1 Introduction SPA

1.1 Browser-based Applications

Benefits

- Work from anywhere, anytime
- Platform independent, including mobile
- No software update, no application, easy maintenance
- Software can be provided as a service (SaaS - pay as you go)
- Code separation

Liabilities

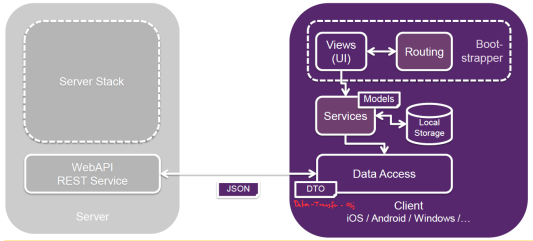
- No data sovereignty (Datenhoheit)
- Limited/restricted hardware access
- SEO - Search engines must execute JavaScript
- More complex deployment strategies

1.2 SPA

A website that fits on a single web page with a user experience similar to that of a desktop application. All code is retrieved with a single page load or resources are dynamically loaded. SPAs use AJAX and HTML5 to create responsive Web apps, without constant page reloads.

1.2.1 Architecture

Website interacts with user by rewriting parts of the DOM. After first load, all interaction with the server happens through AJAX.



1.2.2 Bundling

All JS code must be delivered to the client over potentially slow networks. Bundling and minifying the source leads to smaller SPA footprint. Larger SPAs with many modules need a reliable dependency management. Initial Footprint can be reduced by loading dependent modules on-demand.

1.2.3 WebPack as Bundler

Entry: Start, follows the graph of dependencies to know what to bundle.
Output: Tell webpack where to bundle your application.
Loaders: Transforms these files into modules as they are added to your dependency graph.
Plugins: Perform tasks like bundle optimization, asset management and injection of env variables.
Mode: Enable built-in optimization mechanisms.

1.3 Routing

- Completely on client-side by JS
- Navigation behaves as usual
- Browser needs to fake the URL to change and store page state
- *window.history.pushState*

1.4 Dependency Injection

Benefits

- Reduces coupling between consumer and implementation
- Contracts between classes are based on interfaces
- Supports the open/closed principle
- Allows flexible replacement of an implementation

1.4.1 Decorators

- Provide a way to add annotations / meta-programming syntax
- Can be attached to a class declaration, method, accessor, property or parameter
- Widely used in Angular

2 React

- Library, kein Framework
- Um User Interfaces zu bauen
- View in MVC
- Minimales Featureset
- Entwickelt von Facebook
- Verwendet für: WhatsApp, Insta, AirBnb, etc.

2.0.1 Prinzipien

- Komplexes Problem aufteilen in einfachere Komponenten
- Für eine bessere: Wiederverwendbarkeit, Erweiterbarkeit, Wartbarkeit, Testbarkeit, Aufgabenverteilung

2.1 Entwicklung von UIs

- Beschreibung des UIs
- Event-Handling
- Aktualisieren der Views

2.2 Komponenten und Elemente

- Funktionen die HTML zurückgeben
- Beliebige Komposition von React-Elementen und DOM-Elementen

```
function App() {
 return (
   <div>
     <HelloMessage name="HSR" />
     <img src="/logo.png" />
   </div>
 )
}
```

Parameterübergabe an Funktion

Äquivalent zu Attribut für DOM-Element

2.3 JavaScript XML

React verwendet JSX (blau), eine Erweiterung von JavaScript (gelb). Überall wo JSX verwendet wird, muss *react* importiert werden.

```
const menu = entries.map(entry =>
  <ListItem as="a" to={`/${entry.path}`}>
    <h1>{entry.title.toUpperCase()}</h1>
    <p>{entry.subtitle}</p>
  </ListItem>
)
```

Styles: werden nicht als Strings sondern als Object angegeben.

2.3.1 Conditionals

```
<Container>
 { error &&
   <Message>
     Fehler: {error}
   </Message>
 }
</Container>
```

oder

```
<Container>
 { error
   ? <span>
       Fehler: {error}
     </span>
   : <span>OK!</span>
 }
</Container>
```

2.3.2 Props

Komponenten erhalten alle Parameter/Properties als **props** Objekt.

- *this.props* bei Klassen
- Bei Funktionen als Parameter
- Immer **read-only**

2.3.3 Rendering und Mounting

Mounting: nötig um Komponenten auf Webseite anzuzeigen. *ReactDOM.render*

```
ReactDOM.render(
    <App/>
    document.getElementById('root')
)
```

2.4 React State

React-Klassenkomponenten können einen veränderbaren Zustand haben. Der **state** einer Komponente ist immer privat. Ändert der State, wird auch die Komponente aktualisiert.

```
class Counter extends React.Component {
    state = { counter: 0 }
    // ...
}
```

2.4.1 Event Handler

```
const increment = () => {
    this.setState({counter: this.state.counter +
        1})
} // ...
<button onClick={this.increment}>
```

2.5 Reconciliation

1. React Komponenten werden als virtueller DOM gerendert
2. Wird der **state** geändert, erstellt React einen virtuellen DOM
3. Alter und neuer DOM werden verglichen
4. Erst dann werden geänderte DOM-Knoten im Browser erstellt