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

Speaker notes

- Bevor wir jetzt direkt in React einsteigen, lasst uns erstmal herausfinden, wofür React überhaupt gut ist.

SPA IN PLAIN JAVASCRIPT?

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

Speaker notes

- Hat jemand bedenken wenn wir das machen?

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

Speaker notes

- Also Web Service starten, HTML Seite anlegen und loslegen.

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)

Speaker notes

- Wir schreiben nur zwei Seiten und das auch gerne ohne viel Styling.
- Es geht mir nur darum, mal zu zeigen wie man sowas ohne ein Framework bauen würde.
- Beispielanwendung zeigen, wie dies aussehen könnte.

PRAXIS: WAS BRAUCHEN WIR?

Speaker notes

- In der index.html Datei brauchen wir einen Bereich, in dem unsere zwei Seiten angezeigt werden.
- Möglich wäre auch ein Menü, dass auf beiden Seiten existiert, um auf die Startseite zurück zu kehren.
- XMLHttpRequests kennt ihr bereits aus der letzten Vorlesung, die könnt ihr hier anwenden.
- Per Javascript müssen wir dann das HTML so editieren, dass die Daten vom Backend entsprechend angezeigt werden können.
- Wer die syntax zum editieren von HTML nicht mehr kennt, kann sich gerne dem Internet bedienen.
- Die Listenansicht soll initial angezeigt werden, dass heißt wir brauchen eine Art initialisierung.

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

Speaker notes

- Meine Beispiellösung ist nicht perfekt.
- Versucht gerne eine bessere oder schönere Lösung zu bauen!

SPA IN PLAIN JAVASCRIPT?

Was ist aufgefallen bei einer Implementierung in plain JavaScript?

Speaker notes

- Das schreiben einer SPA in plain JavaScript ist sehr aufwendig.
- Backend Requests, DOM Manipulation sind Aufgaben die in jeder Application auftreten. Braucht man nicht immer neu zu schreiben.
- Routing ist ebenfalls aufwendig. Umbau der Seite, Anpassung der URL.

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

Speaker notes

- React macht vieles einfacher, was wir in plain JavaScript selbst erledigen mussten.
- DOM Manipulation wird durch die JSX Syntax wesentlich einfacher.
- Routing mit entsprechender URL Manipulation wird zum Kinderspiel.
- Axios ist eine vorgefertigtes Tooling um XMLHttpRequests abzusenden.
- Change Detection ist schon mal ein Schlagwort, dass ihr euch merken könnt. Auch darauf werde ich später noch genauer eingehen, wenn es wichtiger für uns wird.

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 }
```

Speaker notes

- React funktions geben als return Wert eine Art HTML Syntax zurück.
- Es darf maximal ein toplevel Tag geben.
- Über geschweifte Klammern können wir dynamische Daten in das HTML einfügen.
- Das macht das Rendering von dynamischen Informationen wesentlich einfacher als mit plain Javascript.

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 }
```

Speaker notes

- In den geschweiften Klammern können wir nicht nur Variablen einbinden, sondern auch verschiedene Expressions nutzen.

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 }
```

Speaker notes

- Auch in den geschweiften Klammern können wir über die JSX Syntax wieder HTML Tags einfügen.

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>
    );
}
```

Speaker notes

- Mit dem BrowserRouter wird eine Stelle markiert, an der die einzelnen Seiten angezeigt werden.
- Die Routes definieren alle Seiten, zu denen navigiert werden kann.
- Mehrere Pfade können zu gleichen Seite führen.
- Der Parameter "element" einer Route, definiert die React Function, zu der navigiert wird.
- Über ein :someParam kann ein Parameter bei der Navigation mitgegeben werden.

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>
}
```

Speaker notes

- Über die Funktion "useNavigate()" bekommt man eine Funktion, mit der man eine Navigation auslösen kann.
- Dieser Funktion übergibt man nun den Pfad. Dabei können natürlich auch Parameter eingefügt werden.

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 }
```

Speaker notes

- Über die Funktion "useParams()" bekommt man ein Objekt mit alle übergebenen Parametern.
- Der Name des Parameters ist dabei der, der in der Route hinter dem Doppelpunkt definiert wurde.
- Über Javascript Destructing wird der Parameter hier aus dem Objekt destructed.

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 }
```

Speaker notes

- Wie ihr sehen könnt brauchen wir nur die URL.
- Als Response bekommen wir ein Javascript Objekt, welches axios bereits aus dem JSON geparsed hat.

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);
}
```

Speaker notes

- Auch POST Requests funktionieren mit axios ziemlich einfach.
- Wir können das Javascript Objekt, das gespeichert werden soll, einfacher als solches übergeben.
- Axios kümmert sich um die JSON Serializierung.

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 }
```

Speaker notes

- Um die Funktionen zum laden und speichern von Daten auf verschiedenen Seiten nutzen zu können, sollen wir sie in eine Klasse auslagern.
- Außerdem bleiben so die React functions frei von solcher Logik und befassen sich nur mit dem Rendering

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 }
```

Speaker notes

- Braucht ihr noch mal eine Erläuterung zu Dependency Injection?
- Über den React Context können wir gewisse Objekte in teilen des DOM's verfügbar machen.
- Mit der Funktion "createContext" erstellt man ein Kontext Objekt von einem gewissen Klassentyp.
- Nun hängen wir einen Provider dieses Kontexts in dem DOM ein.
- Dieser Provider erhält nun das konkrete Objekt, das unter dem Provider im DOM verfügbar ist.

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()

Speaker notes

- React Hooks sind schon fast ein Thema für sich.
- Wenn wir Daten in einer React function speichern wollen, können wir das mit useState() -> gleich mehr dazu
- In unserer Plain Javascript SPA haben wir auch Code für die initialisierung gebraucht. Bzw. wir haben die einzelnen Seiten initial geladen.
- In einer React function können wir nicht einfach asynchronen Code ausführen. Es muss immer direkt etwas zum Rendern zurückgegeben werden.
- Daher gibt es Effects, um asynchron etwas vom Backend zu laden. Hier greifen wir dann auf den Client den wir geschrieben haben zu.

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 }
```

Speaker notes

- Mit der Funktion useState() können wir Daten in einer React function speichern.
- Die useState() function gibt uns ein Attribut "count" über das wir auf den Wert zugreifen können.
- Der zweite Rückgabewert ist eine function, mit der wir den Wert verändern können.
- Bennenen können wir die Rückgabewerte wie wir wollen. (hier wird wieder mit destruction gearbeitet)
- Eine normale Variable würde ihren Wert verlieren, wenn die React function erneut aufgerufen wird.
- useState() erzeugt einen state für die function der besteht. Außerdem wird die React function neu gezeichnet, sobald sich ein state ändert.
- Auch über das onChange event des input fields wird der state neu gesetzt.
- Angezeigt wird der aktuelle Wert, in dem wir die "count" variable als Label des Buttons anzeigen.

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 }
```

Speaker notes

- Mit der Funktion useEffect() definieren wir einen Callback, der zu gewissen Zeitpunkten automatisch von React augerufen wird.
- In diesem Beispiel wird der Callback zum Zeitpunkt der Initialisierung der React function aufgerufen.
- Die React function wird ausgeführt und anschließend wird der Request gegen das Backend gestartet.

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 }
```

Speaker notes

- useEffect() wird nicht nur zur Initialisierung der React function aufgerufen, es wird auch aufgerufen, wenn sich ein state ändert.
- Dieser Code würde daher eine Endlosschleife auslösen. Jedes mal wenn "setData" aufgerufen wird, wird der Callback erneut ausgeführt.

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 }
```

Speaker notes

- useEffect() löst nun keine Endlosschleife mehr aus.

REACT HOOKS

- useEffect()
  - kann zum Aufräumen verwendet

```
export default function Screen2() {
    ...
    const [data, setData] = useState('')

    useEffect(() => {
        dataHttpClient.getData(data)
            .then((data) => setData(data));
        return () => console.log('teardown');
    }, []);

    return <input
            value={data}
            onChange={(e) => setData(e.target.value)} />;
}
```

Speaker notes

- Der Rückgabewert des Callbacks des useEffect() Hooks wird aufgerufen, wenn die React function abgeräumt wird.

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

Speaker notes

- Component Architecture könnte man sich wie Lego vorstellen

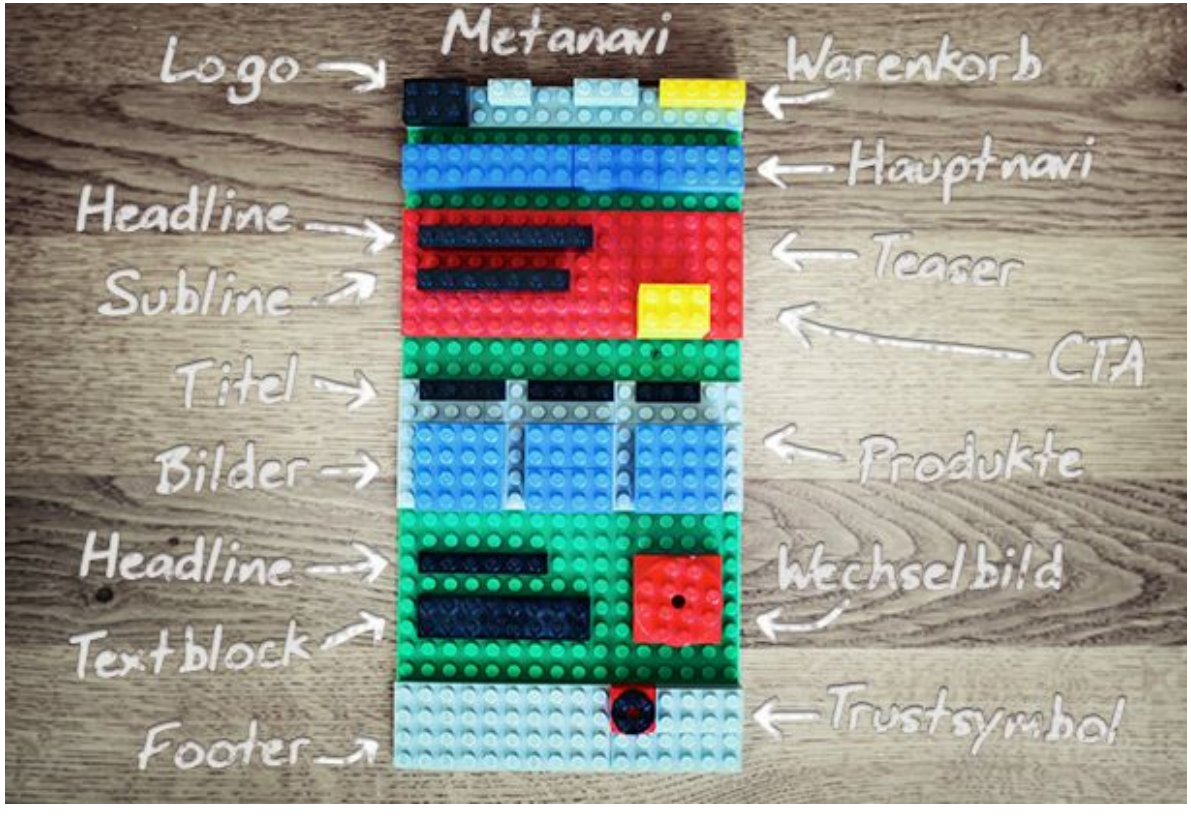
COMPONENT ARCHITECTURE

- 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

Speaker notes

- Die Idee ist grundlegend seine Webseite in einzelne Components aufzuteilen.
- Damit teilt man die Komplexität seiner Seite in kleinere Teile (Components).
- Quasi wie man es aus dem klassischen Softwareengineering kennt. Dort wird auch funktionalität die zusammengehört in Komponenten zusammengefasst.
- Ob das nun Klassen sind (OOP) oder Funktionen, ist egal.
- Über Schnittstellen (vergleich zu Lego die Noppen), können Components dann wieder zusammengesteckt werden.

COMPONENT ARCHITECTURE



COMPONENT ARCHITECTURE

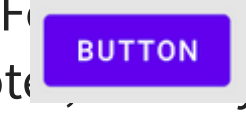
- Was kann alles eine Component sein?
  - Buttons, Text Fields, Labels, etc.
  - Search Bar, Form Groups, Cards, etc.
  - Header, Footer, Overlays, etc.
  - Pages

Speaker notes

- Unter einer Card kann man sich gebündelten Content vorstellen. Möglicherweise mit Bild und Edit Button oder so?
- Eine Component kann also ein sehr kleiner Teil der Anwendung sein, wie z.B. ein einzelner Button
- Eine Component kann aber auch ein Abschnitt sein oder sogar eine ganze Seite, die sich mit einem bestimmten Thema beschäftigt.

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

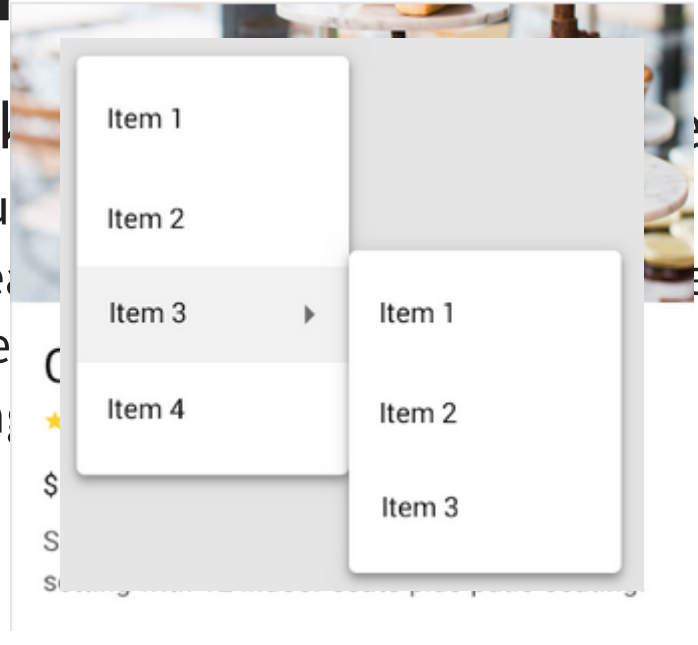
★★★★½ 4.5 (413)

$ • Italian, Cafe

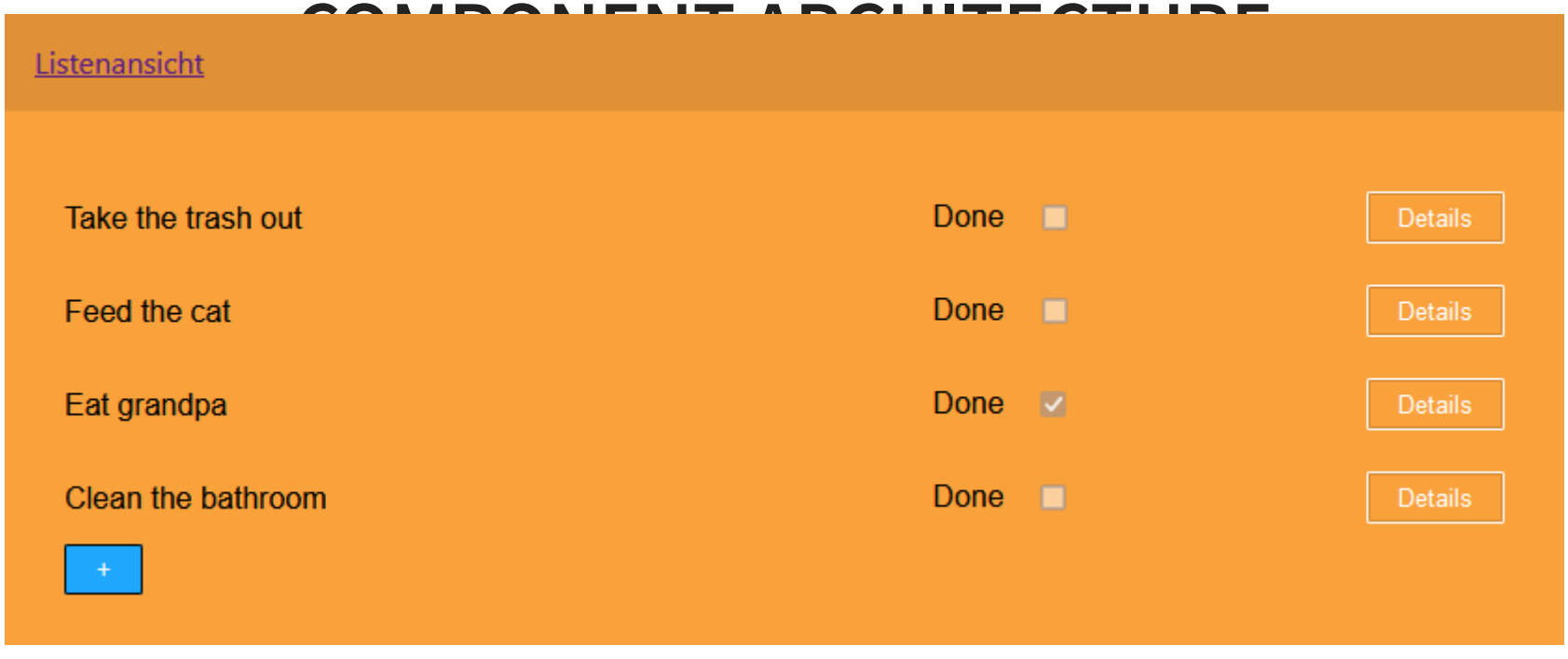
Small plates, salads & sandwiches an intimate setting with 12 indoor seats plus patio seating.

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

Speaker notes

- wir setzen mit der Component Architecture auf klassischen HTML Elementen auf und bauen daraus eigene Components

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>
    );
};
```

Speaker notes

- In diesem Beispiel geben wir mehrere Parameter in die Button function hinein.
- Primary ist ein bool, der Styling zum Button hinzufügt.
- Über das Label setzen wir einen Text.
- ClassName fügt weiteres Styling hinzu.
- OnClick ist ein Callback, mit dem wir zurückmelden können, wenn der Button geklickt wurde. Damit der Nutzer des Buttons weis was passiert.

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 };
```

Speaker notes

- Damit der Nutzer des Buttons weis, was er hineingeben kann und muss, können wir die propTypes verwenden.
- Wir sollten eine klare Schnittstelle definieren, damit der Entwickler, der den Button nutzen möchte, nicht in die Implementierung schauen muss.

COMPONENT ARCHITECTURE

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

Speaker notes

- Konsistenz
  - Komponenten wie Buttons gehören zu Atomen und sollten wiederverwendet werden.
  - Dies spart Zeit, außerdem sehen die Button überall gleich aus. Sorgt für Konsistenz im Styling
- Schnellere Entwicklung
  - Ich muss den Button nicht noch mal für eine andere Seite Stylen oder mit den Code dazu kopieren.
  - Ich kann auf bereits basierende Strukturen aufbauen.
- tiefe Verschachtelungen
  - Große Seiten und Anwendungen kämpfen häufig mit einer sehr hohen Verschachtelungstiefe
  - Durch Komponenten die kein Styling hinzufügen, sondern nur Logik bereitstellen und teilen, wird die Wrapper Hölle noch schlimmer.
  - Dies ist nicht sehr übersichtlich.

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

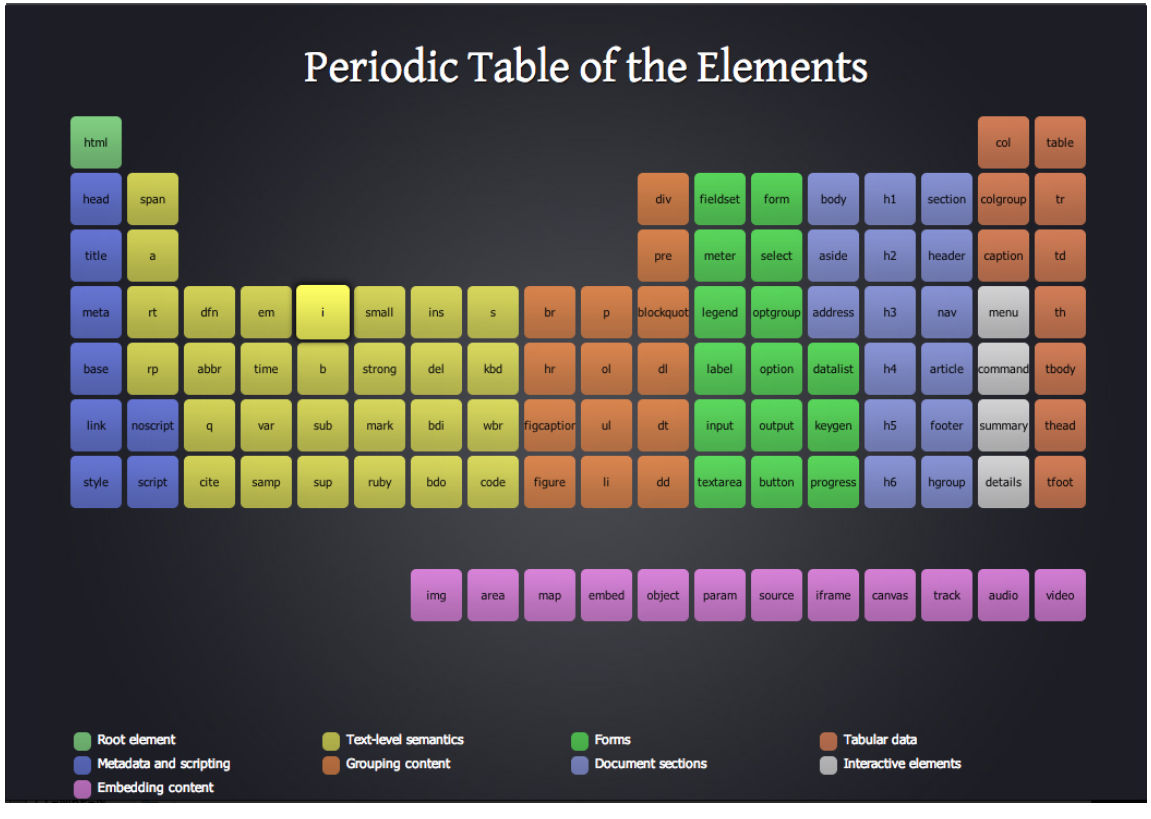
ATOMIC DESIGN

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

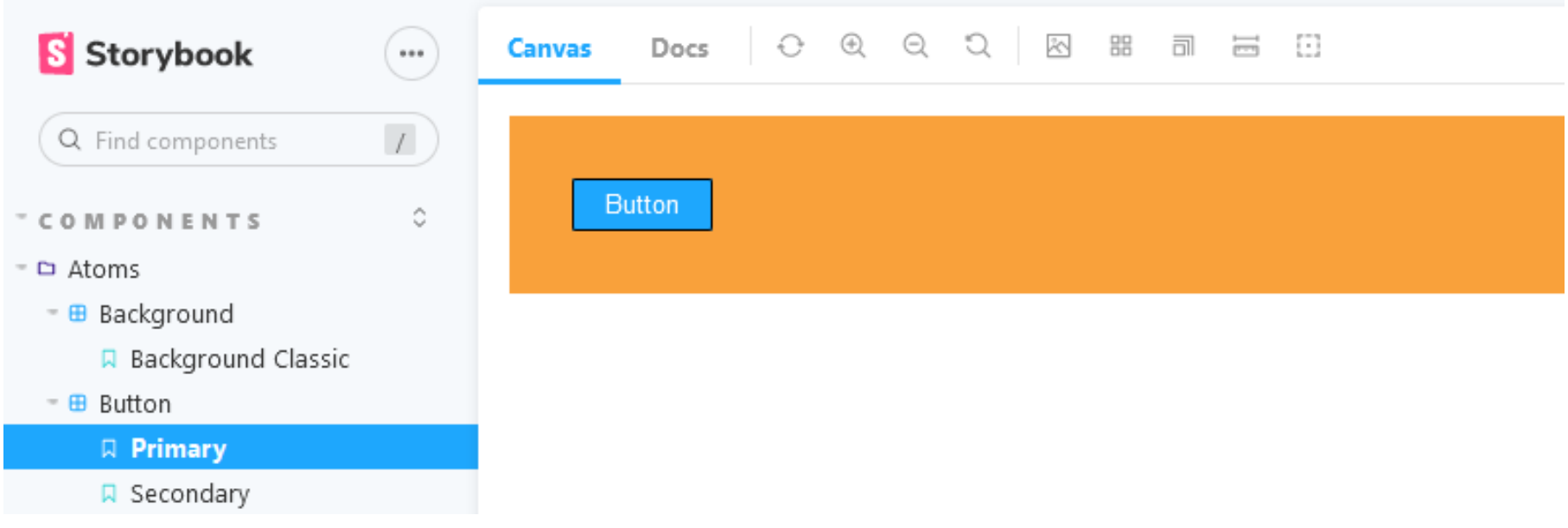
- Atomic Design ordnet Components nach:
  - Atoms - Buttons, Text Fields, etc.
  - Molecules - Search Bar, Form Groups, etc.
  - Organisms - Header, Footer, Overlays, etc.
  - Templates - Schablone
  - Pages - konkrete Seite

Speaker notes

- https://bradfrost.com/blog/post/atomic-web-design/
- Atoms - die Bausteine unserer Anwendung - Buttons, etc.
- Molecules - kleine zusammenschlüsse von Atoms - Suchfelder, Form Groups
- Organisms - fachliche Components. Zusammenschlüsse von Molecules mit denen der User interagieren kann
- Templates - Schablone die den Aufbau der Seite zeigt
- Pages - konkrete Seiten

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 (branch: step_3-atomic_design)

MICRO FRONTENDS

COMPONENT ARCHITECTURE

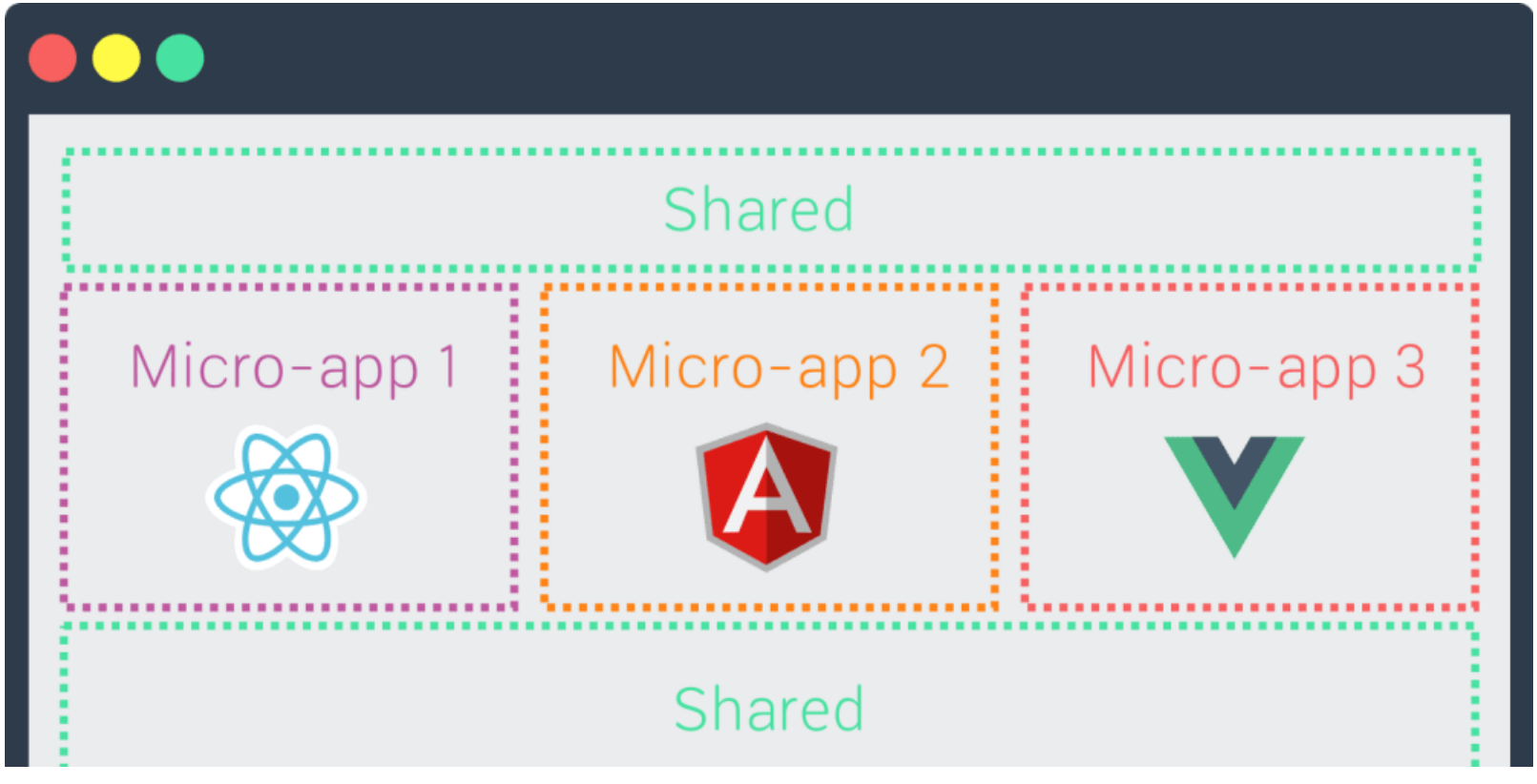
- bringt uns Ordnung und Struktur
- was passiert wenn das Frontend wächst?
- mehrere Teams arbeiten an einem Frontend?
- unterschiedliche Teams
  - mögen unterschiedliche Technologien
  - haben unterschiedliche Arbeitsweisen
  - möchten unabhängig releasen
  - haben unterschiedlichen Codestyle

Speaker notes

- Auch bei einer ordentlich gepflegten Component Architecture wird ein Frontend irgendwann zu groß.
- Meistens wächst das Team ebenfalls, da die Anforderungen wachsen.
- Das Team teilt sich dann meist schon von selbst, da ein einzelner nicht mehr die ganze Domäne überblicken kann.
- Irgendwann lohnt es sich das Team und das Frontend offiziell zu teilen.
- Sonst ist es absehbar, dass die Produktivität sinkt.
- Zum Beispiel durch:
  - Mergeconflicts durch zu viele Änderungen.
  - Konflikte in einem großen Team durch
    - unterschiedlichen Codestyle.
    - unterschiedliche Arbeitsweisen.
    - unterschiedlich favorisierte Technologien.
  - Features können nicht unabhängig voneinander released werden.

MICRO FRONTENDS

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

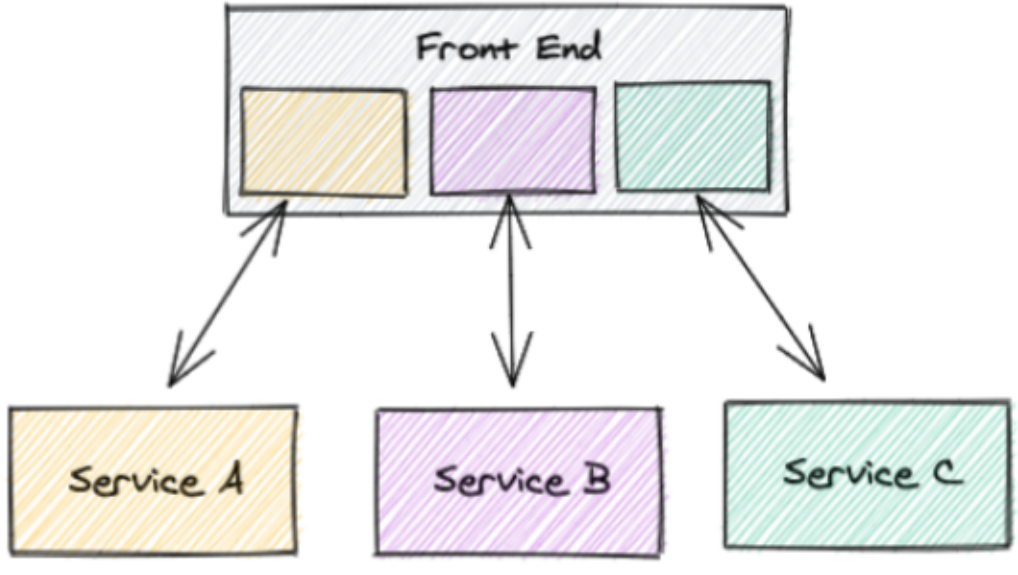


Speaker notes

- Micro Frontends können bei Bedarf zu einem Frontend zusammengesteckt werden.
- Dies muss aber nicht sein, evtl. wird über eine Navigation von einem zum anderen Frontend navigiert.
- Micro Frontends können von verschiedenen Teams mit verschiedenen Sprache und Frameworks gebaut werden.
- Evtl. auch ein eigener release Zyklus.
- Micro Frontends können auch helfen, wenn man eine legacy Anwendung Stück für Stück erneuern möchte.

MICRO FRONTENDS

- reden meist auch mit eigenen Backends
- Micro Services



Speaker notes

- Micro Frontends sind aus Micro Services entstanden
- um auch die Frontends Skalierbarer zu machen
- dies ist nur bei wirklich komplexen Anwendungen zu empfehlen