ATOMIC DESIGN + STORYBOOK

- sortieren des Projekts nach Atomic Design
- Storybook Eintrag erstellen für ein paar Components
- wir schauen uns gleich mal zusammen die Syntax an
- wer nicht mitgekommen ist:
  - [https://gitlab.com/dhbw\\_webengineering\\_2/rich\\_client\\_react](https://gitlab.com/dhbw_webengineering_2/rich_client_react) (branch: step\_3-atomic\_design)

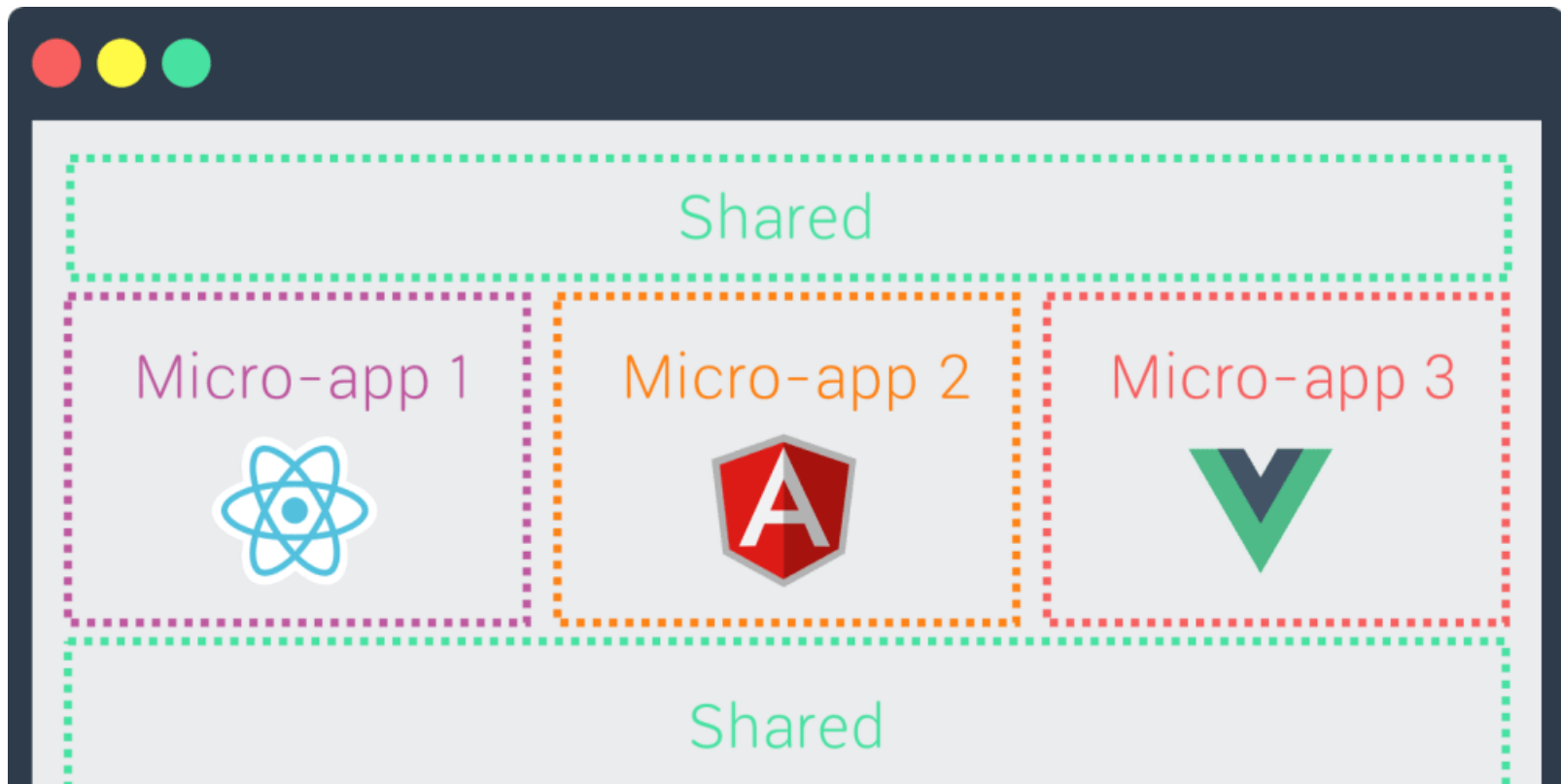
# MICRO FRONTENDS

# COMPONENT ARCHITECTURE

- bringt uns Ordnung und Struktur
- was passiert wenn das Frontend wächst?
- mehrere Teams arbeiten an einem Frontend?
- unterschiedliche Teams
  - mögen unterschiedliche Technologien
  - haben unterschiedliche Arbeitsweisen
  - möchten unabhängig releasen
  - haben unterschiedlichen Codestyle

# MICRO FRONTENDS

- aufteilen des Monolith in mehrere Frontends
- Frontends können zu einem Frontend zusammengesteckt werden



# MICRO FRONTENDS

- reden meist auch mit eigenen Backends
- Micro Services

