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

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 (branch: main)
  - enthält funktionen, um einige Components zu bauen

PRAXIS: LÖSUNG

- Lösung gibt es hier: 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>)}!</
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 (branch: step_0-list_view)

REACT ROUTING

- bietet uns einfache Navigation
- automatische Anpassung der URL

REACT ROUTING

```
function App() {
    return (
    <div className="App">
        <BrowserRouter>
            <Routes>
                <Route path='/' element={<Screen1 />} />
                <Route path='/screen1' element={<Screen1 />
                <Route path='/screen2/:someParam'
                    element={<Screen2 />} />
            </Routes>
        </BrowserRouter>
    </div>
    );
}
```

REACT ROUTING

- mit useNavigate() können wir Navigationen auslösen.

```
export default function SomeScreen() {
    const navigate = useNavigate();
    const someParam = 'someParam';

    return <div>
            <Button label='Screen1' onClick={navigate('/')}
            <Button
                label='Screen2'
                onClick={navigate(`/screen2/${someParam}`)}
    </div>
}
```

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 (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:

```
async saveData(data) {
    return axios.post(
        'https://somedomain.de/post/data',
        data,
        {
            headers: {'Content-Type': 'application/json'},
        },
    ).then((response) => response.data);
}
```

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 (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

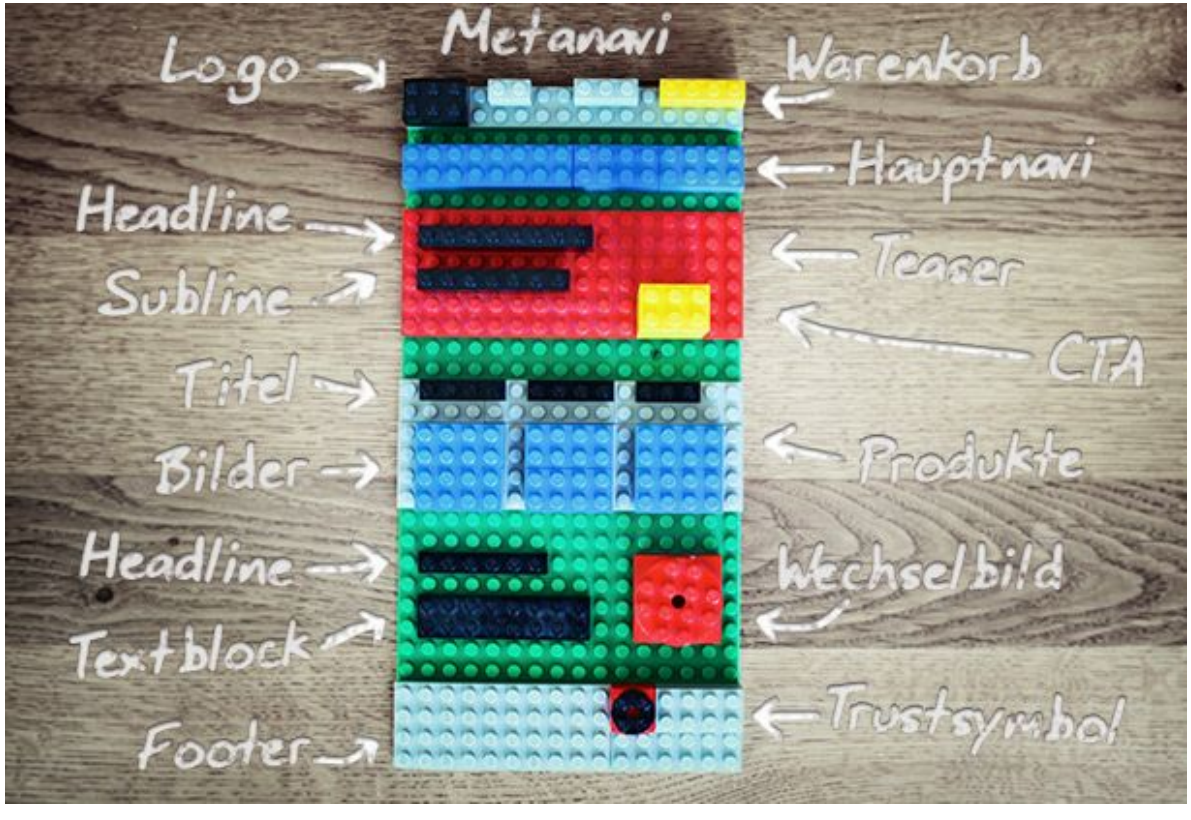
- ladet die Todos über den Client mit dem useEffect() Hook
- für den ListView und den DetailView
- macht die Checkbox im Detailview funktionsfähig
- wer nicht mitgekommen ist:
  - https://gitlab.com/dhbw_webengineering_2/rich_client_react (branch: step_3-react_hooks)

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

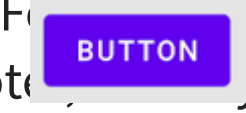


COMPONENT ARCHITECTURE

- 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, F ps, Cards, etc.
  - Header, Foote ys, etc.
  - Pages



COMPONENT ARCHITECTURE

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

Cafe Badilico

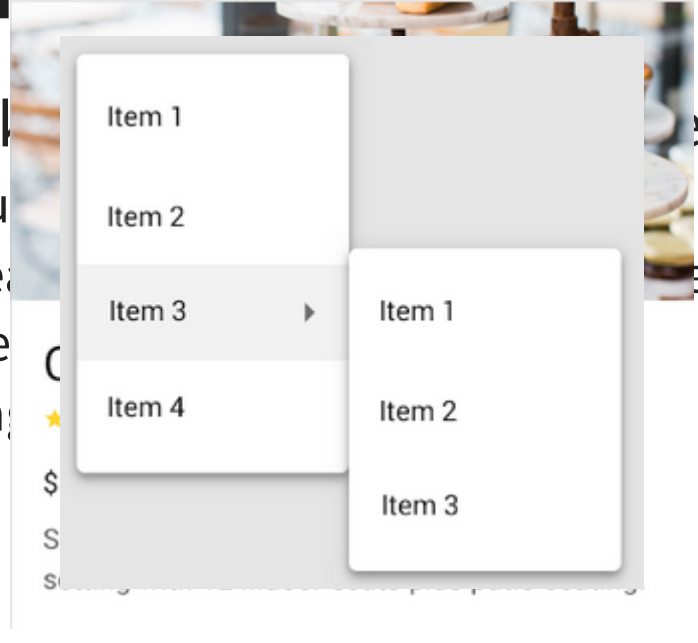
★★★★½ 4.5 (413)

$ • Italian, Cafe

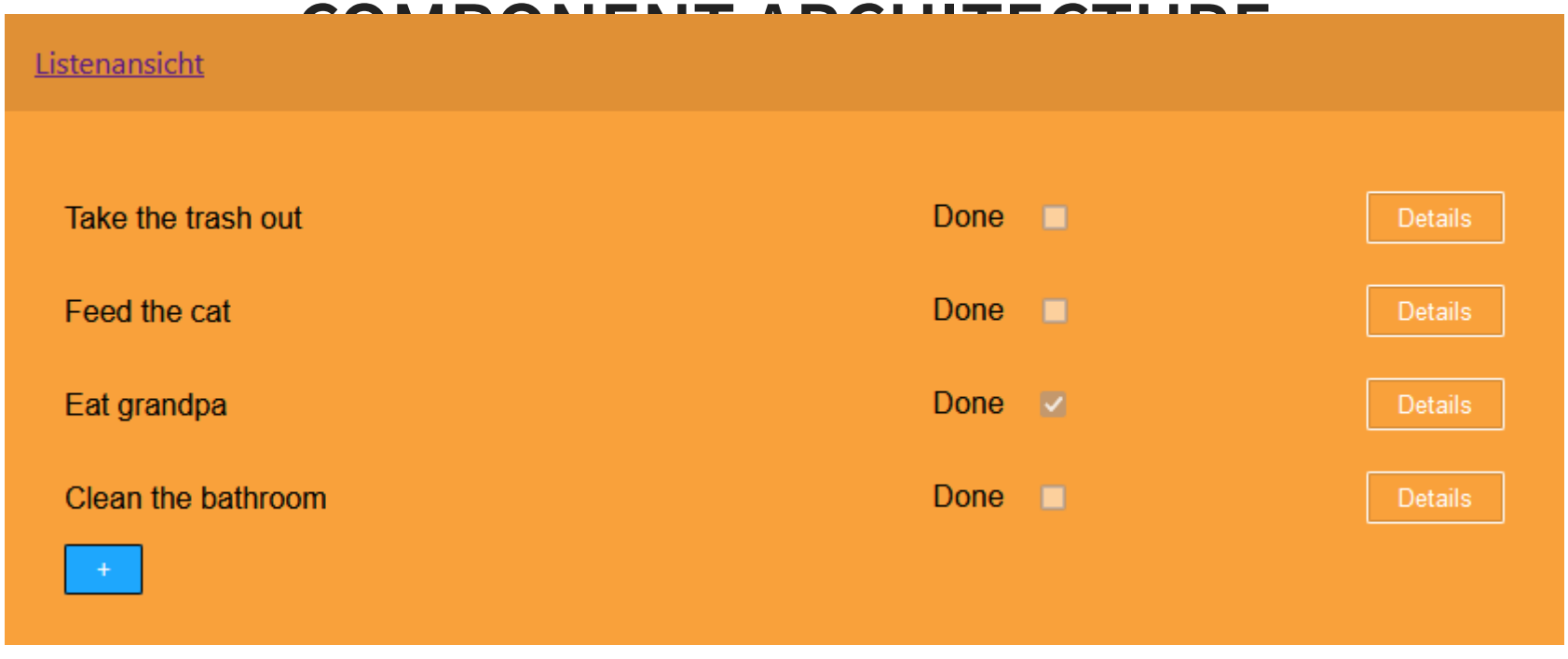
Small plates, salads & sandwiches an intimate setting with 12 indoor seats plus patio seating.

COMPONENT ARCHITECTURE

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



COMPONENT ARCHITECTURE



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

```
export default function Button({ primary, label,
                                 onClick, className }) => {
    const mode = primary ?
        'button--primary' : 'button--secondary';
    return (
        <button
            type="button"
            className={['button', mode, className].join(' '
            onClick={() => onClick()}
        />
        {label}
        </button>
    );
};
```

COMPONENTS IN REACT

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

```
export default function Button({ ... }) {
    ...
};
Button.propTypes = {
    primary: PropTypes.bool,
    label: PropTypes.string.isRequired,
    onClick: PropTypes.func,
    className: PropTypes.string,
};
Button.defaultProps = {
    primary: false,
    onClick: undefined,
    className: '',
};
```

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 (branch: step_3-component_architecture)

ATOMIC DESIGN

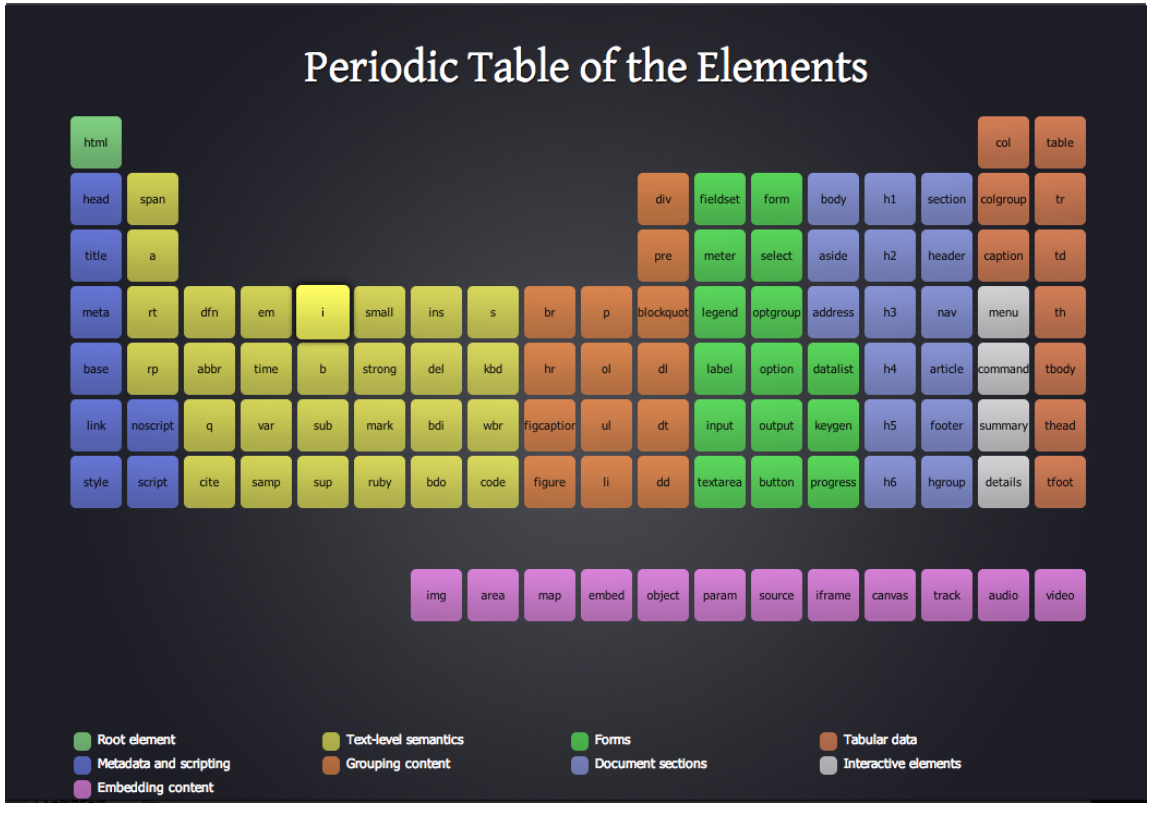
ATOMIC DESIGN

große Frontends mit vielen Components werden unübersichtlich

ATOMIC DESIGN

Strukturierung und Kategorisierung von Components

Ziel ist ein ordentlicher Baukasten an Components



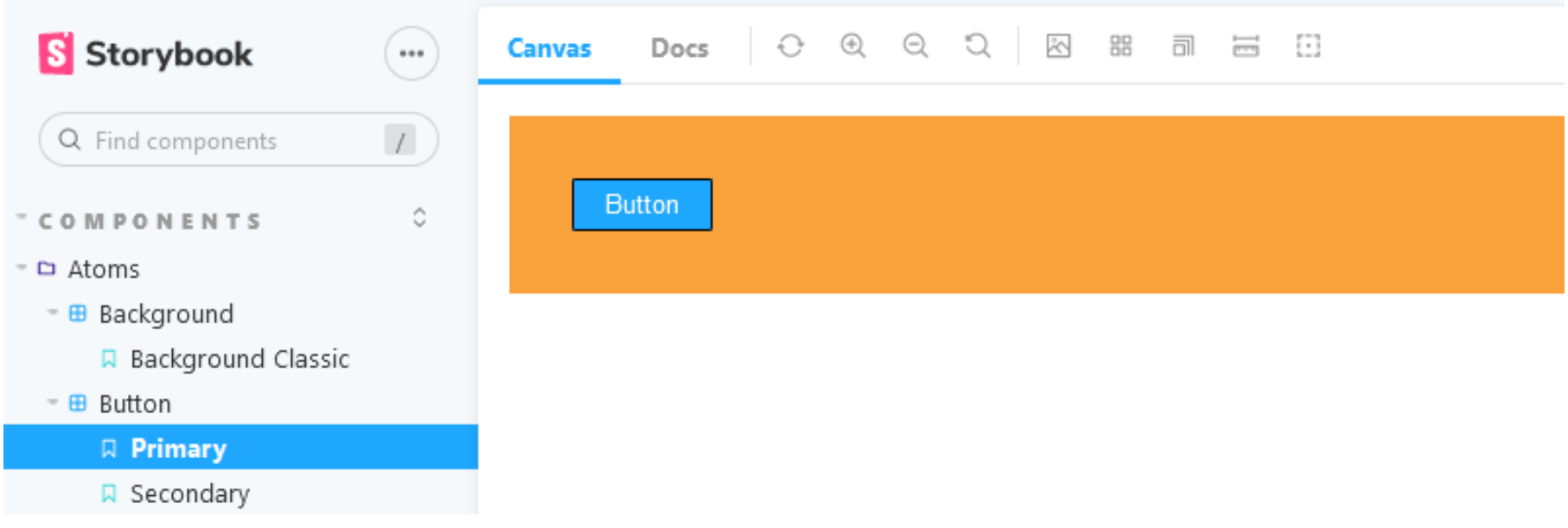
https://bradfrost.com/blog/post/atomic-web-design/

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

```
export default {
    title: 'Components/Background',
    component: Background,
};

const Template = () => <Background />;

export const BackgroundClassic = Template.bind({});
```

STORYBOOK SYNTAX

```
export default {
    title: 'Components/Background',
    component: Background,
};

const Template = () => <Background />;

export const BackgroundClassic = Template.bind({});
```

STORYBOOK SYNTAX

```
export default {
    title: 'Components/Background',
    component: Background,
};

const Template = () => <Background />;

export const BackgroundClassic = Template.bind({});
```

STORYBOOK SYNTAX

```
export default {
    title: 'Components/Background',
    component: Background,
};

const Template = () => <Background />;

export const BackgroundClassic = Template.bind({});
```

STORYBOOK MIT PARAMETERN

```
const Template = (args) => <Background>
                               <Button {...args} />
                           </Background>;

export const Primary = Template.bind({});
Primary.args = {
    primary: true,
    label: 'Button',
};

export const Secondary = Template.bind({});
Secondary.args = {
    label: 'Button',
};
```

STORYBOOK MIT PARAMETERN

```
const Template = (args) => <Background>
                                <Button {...args} />
                           </Background>;

export const Primary = Template.bind({});
Primary.args = {
    primary: true,
    label: 'Button',
};

export const Secondary = Template.bind({});
Secondary.args = {
    label: 'Button',
};
```

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

```
const someMockClient = {
    getData(_) {
        return Promise.resolve(new Data('lala'));
    },
    saveData(data) {
        return Promise.resolve(data);
    }
}

const Template = (args) =>
            <SomeContext.Provider value={someMockClient}>
                <SomeComponent {...args} />
            </SomeContext.Provider>;
```

STORYBOOK MIT ROUTER

```
export const reactRouterDecorator = (Story) => {
    return (
        <MemoryRouter>
            <Routes>
                <Route path="/*" element={<Story />} />
            </Routes>
        </MemoryRouter>
    )
}
```

```
export default {
    title: 'Components/SomeComponent',
    component: SomeComponent,
    decorators: [reactRouterDecorator],
};
```

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 (branch: step_3-atomic_design)

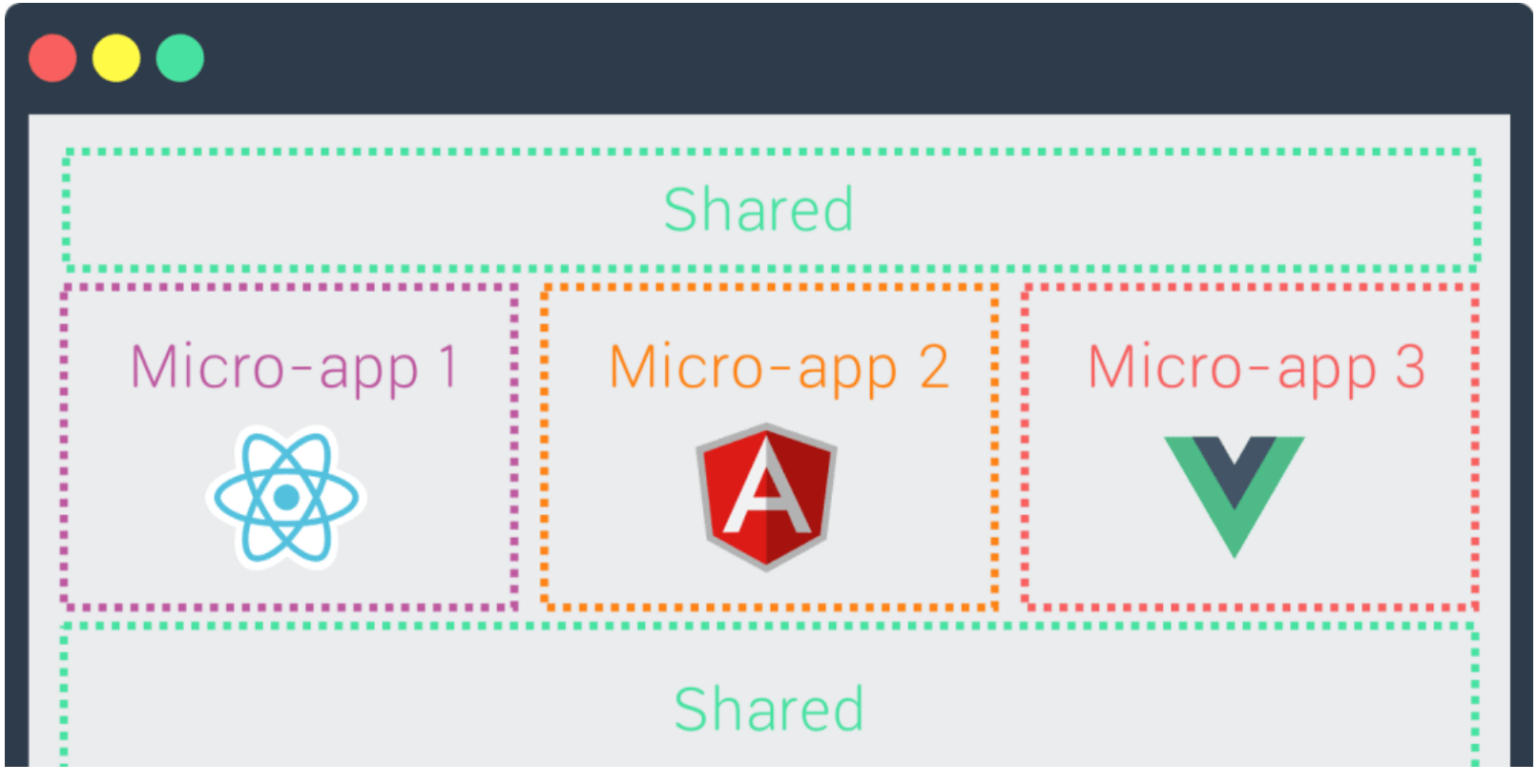
MICRO FRONTENDS

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

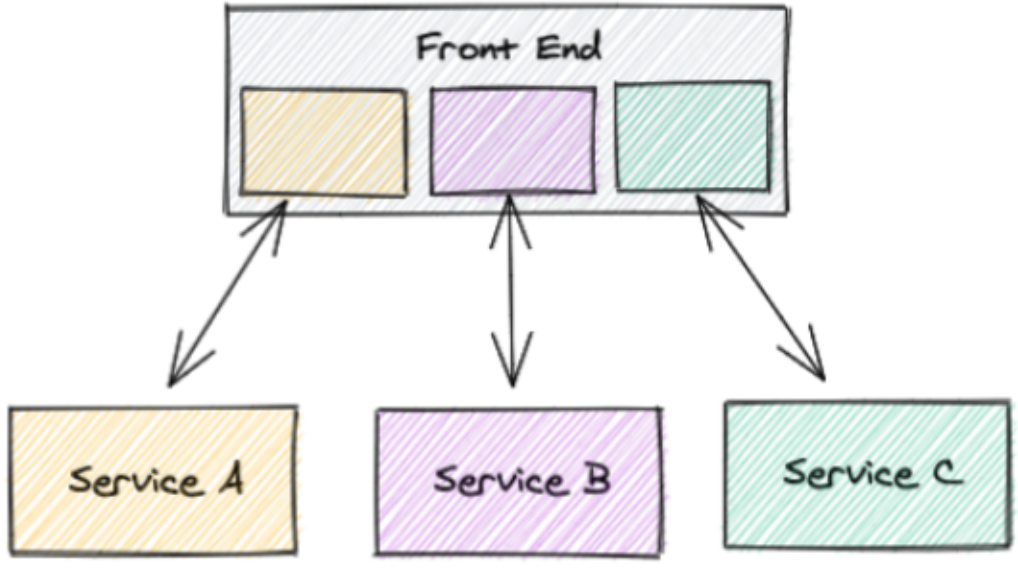
MICRO FRONTENDS

- 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?