

# **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 bei einer Implementierung aufgefallen in plain JavaScript?

# SPA IN PLAIN JAVASCRIPT?

Was ist bei einer Implementierung aufgefallen 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

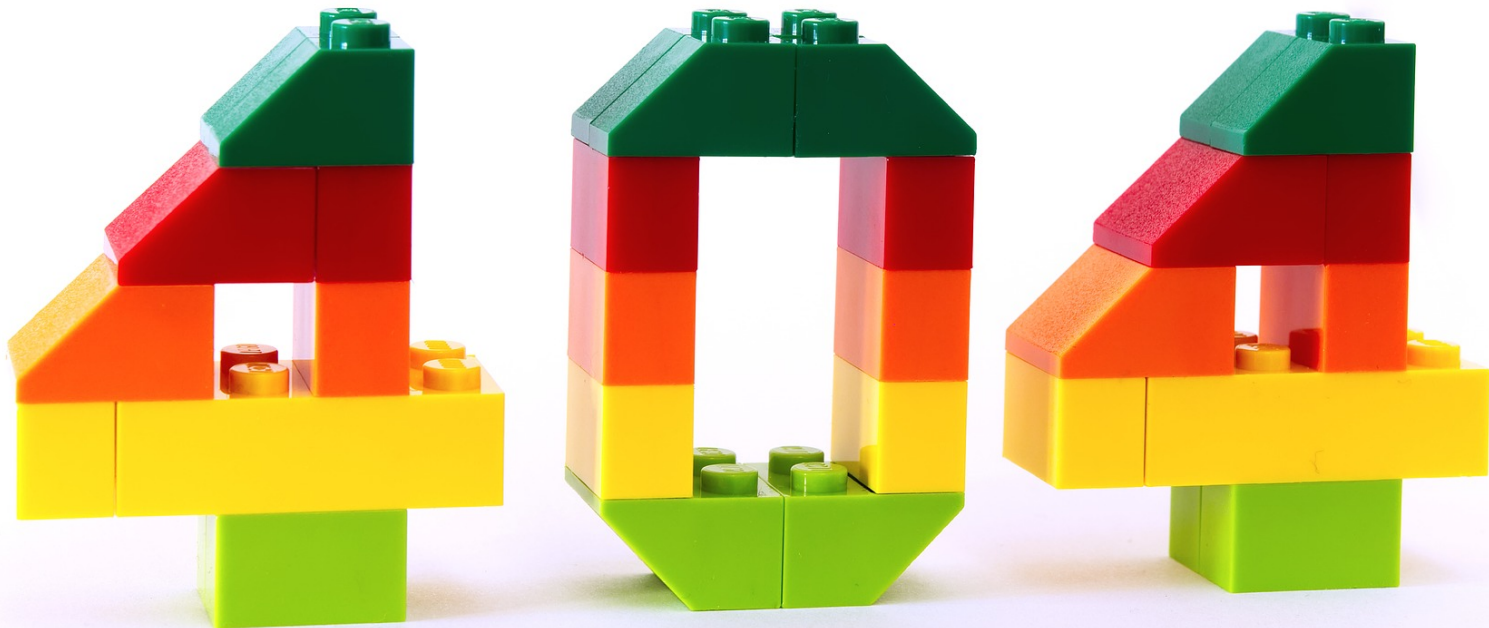
```
1 export default function Screen2() {
2   ...
3   const [data, setData] = useState('')
4
5   useEffect(() => {
6     dataHttpClient.getData(data)
7       .then((data) => setData(data));
8     return () => console.log('teardown');
9   }, []);
10
11   return <input
12     value={data}
13     onChange={(e)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

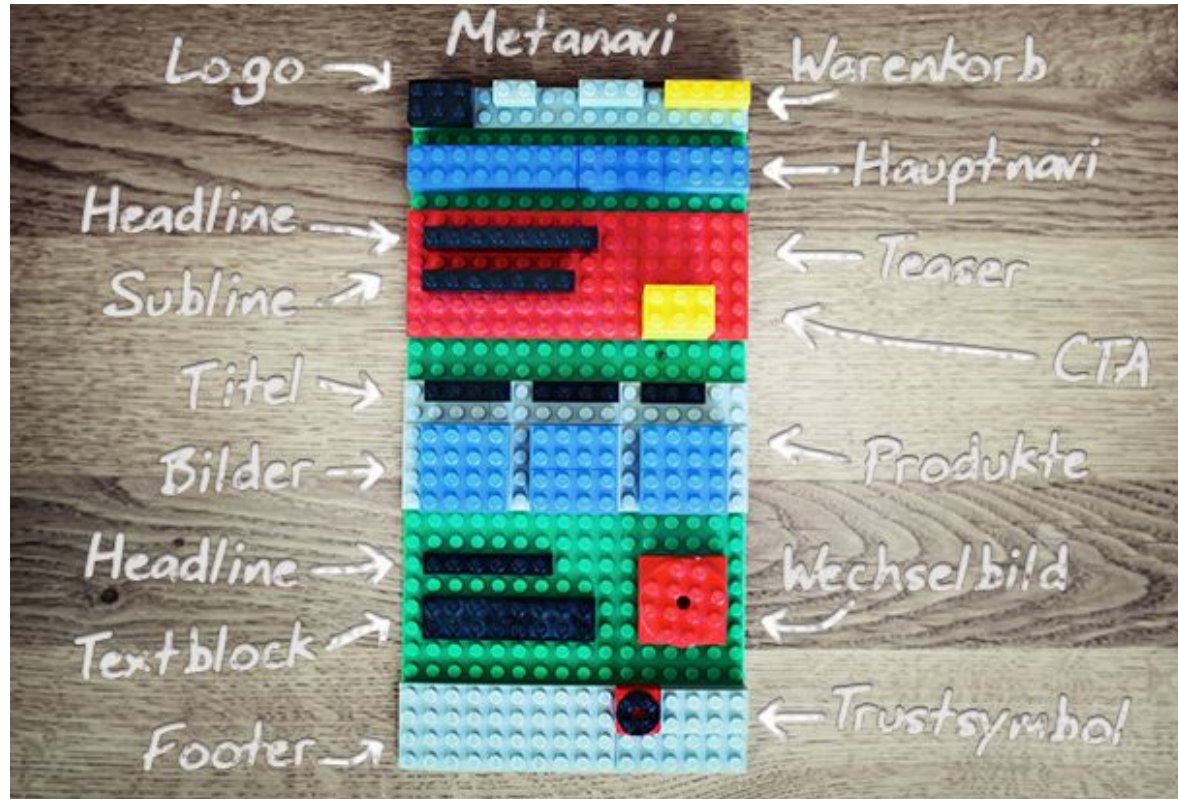
# COMPONENT ARCHITECTURE



# 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

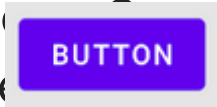


# COMPONENT ARCHITECTURE

- 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

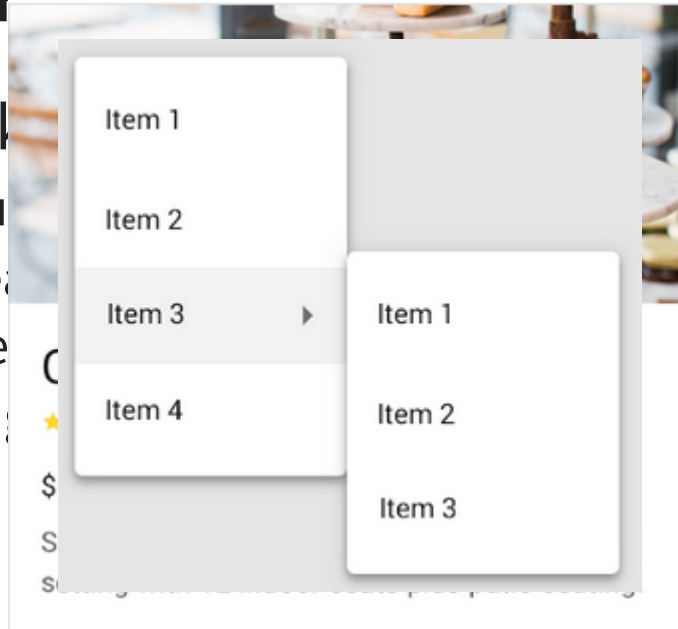
★★★★☆ 4.5 (413)

\$ • Italian, Cafe

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

# COMPONENT ARCHITECTURE

- Was ist ein Component sein?
  - Button
  - Select
  - Header
  - Page





## 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 function kann Parameter entgegennehmen
- Über einen 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

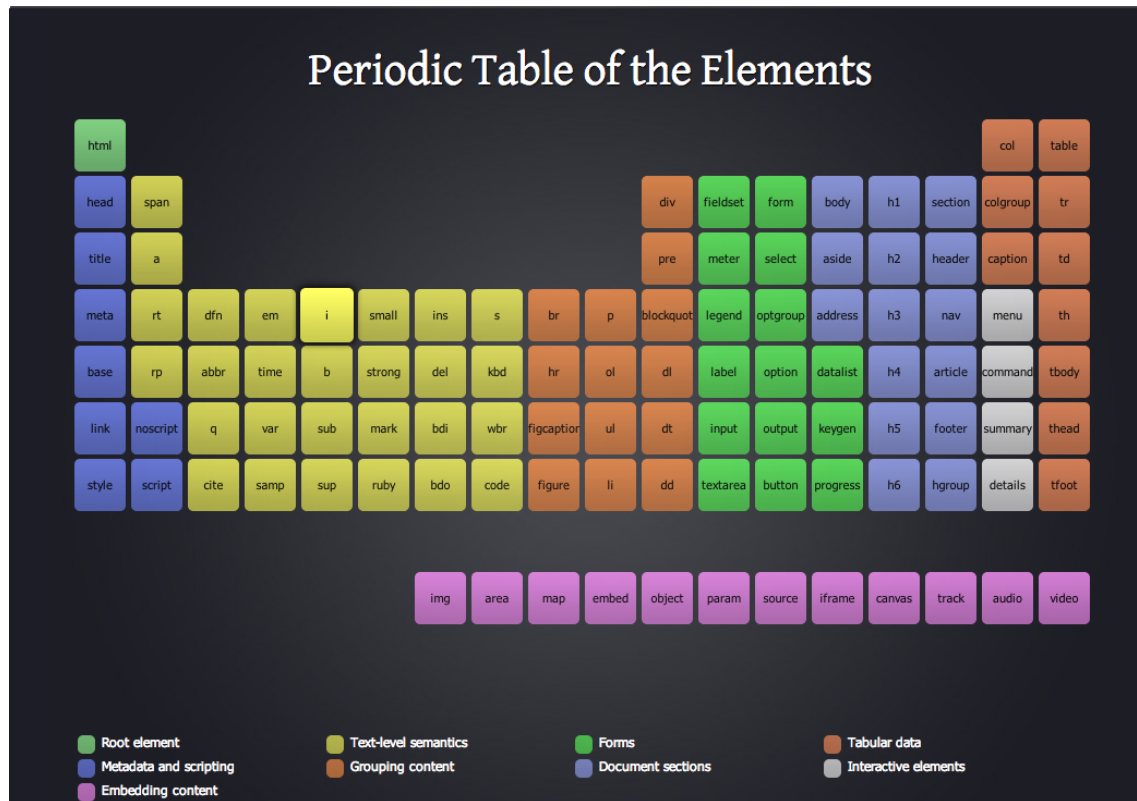




# ATOMIC DESIGN

Strukturierung und Kategorisierung von Components

Ziel ist ein ordentlicher Baukasten an Components



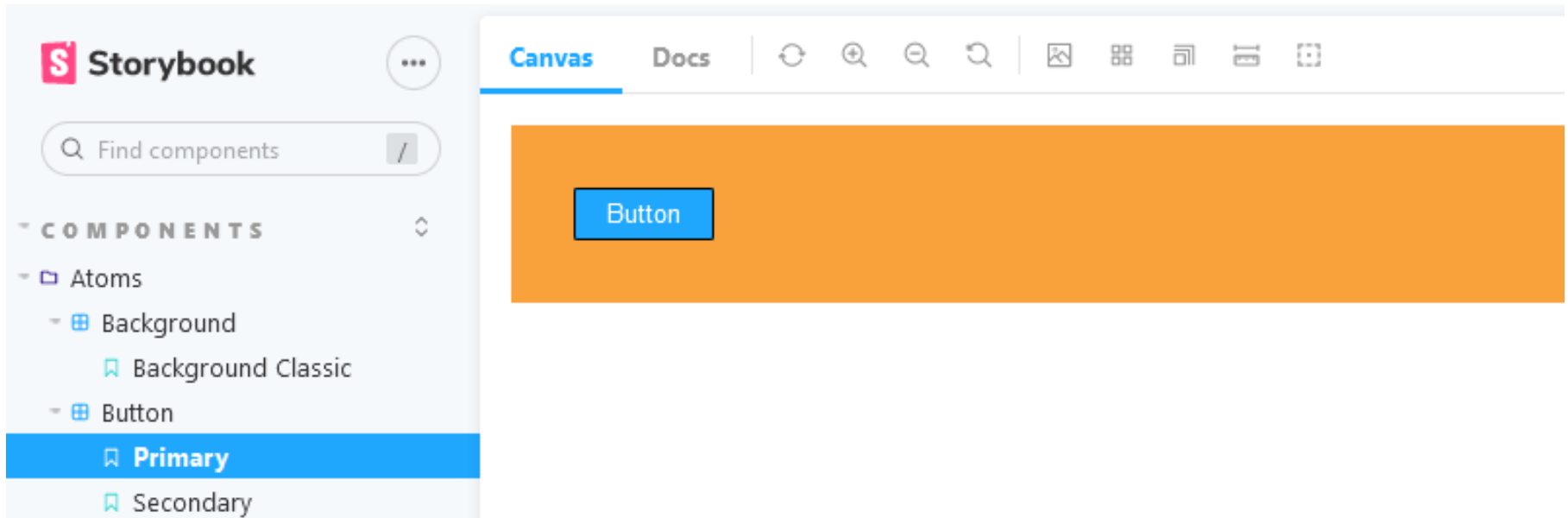
<https://bradfrost.com/blog/post/atomic-web-design/>

# 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



# STORYBOOK SYNTAX

```
1 export default {  
2   title: 'Components/Background',  
3   component: Background,  
4 };  
5  
6 const Template = () => <Background />;  
7  
8 export const BackgroundClassic = Template.bind({});
```

# STORYBOOK SYNTAX

```
1 export default {  
2   title: 'Components/Background',  
3   component: Background,  
4 };  
5  
6 const Template = () => <Background />;  
7  
8 export const BackgroundClassic = Template.bind({});
```

# STORYBOOK SYNTAX

```
1 export default {  
2   title: 'Components/Background',  
3   component: Background,  
4 };  
5  
6 const Template = () => <Background />;  
7  
8 export const BackgroundClassic = Template.bind({});
```

# STORYBOOK SYNTAX

```
1 export default {  
2   title: 'Components/Background',  
3   component: Background,  
4 };  
5  
6 const Template = () => <Background />;  
7  
8 export const BackgroundClassic = Template.bind({});
```

# STORYBOOK MIT PARAMETERN

```
1  const Template = (args) => <Background>
2                                <Button {...args} />
3                                </Background>;
4
5  export const Primary = Template.bind({});
6  Primary.args = {
7    primary: true,
8    label: 'Button',
9  };
10
11 export const Secondary = Template.bind({});
12 Secondary.args = {
13   label: 'Button',
14 };;
```



# STORYBOOK MIT PARAMETERN

```
1  const Template = (args) => <Background>
2                                <Button {...args} />
3                                </Background>;
4
5  export const Primary = Template.bind({});
6  Primary.args = {
7    primary: true,
8    label: 'Button',
9  };
10
11 export const Secondary = Template.bind({});
12 Secondary.args = {
13   label: 'Button',
14 };;
```

# STORYBOOK MIT PARAMETERN

```
1  const Template = (args) => <Background>
2                                <Button {...args} />
3                                </Background>;
4
5  export const Primary = Template.bind({});
6  Primary.args = {
7    primary: true,
8    label: 'Button',
9  };
10
11 export const Secondary = Template.bind({});
12 Secondary.args = {
13   label: 'Button',
14 };;
```

# STORYBOOK MIT PARAMETERN

```
1  const Template = (args) => <Background>
2                                <Button {...args} />
3                                </Background>;
4
5  export const Primary = Template.bind({});
6  Primary.args = {
7    primary: true,
8    label: 'Button',
9  };
10
11 export const Secondary = Template.bind({});
12 Secondary.args = {
13   label: 'Button',
14 };;
```

# STORYBOOK MIT CONTEXTINJECTION

```
1  const someMockClient = {
2    getData(_) {
3      return Promise.resolve(new Data('lala'));
4    },
5    saveData(data) {
6      return Promise.resolve(data);
7    }
8  }
9
10 const Template = (args) =>
11     <SomeContext.Provider value={someMockClient}>
12       <SomeComponent {...args} />
13     </SomeContext.Provider>;
```

# STORYBOOK MIT ROUTER

```
1 export const reactRouterDecorator = (Story) => {
2   return (
3     <MemoryRouter>
4       <Routes>
5         <Route path="/" element={<Story />} />
6       </Routes>
7     </MemoryRouter>
8   )
9 }
```

```
1 export default {
2   title: 'Components/SomeComponent',
3   component: SomeComponent,
4   decorators: [reactRouterDecorator],
5 };
```

# PRAXIS: ATOMIC DESIGN + STORYBOOK

- Sortieren des Projekts nach Atomic Design
- Storybookeintrag erstellen für wenigstens drei weitere Components
- 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

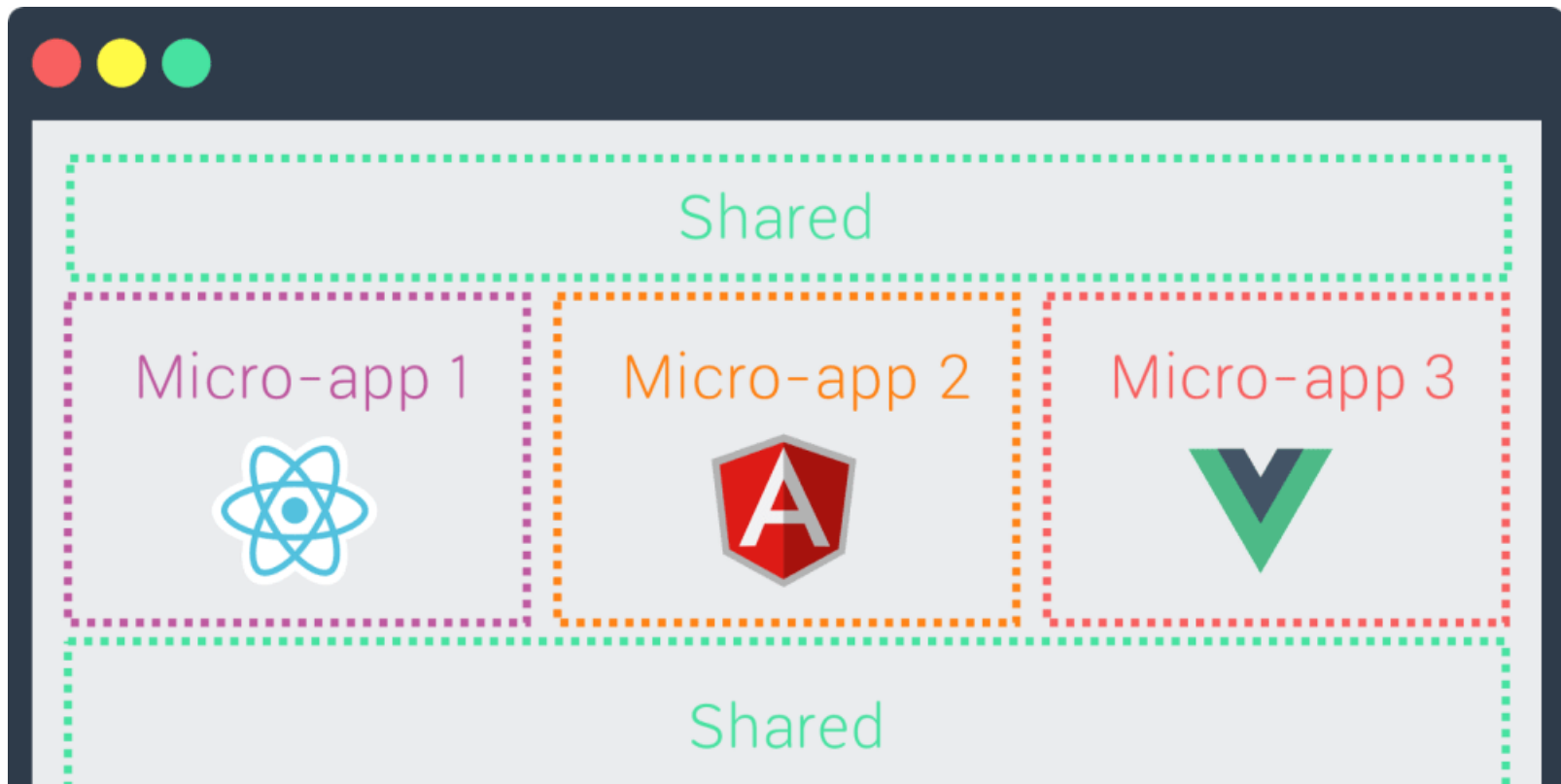
# COMPONENT ARCHITECTURE

- 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



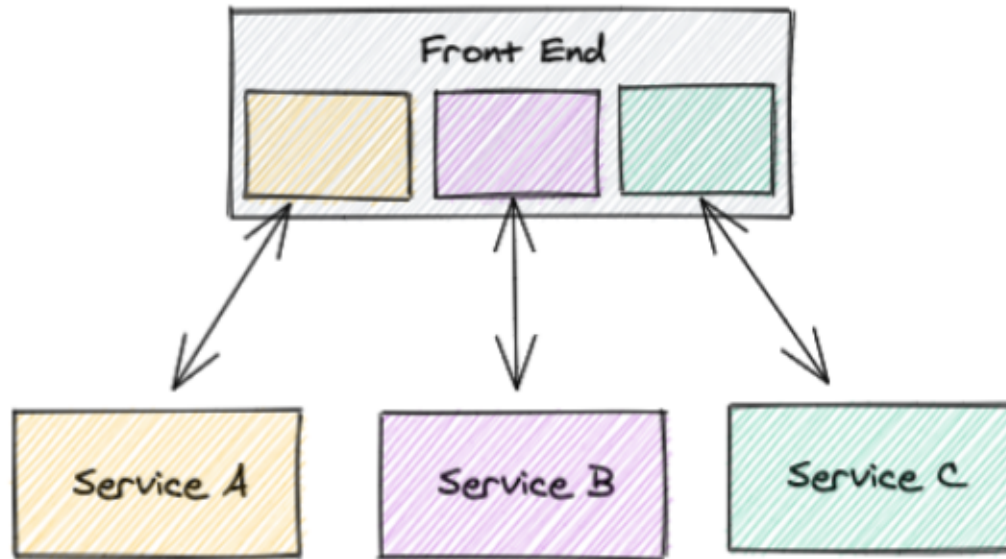
# MICRO FRONTENDS

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



# MICRO FRONTENDS

- Reden meist auch mit eigenen Backends
- Micro Services



# 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?