

Clientseitige Webframeworks wie Angular, ReactJS und OpenUI5

Seminararbeit

für die Prüfung zum

Bachelor of Science (B.Sc.)

des Studiengangs Wirtschaftsinformatik

an der Dualen Hochschule Baden-Württemberg Karlsruhe

Verfasser

Sebastian Greulich, Fabio Krämer

Kurs

WWI16B2

Wissenschaftlicher Betreuer

Prof. Dr. Ratz, Dietmar

Prof. Dr. Pohl, Philipp

Schulmeister-Zimolong, Dennis

Abgabe

07.01.2019

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich meine Seminararbeit mit dem Thema: „*Clientseitige Webframeworks wie Angular, ReactJS und OpenUI5*“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Karlsruhe, 07.01.2019

Sebastian Greulich, Fabio Krämer

Inhaltsverzeichnis

Abkürzungsverzeichnis	IV
Abbildungsverzeichnis	V
Listings	VI
1. Einleitung	1
1.1. Projektumfeld	1
1.2. Aufbau	1
2. Grundlagen	2
2.1. JavaScript	2
2.1.1. Der Sprachstandard ECMAScript	2
2.1.2. Die Obermenge TypeScript	4
2.1.3. Die Erweiterung JSX	4
2.2. Zur Entwicklung benötigte Tools	5
2.3. Webframeworks	6
2.4. Vor- und Nachteile von Frameworks	6
2.5. Black-Box- und White-Box-Frameworks	8
2.6. Webarchitekturen	8
2.6.1. MVC-Modell	8
2.6.2. Single-Page-Applications	9
2.6.3. Websockets	9
3. Angular	11
3.1. Allgemein	11
3.1.1. Einführung in das Framework	11
3.1.2. Vorbereitung der Entwicklungsumgebung	11
3.2. Grundkonzepte	13
3.2.1. Angular-Module	13
3.2.2. Komponenten	14
3.2.3. Templates	15
3.2.4. Services	17
3.3. Verwendung	18
4. ReactJS	19
4.1. Allgemein	19
4.1.1. Einführung in das Framework	19
4.1.2. Vorbereitung der Entwicklungsumgebung	19
4.2. Konzepte	20
4.2.1. Rendern von Elementen	20
4.2.2. Komponenten	21

4.2.3. Rendern von Komponenten und virtueller DOM-Baum	23
4.3. Verwendung	24
5. OpenUI5	25
5.1. Allgemeines	25
5.2. Architektur	25
5.3. Verwendung	27
6. Fazit	29
A. Anhang	30
Literatur	31

Abkürzungsverzeichnis

Abbildungsverzeichnis

1.	Beziehung zwischen ECMAScript und TypeScript	3
2.	Umsetzung des MVC-Musters in Angular	11
3.	Screenshot Angular-Beispielanwendung	13
4.	Lebenszyklus einer React-Komponente	23

Listings

2.1. Eine Klasse Person wird von einem Modul bereitgestellt	3
2.2. Beispiel für die Verwendung von JSX	5
3.1. Das Root-Module in der Datei app.module.ts	14
3.2. Die Komponente AppComponent in der Datei app.component.ts . . .	15
3.3. Die Komponente HelloComponent in der Datei hello.component.ts . .	15
3.4. Das Template in der Datei app.component.html	17
4.1. Beispiel React-Anwendung: HTML-Datei	20
4.2. Beispiel React-Anwendung: Aufruf der Render-Methode	21
4.3. Beispiel React-Anwendung: Die Komponente Hello	21
5.1. Beispiel für das Einbinden von OpenUI5	25

1. Einleitung

1.1. Projektumfeld

Im Zuge des Projektes „Stochastische Unternehmensbewertung mittels Methoden der Zeitreihenanalyse“ soll die Software „business horizon“ weiterentwickelt werden. Mit dieser Software soll es möglich werden, auf Basis von betriebswirtschaftlichen Daten den zukünftigen Unternehmenswert zu bewerten. In „business horizon“ wird momentan das Webfrontendframework AngularJS verwendet. Diese Seminararbeit soll in einem Vergleich mit ReactJS und OpenUI5 bewerten, ob AngularJS aus verschiedenen Gesichtspunkten die beste Oberflächentechnologie für das Projekt darstellt.

1.2. Aufbau

Im darauffolgenden Kapitel 2 „Webframeworks“ werden die technischen Grundlagen zum besseren Verständnis der zu vergleichenden Technologien gelegt. Hierbei wird beispielsweise auf die verwendeten Programmiersprachen JavaScript und TypeScript, aber auch auf verschiedene Möglichkeiten zur Quellcodestrukturierung eingegangen.

Im weiteren Verlauf der vorliegenden Seminararbeit werden die einzelnen client-seitigen Webframeworks AngularJS, ReactJS und OpenUI5 beschrieben. In den Beschreibungen wird auf die allgemeinen Rahmenbedingungen der einzelnen Frameworks eingegangen. Diese bilden beispielsweise die verwendete Lizenz oder auch die Browserkompatibilität. Darüber hinaus wird in jeder Beschreibung die Architektur des Frameworks beleuchtet. Hierzu gehören sowohl die notwendigen Dateikomponenten, als auch das Definieren und Reagieren auf Events. Zum Abschluss jeder Beschreibung wird auf die möglichen Einsatzgebiete der Technologien eingegangen.

Nach den detaillierten Beschreibungen der Webframeworks wird in Kapitel „xy“ erörtert, welche Technologie sich am besten die Software „business horizon“ eignet.

2. Grundlagen

2.1. JavaScript

2.1.1. Der Sprachstandard ECMAScript

ECMAScript spezifiziert den Sprachstandard von JavaScript. Dieser wird seit dem Jahr 1997 Jahren von der European Computer Manufactures Association (kurz: ECMA) weiterentwickelt. Zunächst wurden die Versionen durchnummeriert (ES1, ES2, ES3, ES4, und ES5). Im Jahre 2015 wurde beschlossen, dass jährlich eine neue Version von ECMAScript erscheinen soll. Daher tragen die nachfolgenden Versionen das Veröffentlichungsjahr im Namen (ECMAScript2015, ECMAScript2016, ECMAScript2017, ECMAScript2018, ...).

Die neusten Browser unterstützen meist den aktuellsten ECMAScript Sprachstandard. Allerdings verwendet nicht jeder Benutzer einen neuen Browser. Um die Kompatibilität einer Webanwendung zu gewährleisten, muss der Code in eine von den meisten Browsern unterstützte Version transpiliert werden.(vgl. Woiwode et al. [2018](#), S. 27 ff.; vgl. Terlson [2018](#); vgl. Steyer und Schwab [2017](#), S. 13 ff.)

Im Folgenden wird auf die, für die in [Kapitel 3,4](#) und [5](#) vorgestellten Frameworks, relevantesten Sprachfeatures eingegangen.

In ECMAScript2015 wurden die Variablentypen `let` zur Eingrenzung des Geltungsbereichs einer Variable und `const` zur Deklaration einer Konstanten eingeführt. Zudem können seit ECMAScript2015 auch Klassen und Module in JavaScript definiert werden. Eine Klasse kann mehrere Eigenschaften und Methoden enthalten. Zudem können Klassen voneinander erben.

Jede Datei ist ein eigenes Modul. Module fassen zusammengehörige Codeeinheiten zusammen und können Interfaces, Klassen oder Variablen bereitstellen, die wiederum von anderen Modulen verwendet werden können.(vgl. Woiwode et al. [2018](#), S. 34 f.; vgl. Steyer und Schwab [2017](#), S. 19 ff.)

Das Modul in [Listing 2.1](#) stellt eine Klasse `Person` zur Verfügung. Diese Klasse hat einen Konstruktor und eine Instanzmethode `altere`, die die Person um ein weiteres Jahr altern lässt.

```
1 export class Person{
2     private name    : string;
3     private alter   : number;
4
5     constructor(name: string, alter: number){
6         this.name = name;
7         this.alter = alter;
8     }
9
10    altere(): number{
11        alter = alter + 1;
12        return alter;
13    }
14 }
```

Listing 2.1: Eine Klasse Person wird von einem Modul bereitgestellt

Seit ECMAScript2017 können Dekoratoren für die Angabe von Metainformationen zu einer Klasse verwendet werden. Dies wird beispielsweise von dem in [Kapitel 3](#) vorgestellten Framwework Angular zur Kennzeichnung und Konfiguration der unterschiedlichen Bestandteile verwendet.(vgl. Woiwode et al. [2018](#), S. 30 ff.)

Object Spread Operator

Grafik ersetzen!

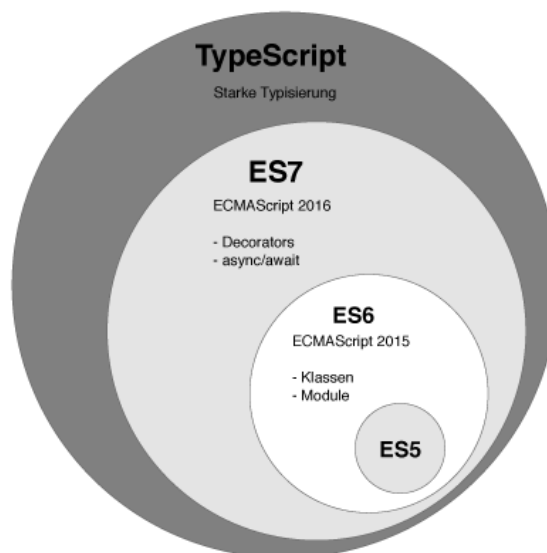


Abbildung 1.: Beziehung zwischen ECMAScript und TypeScript

Quelle: Woiwode et al. ([2018](#), S. 28)

2.1.2. Die Obermenge TypeScript

TypeScript ist eine von Anders Hejlsberg bei Microsoft entwickelte Sprache. Diese erweitert die bestehende ECMAScript Version um weitere Sprachelemente und bildet somit eine Obermenge von JavaScript (siehe [Abbildung 1](#)). Ein Transpilierer übersetzt TypeScript in JavaScript.

TypeScript ergänzt JavaScript unter anderem um ein stärkeres Typsystem. Hierdurch können Typfehler bereits zur Compilezeit erkannt und Tools zur Codeanalyse (automatische Codevervollständigung, Refactoring-Unterstützung, ...) eingesetzt werden. (vgl. Woiwode et al. [2018](#), S. 27 ff.; vgl. Steyer und Schwab [2017](#), S. 13 ff.; vgl. Zeigermann und Hartmann [2016](#), S. 10) Folgende Basistypen stellt TypeScript zur Verfügung: (vgl. Woiwode et al. [2018](#), S. 34 ff.; vgl. Steyer und Schwab [2017](#), S. 16 ff.)

- **number**: Ganz- oder Kommazahl
- **string**: Zeichenkette
- **boolean**: Wahrheitswert
- **Array<typ>**: typisierte Arrays
- **any**: simuliert Standardverhalten von JavaScript – beliebiger Datentyp
- **Function**: Funktion

TypeScript ermöglicht die Verwendung von Interfaces. (vgl. Woiwode et al. [2018](#), S. 40 f.)

2.1.3. Die Erweiterung JSX

Die Spracherweiterung JSX ermöglicht die Verwendung einer HTML ähnlichen Syntax in JavaScript. JSX wird durch einen geeigneten Transpilierer in gültiges JavaScript übersetzt. Zeigermann und Hartmann (vgl. [2016](#), S. 10) empfiehlt an dieser Stelle die Verwendung des Compilers *Babel*.

Die Spracherweiterung kann zum Beispiel bei der Entwicklung mit dem Framework React eingesetzt werden. Dieses Framework verwendet zur Darstellung der Benutzeroberfläche keine Templates. Die Benutzeroberfläche wird stattdessen mit JavaScript Befehlen aufgebaut. Um dies zu erleichtern, kann JSX verwendet werden. Im Folgenden werden die Grundkonzepte von JSX näher erläutert.

Die Ausgabe von dynamische Informationen erfolgt unter Verwendung von JavaScript-Ausdrücken. Diese Ausdrücke müssen in ein paar geschweifter Klammern gepackt werden. Bedingungen können mithilfe des ternären Operatoren überprüft werden. Zur Ausgabe von Listen mit wiederholenden Elementen steht die für Arrays definierte JavaScript Methode `map` zur Verfügung.

JSX bietet zudem die Möglichkeit das Aussehen eines Elements über das `style`-Attribut zu verändern. Das `style`-Attribut ist ein Objekt, daher wird die CSS-Eigenschaft und der zugehörige Wert als Objekt-Literal übergeben. Zudem ist zu beachten, dass die CamelCase-Notation für die CSS-Eigenschaften verwendet werden muss. In Listing 2.2 wird die CSS-Eigenschaft `color` und `font-size` des HTML-Elements gesetzt.

Um XSS-Attacken zu verhindern, werden Strings in JSX automatisch escaped bzw. maskiert. Ein String wird demnach immer als Text interpretiert und dargestellt. (vgl. Zeigermann und Hartmann 2016, S. 59 ff.; vgl. Stefanov 2017, S. 65 ff.)

```
1 const titleColor = 'red';
2 const titleFontSize = 12;
3 const helloWorld = <h1 style = {{
4   color: titleColor,
5   fontSize: titleFontSize + 'px'
6 }}> Hello, World</h1>
```

Listing 2.2: Beispiel für die Verwendung von JSX

NodeJS

2.2. Zur Entwicklung benötigte Tools

Für die Entwicklung einer auf JavaScript basierenden Webanwendung wird unter anderem NodeJS, der Paketmanager npm und ein Modul-Loader benötigt. Demnach werden diese auch bei Verwendung der Frameworks Angular und ReactJS eingesetzt. Im Folgenden werden die Tools näher erläutert.

Die JavaScript Laufzeitumgebung *NodeJS* ermöglicht unter anderem das Ausführen von JavaScript Code auf dem Server. Einige Tools, die bei der Entwicklung einer Webanwendung eingesetzt werden, verwenden NodeJS. Außerdem bietet NodeJS Pakete mit wiederverwendbarem Code, die über den in integrierten Paketmanager *npm* installiert werden können. Die aktuellste Version kann von <https://nodejs.org>

[org/](#) heruntergeladen und installiert werden. (vgl. Steyer und Schwab 2017, S. 1 ff.; vgl. Freeman 2018, S. 7 ff.; vgl. Woiwode et al. 2018, S. 6 ff.)

Wenn Java-Script Module in einer Webanwendung verwendet werden, wird ein Modul-Loader gebraucht. Dieser löst verschiedene Abhängigkeiten auf und erstellt daraus eine Ausgabedatei. Die Verwendung des Modul-Loader *Webpack* wird sowohl für Angular als auch für React Anwendungen empfohlen. *Webpack* unterstützt zudem Hot Module Replacement. Änderungen am Code werden zur Laufzeit übernommen. Dadurch muss eine Anwendung nach einer Änderung nicht komplett neu gebaut werden. (vgl. Woiwode et al. 2018, S. 13, 21; vgl. Zeigermann und Hartmann 2016, S. 9, 295–301; vgl. Hlushko et al. 2018)

2.3. Webframeworks

Webframeworks bilden heutzutage oft die Rahmenstruktur für Oberflächenentwicklungen. Sie sind eine Sammlung von Methoden, welche Funktionalitäten durch einen Methodenaufruf ermöglichen und somit die Struktur der Oberflächen bestimmen. Webframeworks haben sich aufgrund ihrer einfachen Implementierung etabliert. Mit dem Einsatz von Webframeworks muss der Entwickler nicht Experte in jedem Teilgebiet seiner Entwicklung sein, sondern kann schon vorgefertigte und getestete Elemente übernehmen und die spezielle Funktionalität nutzen. Beispielsweise nimmt das Weboberflächenframework Bootstrap dem Entwickler ab, sich um die Implementierung responsiven Fähigkeiten einer Webanwendung zu kümmern. (best practice SE)

Ein Webframework nutzt hierbei ein Grundprinzip der Informatik: die Wiederverwendung. Bei der Wiederverwendung wird vermieden redundanten Quellcode zu schreiben. Dies bietet zahlreiche Vorteile. Zum einen muss sich der Entwickler nicht mehrmals die gleichen Gedanken zur Problemlösung machen, zum anderen muss der Entwickler bei einem gefundenen Fehler nur an einer zentralen Stelle und nicht bei jeder Verwendung nachbessern. (techn Informatik s42)

2.4. Vor- und Nachteile von Frameworks

Zum Abschluss des Kapitels „Webframeworks“ werden einige Vor- und Nachteile bei der Verwendung dieser kritisch betrachtet.

Zuerst werden auf die Nachteile bei der Verwendung von Webframeworks eingegangen:

2. Grundlagen

Jedes Framework hat einen hohen Grad an **Komplexität**. Dieser ist notwendig, um verschiedensten, teils komplexen Anforderungen gerecht zu werden. Doch auch der Anwender muss die Funktionsweise des Frameworks zumindest teilweise verstanden haben um es effizient in seiner Entwicklung einsetzen zu können.

Bei der Nutzung eines Frameworks ergeben sich zwangsläufig **Anhängigkeiten** von diesem. Dies führt dazu, dass der Entwickler sich an eine bestimmte vorgegebene Architektur halten muss. Darüber hinaus muss der Frameworknutzer darauf vertrauen, dass der Frameworkentwickler seine Software ausreichend testet bevor er es an die Nutzer verteilt. Sonst ist die Funktionalität nicht gewährleistet.

Viele Frameworks (oft im Open-Source-Umfeld) leiden oft an **mangelnder Dokumentation**. Dies macht es dem Entwickler schwer, sich in das Framework einzuarbeiten und es von Beginn an effizient zu nutzen.

Sowohl die Komplexität als auch die Abhängigkeit können auf der einen Seite als Nachteil gesehen werden. Jedoch auf der anderen Seite kann durch das umfangreiche Funktionsangebot sehr viel Zeit und Aufwand gespart werden.

Ein Framework wird normalerweise im objektorientierten Umfeld genutzt. Somit gibt es dem Entwickler schon ein bestimmtes **Architekturmuster** vor und er ist gezwungen seine Webanwendung modular aufzubauen.

Aufgrund der Generik eines Webframeworks ist es möglich, dieses für viele verschiedene Anwendungsfälle **wiederverwenden**. Damit spart sich der Entwickler wertvolle Zeit und Aufwand.

Webframeworks sind auf **Erweiterbarkeit** ausgelegt. Man kann den Quellcode jedes Frameworks untersuchen sowie beliebig verändern und erweitern. Dies bietet den Vorteil, wenn man als Entwickler einen anderen Datentyp benötigt oder in eine Funktion einen weiteren Parameter hinzufügen möchte kann man einfach das Framework an die speziellen Bedürfnisse anpassen.

Durch **Inversion of Control** bleibt der Kontrollfluss auf Seiten des Frameworks. Somit wird die Steuerung zur Ausführung bestimmter Unterprogramme an das Framework abgegeben und der Entwickler kann sich voll und ganz auf die Implementierung seiner Geschäftslogik konzentrieren.

Zuletzt ist der Vorteil durch **Standardisierung** anzuführen. Bestimmte Frameworks haben sich in ihrem Umfeld zum Standard entwickelt. Ein gutes Beispiel hierfür ist das PHP-Framework „Medoo“, es bildet den Open-Source-Standard bei SQL-Datenbankzugriffen über PHP. Die Framework Standardisierung verkürzt die

Einarbeitungszeit von Entwicklern in diesem Umfeld, da die aufkommenden Probleme immer auf dieselbe Weise gelöst werden können.

2.5. Black-Box- und White-Box-Frameworks

Bei Frameworks unterscheidet man zwischen Black-Box- und White-Box-Frameworks. Bei einem Black-Box-Frameworks ist der Entwickler gezwungen sich auf vorgefertigte Methoden zu verlassen. Er kann das Framework nur einbinden aber keine logischen Veränderungen daran vornehmen. Bei einem **White-Box-Framework** stehen im Gegensatz dazu oftmals nur abstrakte oder leere Methoden zur Verfügung. Diese Methodenhüllen muss der Entwickler nun in seiner Software mit Inhalt füllen. Bei einem Vergleich dieser beiden Frameworkarten lässt sich feststellen, dass der Entwickler bei einem **White-Box-Framework** dieses im Ganzen verstanden haben muss um die Lücken mit seinem speziellen anwendungsfallspezifischen Code zu füllen. Wohingegen das Black-Box-Framework eine Standardlösung darstellt, welche sich nicht näher auf den speziellen Anwendungsfall anpassen lässt, aber dafür nur eine geringe Einarbeitungszeit seitens des Entwicklers fordert.

2.6. Webarchitekturen

2.6.1. MVC-Modell

Das Entwurfsmuster MVC stammt ursprünglich aus den 1970er Jahren. Es wurde ursprünglich für Desktopanwendungen entwickelt, wird nun aber auch sowohl im Web, als auch bei mobilen Anwendungen verwendet. Die Idee hinter dem MVC-Konzept ist die Trennung von Bestandteilen einer Anwendung in jeweils eigene Verantwortlichkeiten. Diese Verantwortlichkeiten werden im Vorhinein in der Definition geregelt. Somit erreicht man voneinander unabhängige Komponenten, welche eigenständig und austauschbar sind. Mit dem MVC-Konzept vermeidet man darüber hinaus eine Vermischung von Aufgaben und der damit verbundenen Logik.

Der MVC-Ansatz umfasst 3 Komponenten. Die Erste davon ist das Model. Es steht für das zu präsentierende Datenmodell einer Anwendung. Die Zweite Komponente stellt die View dar. Sie ist für die Präsentation der im Model enthaltenen Daten zuständig. In den Verantwortungsbereich der View fällt außerdem das Anzeigen von Benutzerinteraktionsmöglichkeiten wie beispielsweise ein Button. Die letzte Komponente bildet der Controller. Er bildet die Steuerungseinheit der Anwendung

und kümmert sich um die Logik einer Applikation. Zu seinen Aufgaben gehört die Vermittlung zwischen dem Model und der View sowie der Interaktion mit dem Benutzer. Der Controller muss sicherstellen, dass alle Anpassungen in der View konsistent auf das Datenmodell angewandt werden.

2.6.2. Single-Page-Applications

Bei dem zuvor beschriebenen MVC-Modell dient der Browser lediglich als einfaches Anzeigeprogramm. Bei jeder Interaktion wird eine Nachricht an den Server geschickt und es wird eine neue Webseite als Antwort zurückgesendet. Früher war dies aufgrund von langsamen Browsern die einzige Möglichkeit, auf Benutzereingaben zu reagieren. Im Laufe der Zeit wurden die Webbrowser immer performanter und damit wurde es komfortabel möglich, mittels JavaScript die Webseite ohne den Server zu verändern.

Um das Jahr 2005 fing die Technologie AJAX (Asynchronous JavaScript and XML) an, sich als Webkommunikationsstandard zu etablieren. Hierbei war die Technik, die AJAX zu Grunde liegt nicht neu, denn es ist lediglich eine Kombination bestehender Technologien mit einer XMLHttpRequest. Durch AJAX wurde es somit möglich, asynchrone Anfragen an den Webserver zu senden und die erhaltenen Daten in die bereits geöffnete Webseite zu integrieren. Nun war die Single-Page-Application geboren, denn das Grundgerüst der Webseite bei dieser Technik nur beim initialen Aufruf geladen werden. Heutzutage nutzen fast alle Webframeworks die AJAX-Technik um die Ladezeiten der Webseiten zu verkürzen, falls nur eine kleine Änderung vorliegt. Ein weiterer Vorteil ergibt sich darüber hinaus durch die Asynchronität, denn falls eine schlechte Internetverbindung vorliegt zeigt der Browser während der Ladezeit nicht eine weiße Ladeseite sondern der Benutzer kann die Webinhalte weiter konsumieren.

2.6.3. Websockets

Mit den bisher beschriebenen Technologien musste der Client immer eine Anfrage an den Server schicken, doch bei bestimmten Anwendungsfällen ist eine persistente und bidirektionale Verbindung zum Server notwendig. Beispielsweise will man bei Chat-Anwendungen oder Live-Sport-Ticker immer in Echtzeit die aktuellen Informationen bereitgestellt bekommen. Doch die bisher gängigen HTTP-Verbindungen sehen diese Art der Verbindung nicht vor. Auf ihrer Basis müsste der Benutzer die Webseite immer manuell aktualisieren. Mittels eines Websockets kann eine solche persistente Client-Server-Verbindung geöffnet werden. Ein WebSocket basiert auf einem eigenen

2. Grundlagen

Protokoll, dieses muss zuerst über HTTP-Anfragen verhandelt werden. Wenn Client und Server der Verbindung zugestimmt haben, besteht zwischen den beiden Parteien ein dauerhafter „Tunnel“, über diesen können Informationen mit geringer Latenz ausgetauscht werden. Im Vergleich zu einer herkömmlichen HTTP-Verbindung wird der überflüssige Header eingespart, das führt zu weniger Internettraffic. Wenn eine der beiden Parteien die dauerhafte Verbindung trennen möchte, muss sie der Anderen eine „Close“-Nachricht senden. (SPA-Dev)

3. Angular

3.1. Allgemein

3.1.1. Einführung in das Framework

Angular ist ein von Google verwaltetes Open Source Framework, mit dem unter anderem clientseitige Single-Page-Applikationen entwickelt werden können. Die erste Version des Angular-Frameworks hat den Namen AngularJS. Alle nachfolgenden Versionen tragen den Namen Angular, da es zwischen der ersten und zweiten Version des Frameworks grundlegende Änderungen gab.

Das Framework wird kontinuierlich weiterentwickelt. Monatlich soll eine Minor-Version und alle sechs Monate eine Major-Version erscheinen. Die aktuellste stabilste Version von Angular hat die Versionsnummer 7.1.4. (vgl. Woiwode et al. 2018, S. vii ff.; vgl. Freeman 2018, S. 3 ff.)

Angular ist in der Sprache TypeScript geschrieben. Angular kann mit TypeScript, JavaScript oder Dart genutzt werden. (vgl. Woiwode et al. 2018, S. vii f.; vgl. Steyer und Schwab 2017, S. 13)

Das in Kapitel 2 beschriebene MVC Entwurfsmuster kann in einer Angular-Anwendung umgesetzt werden. Die Abbildung 2 zeigt die einzelnen Bestandteile der Anwendung. Dabei implementiert das Template die View, die Komponente den Controller und das Modell das Model. (vgl. Freeman 2018, S. 34 ff.) Im weiteren Verlauf werden die Bestandteile näher beschrieben.

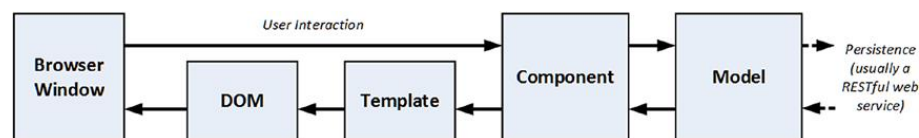


Abbildung 2.: Umsetzung des MVC-Musters in Angular

Quelle: Freeman (2018, S. 35)

3.1.2. Vorbereitung der Entwicklungsumgebung

Für die Entwicklung einer Angular-Anwendung wird unter anderem NodeJS (siehe Abschnitt 2.2), AngularCLI und eine geeignete Entwicklungsumgebung benötigt.

3. Angular

Das Angular Commandline Interface (kurz: AngularCLI) unterstützt den Entwickler beim Erzeugen und Verwalten einer Angular-Anwendung. AngularCLI erzeugt unter anderem das Grundgerüst einer Angular-Anwendung und richtet den TypeScript Compiler, Werkzeuge zur Testautomatisierung und den Build-Prozess ein. Die aktuelle Version von AngularCLI kann über den Paketmanager *npm* installiert werden. (vgl. Steyer und Schwab 2017, S. 1 ff.; vgl. Freeman 2018, S. 7 ff.; vgl. Woiwode et al. 2018, S. 6 ff.)

Sowohl Woiwode et al. (vgl. 2018, S. 3 ff.) als auch Steyer und Schwab (vgl. 2017, S. 3 ff.) empfehlen die Verwendung der frei verfügbaren Entwicklungsumgebung Visual Studio Code. Die Entwicklungsumgebung unterstützt die Entwicklung in TypeScript und lässt sich leicht durch Plug-Ins, die die Entwicklung erleichtern sollen, erweitern. Visual Studio Code ist für Linux, Mac und Windows verfügbar und kann unter <https://code.visualstudio.com/Download> heruntergeladen werden.

Struktur einer durch AngularCLI erzeugte Angular-Anwendung:

Evtl. an dieser Stelle den Source Code Ordner weg lassen und näher auf die anderen Dateien eingehen.

```
├ example/
│   ├── src/
│   │   ├── app/
│   │   │   ├── -app.component.css ⇒ CSS-Datei von AppComponent
│   │   │   ├── -app.component.html ⇒ Template von AppComponent
│   │   │   ├── -app.component.ts ⇒ Komponente AppComponent
│   │   │   ├── -app.module.ts ⇒ Root-Modul AppModule
│   │   ├── assets/
│   │   ├── environments/
│   │   ├── -index.html
│   │   ├── -main.ts
│   │   ├── -styles.css
│   │   ├── -...
│   │   ├── node_modules/
│   │   ├── e2e/
│   │   └ -...
```

Hello World!

Name:

Abbildung 3.: Screenshot Angular-Beispielanwendung

3.2. Grundkonzepte

Im Folgenden werden die in Angular verwendeten Konzepte näher erläutert. Die Beispielanwendung in [Abbildung 3](#) wird zur Erklärung der Konzepte verwendet. Eine Änderung des Namens im Input-Feld bewirkt eine Veränderung des Namens in der Überschrift.

3.2.1. Angular-Module

Eine Angular-Anwendung kann aus mehreren Angular-Modulen bestehen und ist demnach modular aufgebaut. Angular-Module sind nicht mit den in [Unterabschnitt 2.1.1](#) vorgestellten JavaScript-Modulen zu verwechseln. (vgl. Steyer und Schwab 2017, S. 103 ff.; vgl. Woiwode et al. 2018, S. 301 ff.) Die Module einer Angular-Anwendung können in Root-Module, Feature-Module und Shared-Module unterteilt werden. Wenn ein Client eine auf Angular basierte Seite anfordert, dann schickt der Server den Inhalt der *index.html* als Antwort an den Client zurück. Der Client führt daraufhin die im HTML-Dokument enthaltenen Skript-Elemente aus. Dabei wird die Angular-Plattform initialisiert und das Root-Modul übergeben. Dieses Root-Modul konfiguriert die Angular-Anwendung. (vgl. Steyer und Schwab 2017, S. 60; vgl. Freeman 2018, S. 226 ff.) Das Root-Modul der Beispielanwendung ist das Modul *AppModule* siehe [Listing 3.1](#).

Module werden im Allgemeinen durch den Dekorator *@NgModule* gekennzeichnet. Ein Modul kann verschiedene weitere Module über die Eigenschaft *import* importieren und damit die bereitgestellten Funktionalitäten verwenden. Die vom Modul verwendeten Direktiven, Komponenten und Pipes werden in der Eigenschaft *declarations* angegeben. Jedes Root-Modul muss die Eigenschaft *bootstrap* führen. Diese Eigenschaft spezifiziert die Komponente, die beim Starten der Anwendung geladen werden soll.

Das Modul *AppModule* in [Listing 3.1](#) importiert die Module *NgModule*, *BrowserModule* und *FormsModule* und deklariert die zugehörigen Komponenten *AppComponent* und

3. Angular

HelloComponent. Beim Starten der Anwendung soll die Komponente *AppComponent* aufgerufen werden.

```
1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3 import { FormsModule } from '@angular/forms';
4
5 import { AppComponent } from './app.component';
6 import { HelloComponent } from './hello.component';
7
8 @NgModule({
9   imports:      [ BrowserModule, FormsModule ],
10  declarations: [ AppComponent, HelloComponent ],
11  bootstrap:    [ AppComponent]
12 })
13 export class AppModule { }
```

Listing 3.1: Das Root-Module in der Datei app.module.ts

Feature Module ermöglichen die Gruppierung einer Anwendung in Anwendungsfällen. Mithilfe von Shared-Module können die Teile einer Anwendung zusammengefasst werden, die unabhängig vom Anwendungsfall verwendet werden können. (vgl. Freeman 2018, S. 528 ff.; vgl. Google 2018a; vgl. Steyer und Schwab 2017, S. 105 ff.)

3.2.2. Komponenten

Komponenten sind Klassen, die Daten und Logik zur Anzeige in den zugehörigen Templates bereitstellen. Diese ermöglichen die Aufteilung einer Angular-Anwendung in logisch getrennte Teile. (vgl. Freeman 2018, S. 401)

Eine Komponente wird durch den Dekorator *@Component* gekennzeichnet und kann über verschiedene Dekorator-Eigenschaften konfiguriert werden. Die Eigenschaft *selector* identifiziert das HTML-Element, dass durch diese Komponente repräsentiert wird. Zur Anzeige der bereitgestellten Daten kann entweder ein Inline-Template *template* definiert oder auf ein externes Template *templateUrl* verwiesen werden. (vgl. Google 2018b; vgl. Freeman 2018, S. 405; vgl. Steyer und Schwab 2017, S. 47 ff.)

Beschreibung von Input- und Output Eigenschaften.

Die Beispielanwendung enthält die Komponenten *AppComponent* (siehe Listing 3.2) und *HelloComponent* (siehe Listing 3.3).

```
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'my-app',
5   templateUrl: './app.component.html',
6   styleUrls: [ './app.component.css' ]
7 })
8 export class AppComponent {
9   name: string;
10 }
```

Listing 3.2: Die Komponente AppComponent in der Datei app.component.ts

```
1 import { Component, Input } from '@angular/core';
2
3 @Component({
4   selector: 'hello',
5   template: '<h1>Hello {{name}}!</h1>',
6   styles: ['h1 { font-family: Lato; }']
7 })
8 export class HelloComponent {
9   @Input() name: string;
10 }
```

Listing 3.3: Die Komponente HelloComponent in der Datei hello.component.ts

3.2.3. Templates

Zur Darstellung von Komponenten nutzt Angular Templates. Ein Template besteht aus HTML Code, der um Angular spezifische Konzepte wie Direktiven, Datenbindungsausdrücke und Pipes erweitert wird. (vgl. Google [2018b](#); vgl. Steyer und Schwab [2017](#), S. 52)

Mit Direktiven kann einem Element zusätzliches Verhalten hinzugefügt werden. (vgl. Steyer und Schwab [2017](#), S. 265; vgl. Freeman [2018](#), S. 401) In Angular werden folgende drei Arten von Direktiven unterschieden. (vgl. Google [2018d](#))

- Strukturelle Direktiven
- Attribut-Direktiven
- Komponenten

3. Angular

Die strukturellen Direktiven ändern die Struktur des zugehörigen HTML-Elements, indem sie HTML-Elemente hinzufügen oder entfernen. Hierfür verwenden die strukturellen Direktiven Templates, die beliebig oft gerendert werden. (vgl. Steyer und Schwab 2017, S. 269 ff.; vgl. Freeman 2018, S. 365) Beispiele für strukturellen Direktiven aus Angular sind (vgl. Freeman 2018, S. 261 ff.):

ngIf Fügt dem HTML-Dokument Inhalt hinzu, wenn die Bedingung wahr ist.

ngfor Fügt für jedes Item einer Datenquelle den gleichen Inhalt dem HTML-Dokument hinzu.

ngSwitch Fügt dem HTML-Dokument, abhängig vom Wert eines Ausdrucks, Inhalt hinzu.

Das Verhalten und das Aussehen des zugehörigen HTML-Elements kann durch die Attribut-Direktiven verändert werden. Diese Direktiven fügen oder entfernen dem zugehörigen HTML-Element Attribute. (vgl. ebd., S. 339) Beispiele für Attribut-Direktiven aus Angular-JS (vgl. ebd., S. 249 ff.):

ngStyle Mit dieser Direktive können unterschiedliche Style-Eigenschaften dem Element hinzugefügt werden.

ngClass Weißt dem Element ein oder mehrere Klassen hinzu.

Mittels einer Komponente kann einem HTML-Element eine View hinzugefügt werden. Komponenten sind nämlich Direktiven mit einer eigenen View. (vgl. Steyer und Schwab 2017, S. 265)

Die von Angular bereitgestellten Direktiven (engl. Built-In Directives) können durch selbst entwickelte Direktiven erweitert werden. (vgl. Freeman 2018, S. 261)

Der Datenaustausch zwischen der Komponente und dem Template erfolgt durch Datenbindungsausdrücke. Ein Datenbindungsausdruck bindet einen JavaScript-Ausdruck an ein Ziel. Das Ziel kann entweder eine Attribut-Direktive oder eine Eigenschaft des zugehörigen HTML-Elements sein. Der JavaScript-Ausdruck ermöglicht den Zugriff auf die Eigenschaften und Methoden der Komponente. (vgl. Freeman 2018, S. 237 ff.; vgl. Steyer und Schwab 2017, S. 52 f.; vgl. Google 2018c) Es gibt insgesamt drei Arten von Data-Bindings, die anhand der Flussrichtung der Daten unterschieden werden können.

Mit Pipes können die Daten vor der Ausgabe sortiert, formatiert oder gefiltert werden. (vgl. Steyer und Schwab 2017, S. 83 ff.)

Richtung	Syntax	Verwendung
One-way Komponente -> View	{{Ausdruck}} [Ziel]="Ausdruck"	Interpolation, Eigenschaft, Attribut, Klasse, Style
One-way Komponente <- View	(Ziel)="Ausdruck"	Events
Two-way	[(Ziel)]="Ausdruck"	Formular

Tabelle 3.1.: Arten von Data-Bindings

```

1 <hello name="{{name}}"></hello>
2 <label for="name">Name:</label>
3 <input name="name" [(ngModel)]="name">

```

Listing 3.4: Das Template in der Datei app.component.html

3.2.4. Services

Laut Freeman (vgl. 2018, S. 474) kann jedes Objekt, dass durch Dependency Injection verwaltet und verteilt wird, als Service bezeichnet werden. Services stellen wiederverwendbare Routinen oder Daten zur Verfügung, die von Direktiven, Komponenten, weiteren Services und Pipes verwendet werden können. (vgl. Freeman 2018, S. 467 ff.; vgl. Steyer und Schwab 2017, S. 89)

Services verringern die Abhängigkeiten zwischen den Klassen, durch Dependency Injection. Hierdurch können Beispiel Unit-Tests einfacher durchgeführt werden. (vgl. Freeman 2018, S. 469)

Um einen Service zu verwenden, muss dieser entweder global in einem Modul oder lokal in einer Komponente registriert werden. Dies geschieht durch Einrichten eines Providers beim Modul oder der Komponente. Der Provider verknüpft ein Token mit einem Service. Ein Service kann eine Klasse, ein Wert, eine Funktion, eine Factory oder eine Weiterleitung sein. Aus diesem Grund gibt es unterschiedliche Provider.

Sobald ein Service global registrierte wurde, steht dieser auch in weiteren Modulen zur Verfügung. Ein lokal registrierter Service kann dahingegen nur von der jeweiligen Komponente und den direkten und indirekten Kindkomponenten verwendet werden.

Zur Nutzung eines Services muss die jeweilige Klasse den Service importieren und im Konstruktor deklarieren. Beim Erzeugen einer Instanz der Klasse injiziert Angular

3. Angular

den jeweiligen Service.(vgl. Steyer und Schwab [2017](#), S. 92 ff.; vgl. Freeman [2018](#), S. 474 ff.)

3.3. Verwendung

4. ReactJS

4.1. Allgemein

4.1.1. Einführung in das Framework

ReactJS ist eine von Facebook entwickelte JavaScript Bibliothek zur Entwicklung von Benutzeroberflächen. Im Gegensatz zu Angular ist ReactJS ein reines View-Framework. Das Framework wird unter anderem bei Facebook, Instagram, Netflix, Airbnb und dem Content Management System Wordpress eingesetzt. Es bietet einige Vorteile bei der Entwicklung von Anwendungen mit großen Benutzeroberflächen mit Daten, die sich häufig verändern.

Das Framework ist in JavaScript geschrieben und kann mit JavaScript oder einer in JavaScript übersetzbare Sprache wie TypeScript verwendet werden.(vgl. Gackenheimer 2015, S. 1 ff.; vgl. Zeigermann und Hartmann 2016, S. 3 ff.)

[An dieser Stelle eventuell noch mehr auf das Problem eingehen.](#)

4.1.2. Vorbereitung der Entwicklungsumgebung

Für die Entwicklung einer React-Anwendung wird unter anderem *NodeJS* (siehe [Abschnitt 2.2](#)), ein Übersetzer (bspw. *Babel* siehe [Unterabschnitt 2.1.3](#)), ein Modul-Loader (bspw. *Webpack* siehe [Abschnitt 2.2](#)) und die React-Bibliothek benötigt. Die React-Bibliothek bestehendes aus *react* und *react-dom* kann über den Paketmanager *npm* installiert werden. (vgl. Stefanov 2017, S. 92 ff.; vgl. Zeigermann und Hartmann 2016, S. 8 ff.)

Erweiterungen: React Router, Redux (Flux- Implementierung)

React Developer Tools

React

4.2. Konzepte

4.2.1. Rendern von Elementen

Eine React-Anwendung benötigt ein Wurzelement auf einer HTML-Seite. Unter diesem HTML-Element rendert React die Benutzeroberfläche der React-Anwendung. Das Rendern bezeichnet dabei den Vorgang, der schlussendlich zum Einfügen von HTML-Elementen im DOM führt. Dieser Vorgang wird in [Unterabschnitt 4.2.3](#) näher betrachtet.

Die Benutzeroberfläche einer React-Anwendung wird durch React-Elemente repräsentiert. Ein React-Element ist ein JavaScript-Objekt. Es ist unveränderbar und kann nur durch Erzeugen eines neuen Elements aktualisiert werden.

Das [Listing 4.1](#) zeigt die HTML-Seite der Beispielanwendung. Die Benutzeroberfläche der Beispiel React-Anwendung aus [Listing 4.2](#) soll unter dem Element mit der ID *app* gerendert werden. Dies erfolgt durch Aufruf der Methode „ReactDOM.render()“. Diese veranlasst React zum Rendern des im ersten Parameter definierten React-Elements in das im zweiten Parameter angegebenen Wurzelements. Das React-Element wird unter Verwendung der Spracherweiterung JSX (siehe ??) definiert.

```
1 <!DOCTYPE html>
2 <html>
3   ...
4   <body>
5     <div id="app"></div>
6     <script src="src/app.js"></script>
7   </body>
8   ...
9 </html>
```

Listing 4.1: Beispiel React-Anwendung: HTML-Datei

In diesem Beispiel wird der Methode die Komponente App, die ein React-Element zurückliefert, übergeben. Auf die Komponenten wird in [Unterabschnitt 4.2.2](#) näher eingegangen. (vgl. Zeigermann und Hartmann 2016, S. 4 ff., 26 ff.; vgl. Facebook 2018c; vgl. Facebook 2018a; vgl. Stefanov 2017, S. 2 ff.)

```

1 import "./styles.css";
2 import React from "react";
3 import ReactDOM from "react-dom";
4 import Hello from "./hello";
5
6 class App extends React.Component {
7   render() {
8     return (
9       <ErrorBoundary>
10        <Hello name="World" />
11      </ErrorBoundary>
12    );
13  }
14 }
15
16 class ErrorBoundary extends React.Component {
17   ...
18 }
19
20 ReactDOM.render(<App />, document.getElementById("app"));

```

Listing 4.2: Beispiel React-Anwendung: Aufruf der Render-Methode

4.2.2. Komponenten

Komponenten sind das zentrale Element in ReactJS. Sie enthalten sowohl die Logik als auch die zugehörige Anzeige. Eine Komponente kann entweder als Klasse oder als Funktion (engl. functional components) implementiert werden.

Eine Funktion, die eine React-Komponente implementiert, muss genau ein React-Element als Rückgabewert zurückgeben. Die Klasse muss von *React.Component* erben und die Methode *render()*, die ein React-Element zurückgibt, implementieren. Der Klassenname oder der Funktionsname entspricht dem Namen der Komponente. (vgl. Zeigermann und Hartmann 2016, S. 80 ff.) Das [Listing 4.3](#) zeigt eine Komponente Hello, die als Komponentenklasse implementiert ist.

```

1 import "./styles.css";
2 import React from "react";
3 import ReactDOM from "react-dom";
4
5 export default class Hello extends React.Component {
6   constructor(props) {

```

```

7   super(props);
8   this.state = { name: this.props.name };
9   }
10
11  onChange(event) {
12    this.setState({ name: event.target.value });
13  }
14
15  render() {
16    return (<div>
17      <h1>Hello {this.state.name}</h1>
18      <input onChange={e => this.onChange(e)} />
19    </div>);
20  }
21 }

```

Listing 4.3: Beispiel React-Anwendung: Die Komponente Hello

Durch Eigenschaften (engl. Properties) lässt sich das Aussehen und Verhalten einer Komponente von außen beeinflussen. Einer Komponente können Eigenschaften (engl. Properties) in Form eines Objektes übergeben werden. Das Property-Objekt kann von der Komponente nicht verändert werden.

Bei Komponentenfunktionen wird das Objekt der Funktion und bei Komponentenklassen dem Konstruktor der Klasse übergeben. Nach der Weitergabe des Objekts an die Oberklasse `React.Component` stehen die Properties über die Instanzvariable *props* zur Verfügung. (vgl. Zeigermann und Hartmann 2016, S. 24 f., 83–88; vgl. Stefanov 2017, S. 12–17)

Die Komponentenkategorie bietet gegenüber der Komponentenfunktion einige zusätzliche Funktionen. Eine Komponente, die als Klasse implementiert wird, hat einen Zustand (engl. State). Dieser wird in der Instanzvariable *state* gehalten und kann nur von der Komponente gelesen und verändert werden. Die Änderung des States sollte hauptsächlich über die Methode *setState()* erfolgen. Diese Methode erwartet ein Objekt mit Key-Value-Paaren oder eine Callback-Funktion. Durch den Aufruf der Methode wird der State mit dem bisherigen State zusammengeführt. (vgl. Zeigermann und Hartmann 2016, S. 24 f., 89–93; vgl. Stefanov 2017, S. 17 f.)

Die Komponente Hello in Listing 4.3 verwendet einen über die Properties übergebenen Namen als initialen Wert für den im State gehaltenen Namen. Bei Änderungen im Inputfeld wird das Event `OnChange` ausgelöst, die Methode `OnChange` aufgerufen und der vom State gehaltene Name über die Methode `setState` geändert. Der Aufruf der Methode führt zum erneuten Rendern der Anwendung. Der in der Überschrift angezeigte Name wird angepasst.

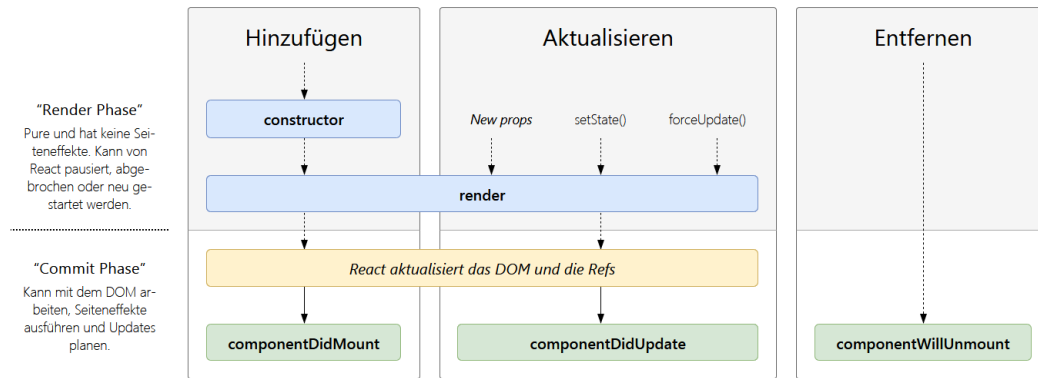


Abbildung 4.: Lebenszyklus einer React-Komponente

Quelle: Maj (2018)

Eine Komponente hat einen gewissen Lebenszyklus. In diesen Zyklus kann bei Verwendung einer Komponentenkasse durch Überschreiben von Lebenszyklus-Methoden (engl. lifecycle-methods) eingegriffen werden. Die [Abbildung 4](#) zeigt die gebräuchlichsten Lebenszyklus-Methoden einer Komponente. (vgl. Zeigermann und Hartmann 2016, S. 96–100; vgl. Facebook 2018b)

In einer Komponente können weitere Komponenten eingebunden werden. Eine Komponente wird dabei durch ihren Namen identifiziert. Das Einbinden erfolgt durch Verwendung der Komponente in der Ausgabe. Dieser können Attribute über die Properties mitgegeben werden. (vgl. Zeigermann und Hartmann 2016, S. 111 ff.) In diesem Beispiel wird die Komponente Hello aus [Listing 4.3](#) und die Komponente ErrorBoundary aus [Listing 4.2](#) in der Komponente App verwendet.

4.2.3. virtueller DOM-Baum

Eine Änderung an den Properties oder am State einer Komponente veranlasst React zum erneuten Rendern. Beim Rendern wird das von einer Komponente bereitgestellte React-Element allerdings nicht direkt in ein DOM-Element umgewandelt und in das native DOM eingesetzt. An stattdessen wird beim Aufruf der Render-Methode eine virtuelle Repräsentation des DOM-Baums generiert.

Anschließend vergleicht React den virtuellen DOM-Baum vor und nach der Änderung. Auf Grundlage der ermittelten Unterschiede generiert React eine minimale Anzahl an DOM-Operationen. Diese Operationen werden dann auf den nativen DOM-Baum angewandt und stellen somit den gewünschten Zustand her. Das Rendern ist ein asynchroner Vorgang. Änderungen werden zum Teil zusammengefasst und nicht sofort angewandt.

React-Elemente sind einfache und leichtgewichtige JavaScript-Objekte. Daher können die Änderungen zwischen den zwei virtuellen DOM-Bäumen schnell ermittelt werden. Durch dieses Verfahren werden teure DOM-Manipulationen reduziert und damit die Performance der Web-Anwendung verbessert.(vgl. Zeigermann und Hartmann [2016](#), S. 23 f., 60 f., 90 f.; vgl. Stefanov [2017](#), S. 53 f.; vgl. Facebook [2018c](#))

4.3. Verwendung

5. OpenUI5

5.1. Allgemeines

Das dritte, im Rahmen dieser Seminararbeit betrachtete clientseitige Webframework heißt OpenUI5. Es ist ein Open-Source-Projekt der SAP SE und wurde im Dezember 2013 unter der Apache-Lizenz 2.0 veröffentlicht. Diese Lizenz erlaubt das freie Verwenden, Verteilen und Modifizieren der Software, durch diese Eigenschaften ist sie mit der GNU General Public License 3 kompatibel. Das Pendant zu OpenUI5 ist SAPUI5, es ist die proprietäre Version von OpenUI5. SAPUI5 hat den gleichen Funktionsumfang wie OpenUI5 und wird von der SAP SE kostenfrei an ihre Kunden vertrieben. Der Vorteil von SAPUI5 ist ein umfangreicher Softwaresupport, welcher für OpenUI5 nicht verfügbar ist. Da man für den Einsatz von SAPUI5 SAP-Kunde sein muss wird dieses Produkt im Rahmen dieser Seminararbeit nicht weiter betrachtet.

Das OpenUI5-Framework basiert auf HTML5, CSS3 und JavaScript. Daher ist es auch mit allen Browsern kompatibel, welche diese Technologien unterstützen. Dazu zählen der Internet Explorer ab Version 9, Firefox ab Version 4 und Google Chrome ab Version 1.

5.2. Architektur

OpenUI5 ist ein, wie in den Technischen Grundlagen beschrieben, clientseitiges JavaScript Black-Box-Framework. Dies bedeutet, man kann das Framework einfach mittels `<script>`-Befehl in ein HTML-Dokument importieren. Hierbei können wichtige Interfaceparameter übergeben werden, wie beispielsweise das zu verwendende Theme. Bei näherem Betrachten fällt auf, dass OpenUI5 auf einer modifizierten bootstrap-Bibliothek basiert.

```
1 <script id="sap-ui-bootstrap"
2   src=https://openui5.hana.ondemand.com/resources/sap-ui-core.js
3   data-sap-ui-theme="sap_belize">
4 </script>
```

Listing 5.1: Beispiel für das Einbinden von OpenUI5

5. OpenUI5

Nachdem die Bibliothek in das Dokument integriert wurde, kann man in einem JavaScript-Teil des HTML-Dokumentes verschiedene UI-Elemente generieren und diese auf der Webseite anzeigen lassen. Dieses Verfahren eignet sich jedoch aufgrund der schlechten Strukturierungsmöglichkeiten nur für sehr kleine und kompakte Webanwendungen mit geringem Funktionsumfang.

Für größere Webentwicklungsprojekte wird auf der OpenUI5-Dokumentationshomepage die Entwicklungsumgebung Eclipse empfohlen. Für diese wurde von der SAP das „SAPUI5 Application Development“-Add-on entwickelt. Dieses ist mit OpenUI5 kompatibel und dient als Assistent zur Erstellung neuer UI5-Entwicklungsprojekte. Der Assistent erstellt nach seinem Durchlauf eine Dateistruktur. Die wichtigsten Dateielemente lauten:

- Index.html
- Component.js
- View.xml
- Controller.js
- i18n.properties

Auf die Funktion und Struktur der Elemente wird im Verlauf dieses Kapitels detailliert eingegangen.

Die index.html-Datei ist der erste Schritt für den Browser beim Öffnen einer Webseite. Bei OpenUI5 werden in der index.html allgemeine Parameter wie beispielsweise Dateipfade oder Kompatibilitätsinformationen festgelegt. Darüber hinaus wird von ihr auf die anderen Bestandteile der Webanwendung verwiesen.

In der Component.js werden alle Datenverbindungen beschrieben. Hierzu gehören beispielsweise einfach eingebundene JSON-Dateien oder auch Verbindungen zu einem ODATA-Webservice, welcher dem UI5-Frontend Daten zur Anzeige bereitstellt. Darüber hinaus wird in der Component.js festgelegt, welche Art der Datenbindung zum Webservice besteht. Möglich ist hierbei nur das bloße Anzeigen, aber auch das Verändern auf der Datenbank.

Die View.xml ist von der Struktur eine XML-Datei. Wie ihr Name schon aussagt, wird in ihr die visuelle Anzeige beschrieben. Die XML-Datei eignet sich in ihrem Aufbau sehr gut, eine grafische Oberfläche zu strukturieren. Denn in ihr kann man verschiedene Anzeigeelemente klar erkennen und voneinander trennen. Dazu gehören zum Beispiel Header, Footer oder einzelne Container mit verschiedenem Inhalt.

Die View in UI5 ermöglicht auch die einfache Datenbindung zur Datenbank ohne zusätzliche Programmierung im Controller.

Die Datei Controller.js enthält die programmspezifische Logik der UI5-Webanwendung. In ihr können mittels JavaScript die gewünschten Funktionen ausprogrammiert werden. Eine typische Funktion stellt beispielsweise, das Öffnen eines Dialogfensters beim Drücken eines zuvor in der View definierten Buttons dar. Somit fügt der Controller der einfachen View Funktionen, welche über einfache Tabellenanzeigen oder -änderungen hinausgehen hinzu. Es ist darüber hinaus auch möglich, im Controller weitere Frameworks zu laden und zu nutzen. Als mögliches Anwendungsbeispiel wäre hierbei das Einbinden eines Barcodescannerframeworks (zum Beispiel quagga.js) um mit einer Handykamera industrielle Barcodes zu lesen.

Das letzte wichtige OpenUI5-Fragment stellt die Datei i18n.properties dar. Sie ist für Webanwendungen im internationalen Umfeld sehr wichtig. Der Entwickler kann mehrere i18n-Dateien mit Übersetzungen in diversen Sprachen erstellen. Das OpenUI5-Framework stellt daraufhin sicher, dass die passende Sprache als Anzeigesprache verwendet wird. Die Syntax der Dateibenennung hierbei lautet i18n_XX.properties, wobei das XX für das Länderkürzel der Sprache steht (z.B. Deutsch: i18n_DE.properties).

5.3. Verwendung

OpenUI5 besitzt eine sehr umfassende und übersichtliche Dokumentation unter www.openui5.hana.ondemand.com/#/api. Diese umfasst darüber hinaus viele Anwendungsbeispiele aus der Praxis. Hierbei ist es möglich, den verwendeten Code zu analysieren und in der eigenen Anwendung zu verwenden. OpenUI5 wurde von der SAP entwickelt um die veralteten SAP ERP-Transaktionen zu ersetzen. In diesem Umfeld hat OpenUI5 auch die größten Stärken. Beispielsweise ist es sehr komfortabel möglich Single-Screen-Geschäftsanwendungen zu erstellen. Diese Anwendungen haben ihren Fokus zumeist auf dem Anzeigen, Pflegen und Auswerten von Daten aus einer Datenbank. Jedoch ist es aufgrund der Verwendung von HTML5 auch möglich, verschiedenste multimediale Inhalte wie beispielsweise Audio oder Video wiederzugeben. Einen weiteren Vorteil von OpenUI5 stellt die Responiveness der Oberfläche dar. Das Framework greift auf die Bibliotheken von bootstrap zu, welche dafür sorgen, dass die Weboberfläche auf jeder Geräteklasse optimal dargestellt wird. Bei der Datenanbindung setzt OpenUI5 auf die OData-Technologie, diese ermöglicht es Daten im JSON-Format asynchron über AJAX zu übertragen. Das auf HTTP basierende OData-Protokoll ist ein von Microsoft entwickelter offener Standard zur

5. *OpenUI5*

Datenübertragung. Der letzte Vorteil von OpenUI5 ist die mögliche Verwendung des MVC-Modells. Dieses vereinfacht die Übersichtlichkeit und Wartbarkeit der Webanwendung.

6. Fazit

A. Anhang

Literatur

- Facebook, Hrsg. (2018a). *React: Add React to a Website*. URL: <https://reactjs.org/docs/add-react-to-a-website.html> (besucht am 29.12.2018).
- Hrsg. (2018b). *React: React.Component*. URL: <https://reactjs.org/docs/react-component.html> (besucht am 28.12.2018).
 - Hrsg. (2018c). *React: Rendering Elements*. URL: <https://reactjs.org/docs/rendering-elements.html> (besucht am 29.12.2018).
- Freeman, Adam (2018). *Pro Angular 6*. 3. Aufl. Berkeley, CA: Apress.
- Gackenhaimer, Cory (2015). *Introduction to React*. The expert's voice in web development. New York, New York: Apress.
- Google, Hrsg. (2018a). *Angular: Attribute Directives*. URL: <https://angular.io/guide/attribute-directives> (besucht am 12.12.2018).
- Hrsg. (2018b). *Angular: Introduction to components*. URL: <https://angular.io/guide/architecture-components> (besucht am 13.12.2018).
 - Hrsg. (2018c). *Angular: Introduction to modules*. URL: <https://angular.io/guide/architecture-modules> (besucht am 15.12.2018).
 - Hrsg. (2018d). *Angular: Template Syntax*. URL: <https://angular.io/guide/template-syntax> (besucht am 17.12.2018).
- Hlushko, Eugene et al. (2018). *Webpack: Why webpack*. URL: <https://webpack.js.org/concepts/why-webpack/> (besucht am 30.12.2018).
- Maj, Wojciech (2018). *React Lifecycle Methods diagram*. URL: <http://projects.wojtekmaaj.pl/react-lifecycle-methods-diagram/> (besucht am 28.12.2018).
- Stefanov, Stoyan (2017). *Durchstarten mit React: Web-Apps einfach und modular entwickeln*. 1. Aufl. Heidelberg: dpunkt.verlag.
- Steyer, Manfred und Daniel Schwab (2017). *Angular: Das Praxisbuch zu Grundlagen und Best Practices*. 2. Aufl. Heidelberg: O'Reilly.
- Terlson, Brian, Hrsg. (2018). *ECMAScript: 2018 Language Specification*. URL: <https://www.ecma-international.org/ecma-262/9.0/> (besucht am 26.12.2018).

- Woiwode, Gregor et al. (2018). *Angular: Grundlagen, fortgeschrittene Techniken und Best Practices mit TypeScript - ab Angular 4, inklusive NativeScript und Redux*. 1. Aufl. ix edition. Heidelberg: dpunkt.verlag.
- Zeigermann, Oliver und Nils Hartmann (2016). *React: Die praktische Einführung in React, React Router und Redux*. 1. Aufl. Heidelberg: dpunkt.