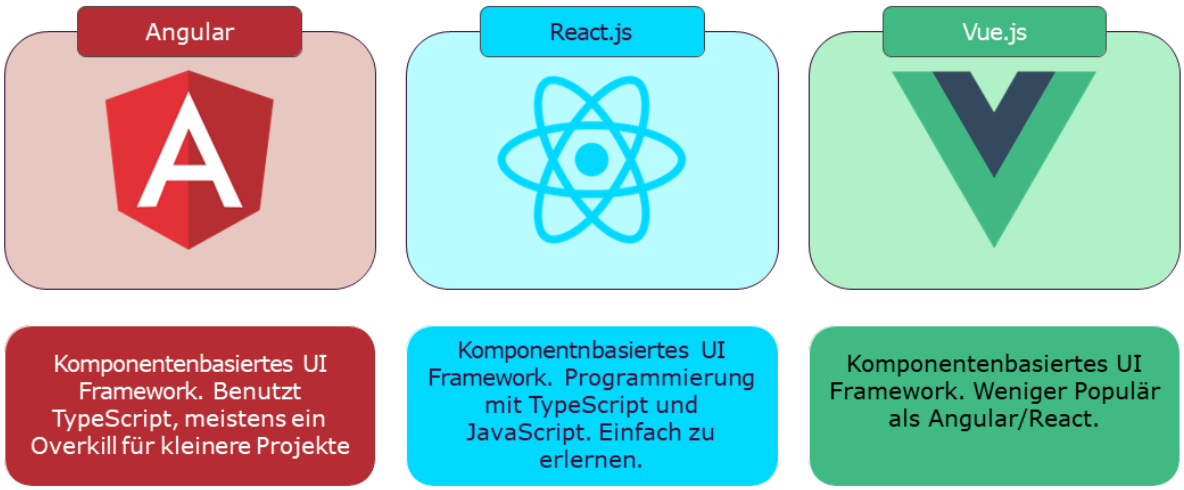
4 Einführung React

4.1 Frameworks zur Komponentenweisen UI Programmierung



Momentan gibt es drei grosse “konkurrenzierende” Frameworks zur Erstellung von Web-Apps. Alle drei wählen einen komponentenweisen Ansatz zur Erstellung von Applikationen im Webbrowser, respektive auf mobilen Geräten.
Wir sehen und in den nächsten Wochen “React.js” oder kurz React an.

4.2 Was ist React.js ?

React ist eine JavaScript Bibliothek um User-Interfaces zu erstellen. Die Bibliothek erschien erstmals 2013 und wird von Facebook Inc. und einer Open Source Community entwickelt. Die aktuelle Version ist 18.x und erschien 2022. React ist Open Source und der gesamte Code ist auf GitHub erhältlich: https://github.com/facebook/react

React ist eine Client-Side Bibliothek, d.h. alles läuft lokal im Browser nachdem die Webseite heruntergeladen wurde.

Einige Beispiele für Webseiten, welche mit React gemacht sind:

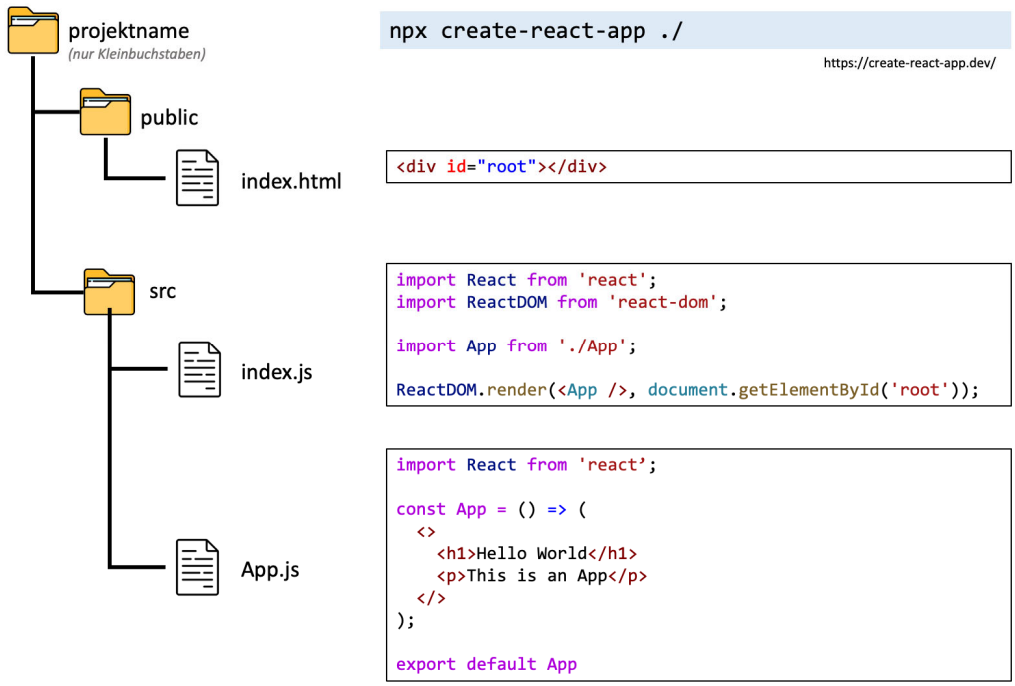
Facebook, WhatsApp, Netflix, Instagram, Airbnb, Uber, BBC, PayPal, Dropbox, Reddit

Mit React können wir sogenannte “**Single Page Applications**” (SPA) erstellen, das heisst eine einzelne Webseite wird heruntergeladen und wir können auf dieser interagieren.

Bei React steht die **Wiederverwendbarkeit** von Komponenten im Zentrum.

4.3 Ein erstes Projekt

React Projekte sollten eine vorgegebene Datei-Struktur verwenden. Zunächt haben wir einen Ordner mit dem Projektnamen, dieser darf nur aus Kleinbuchstaben bestehen. Im Projektordner haben wir einen Ordner «public» in diesem ist der HTML drin und ein Ordner «src», in welchem JavaScript und CSS Files enthalten sind.



Am einfachsten lässt sich ein komplett neues Projekt über

```
npx create-react-app ./
```

innerhalb des Projektordners erstellen. Danach kann in der Regel der Inhalt des src Ordners gelöscht werden und durch die Minimalkonfiguration ersetzt werden.

Es ist wichtig, dass dieser Ordner in Visual Studio Code direkt geöffnet ist und keine tiefere Struktur und andere Daten sichtbar sind.

1 Projekt = 1 Ordner

Nachdem das Projekt erstellt ist, löschen wir alles im «public» und im «src» Ordner und Erstellen folgende Files neu:

public/index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>React App</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

src/index.js

```
import React from 'react';
import ReactDOMClient from 'react-dom/client';

import App from './App';

const root =
ReactDOMClient.createRoot(document.getElementById('root'))
root.render(<App />)
```

src/App.js

```
import React from 'react';

function App() {
    return (
    <>
        <h1>Hello World</h1>
        <p>This is an App</p>
    </>);
}

export default App;
```

Nun können wir das Projekt starten indem wir in der Konsole folgendes eintippen:

```
npm start
```

Der Standard Webbrowser wird geöffnet und wir sehen unsere erste einfache App.

In diesem Beispiel haben wir einen Komponente «App» erstellt, welche ganz einfach einen Titel und einen Paragraphen erstellt. Was ist aufgefallen? Wir haben HTML Tags direkt in JavaScript geschrieben. Wir brauchen also praktisch keine HTML Files mehr zu editieren und können alles über JavaScript erledigen. Diese Tags sind effektiv kein HTML sondern JSX (JavaScript XML), welches es ermöglicht HTML Code in JavaScript zu schreiben. Dies ist React-Spezifisch.

JSX kann templates verwenden, verändern wir App.js folgendermassen:

```
import React from 'react';

function App() {
    var template = "Dies ist ein Template Text";

    return (
    <>
        <h1>Hello World</h1>
        <p>{template}</p>
    </>);
}

export default App;
```

Innerhalb von geschweiften Klammern können Variablennamen und Funktionen stehen, welche dann einfach ersetzt werden!

Machen wir doch noch einen Button hinzu und eine Zählvariable.

```
import React, { useState } from "react";

function App() {
    const [count, setCount] = useState(0);

    function increment() {
        setCount(count+1);
    }

  return (
    <>
    <p>{count}</p>
    <button onClick={increment}>Click Here</button>
    </>
  );
}

export default App;
```

4.4 Objektorientierer Ansatz

Komponenten können auch mehr objektorientiert mit Klassen erstellt werden.

```
import React, {Component} from "react";

class App extends Component {
    render() {
        return <h1>Hello World</h1>
    }
}

export default App;
```

Wenn wir eine Klasse erzeugen müssen wir die “render()” Methode überschreiben um die Komponente darzustellen. Es gibt noch einige weitere wichtige Methoden, welche wir überschreiben können.

4.4.1 Konstruktor

Der Konstruktor wird innerhalb der Klasse als erstes definiert. Im Konstruktor werden vor allem die State-Variablen gesetzt.

```
constructor(props) {
    super(props)
    this.state = {count : 0 };
}
```

4.4.2 componentDidMount()

Diese Methode wird aufgerufen, sobald die Komponente in den DOM Baum hinzugefügt wird.

4.4.3 componentDidUpdate()

Diese Methode wird aufgerufen, sobald die Komponente einen Update erhalten hat.

4.4.4 componentWillUnmount()

Diese Methode wird aufgerufen, sobald die Komponente aus dem DOM Baum entfernt wird.

Wir können so relativ einfach einen Counter, welcher mit setInval alle n Millisekunden aufgerufden wird implementieren:

```
import React, {Component} from "react";

class App extends Component {
    constructor(props) {
        super(props);
        this.state = {count: 0};

        // Event-Handler registrieren:
        this.update = this.update.bind(this);
    }

    update() {
        this.setState({ count: this.state.count + 1 });
    }

    componentDidMount() {
        this.interval = setInterval(this.update, 500);
    }

    componentWillUnmount() {
        clearInterval(this.interval);
    }

    render() {
        return (
        <>
            <h1>Hello World</h1>
            {this.state.count}
        </>)
    }
}

export default App;
```

Wir könnten jetzt auch z.B. eine Komponente <Uhr> erstellen mit

d = new Date();

d.toDateString(); resp. d.toTimeString(); und diese entsprechend mit setInverval updaten.

4.5 Komponenten Wiederverwenden

Die Stärke des Komponentenbasierten Ansatzes ist, dass wir Komponenten wiederverwenden können. Erstellen wir doch den Währungsumrechner aus Kapitel 3.3.3 mit React:
Dazu erstellen wir das File “src/Converter.js”

```
import React, {Component} from "react";

class Converter extends Component {
    constructor(props) {
      super(props);
      this.state = {source: 0, dest: 0};
      this.rate = 1.05;

      // Event-Handler registrieren
      this.handleChange = this.handleChange.bind(this);
      this.handleClick = this.handleClick.bind(this);
    }

    handleChange(event) {
        this.setState({source: event.target.value});
    }

    handleClick(event) {
      this.setState({dest: this.state.source * this.rate});
    }
    render() {
        return(
        <>
        <h2>Converter</h2>
        <form action="#">
            <div>
                CHF: <input id="src" type="number" step="any"
                           onChange={this.handleChange} />
            </div>
            <div>
                EUR: <input id="dest" type="number"
                     value={this.state.dest} step="any" readOnly />
            </div>
            <div>
                <button onClick={this.handleClick}>Convert</button>
            </div>
        </form>
        </>
        );
    }
  }
export default Converter;
```

Nun können wir gewisse Variablen ändern und direkt über die props konfigurieren und danach z.B. den Converter folgendermassen verwenden:

```
<Converter title="Meter zu Meilen" source="Kilometer" dest="Meilen"
rate="0.621371"/>

<Converter title="Franken zu Euro" source="CHF" dest="EUR"
rate="1.04"/>
```

Die Attribute werden ganz einfach über die props geholt:

```
constructor(props) {

super(props);

this.titel = props.title;
this.sourcename = props.source;
this.destname = props.source;
this.rate = props.rate;

}
```

4.6 Deployment

Haben wir eine zufriedenstellende Seite mit React.js erstellt, so können wir mit dem Deployment beginnen.

Zunächst müssen wir in package.json die homepage definieren. Da wir lokal und mit relativem Pfad arbeiten wollen, definieren wir die Homepage auch relativ. Wir könnten hier den kompletten Servernamen etc. definieren, aber das ist moistens nicht nötig.

```
{
  "name": "project",
  "version": "0.1.0",
  "homepage": "./",
...
```

Dann können wir im Terminal “npm run build” aufrufen und im Ordner “build” wird nun eine lauffähige Version erstellt. Diese kann bei Bedarf auf einen Server kopiert werden.