DHBW
Duale Hochschule
Baden-Württemberg

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

Abbildungsverzeichnis

Listings

1. Einleitung

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 unterstütze 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.

```
export class Person{
   private name   : string;
   private alter  : number;

   constructor(name: string, alter: number){
      this.name = name;
      this.alter = alter;
   }

   altere(): number{
      alter = alter + 1;
      return alter;
   }
}
```

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!

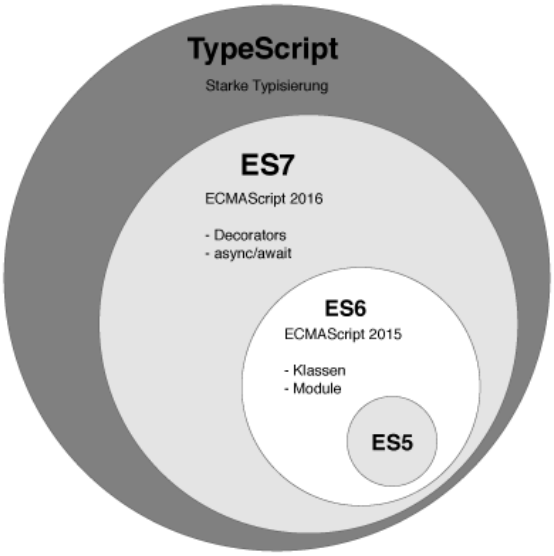

Abbildung 1.: Beziehung zwischen ECMAScript und TypeScript
Quelle: Woiwode et al. (2018, S. 28)

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.

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 sytle-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

3. Angular

3.1. Allgemein

3.1.1. Einführung in das Framework

Angular ist ein von Google verwaltetes Open Source Framework für die Entwicklung von Single-Page-Applikationen. Die erste Version des Angular-Frameworks hat den Namen AngularJS. Alle nachfolgenden Versionen tragen den Namen Angular, da es wischen 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.)

Das Framework selbst 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 **??** beschriebene MVC Entwurfsmuster kann in einer Angular-Anwendung umgesetzt werden. Die **??** 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 von Freeman (ebd., S. 35)

3.1.2. Vorbereitung der Entwicklungsumgebung

Für die Entwicklung einer Angular-Anwendung wird unter anderem NodeJS, AngularCLI, eine Entwicklungsumgebung und ein Browser benötigt.

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.

Die JavaScript Laufzeitumgebung *NodeJS* ermöglicht das Ausführen von JavaScript Code auf dem Server. Einige Tools, die zur Entwicklung einer Angular-Anwendung verwendet werden, verwenden NodeJS. Außerdem bietet NodeJS einige Pakete mit wiederverwendbarem Code, die über den in integrierten Paketmanager *npm* installiert werden können. Die aktuellste Version kann von `https://nodejs.org/` heruntergeladen und installiert werden.

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* siehe Listing 3.1 installiert werden. (vgl. Steyer und Schwab 2017, S. 1 ff.; vgl. Freeman 2018, S. 7 ff.; vgl. Woiwode et al. 2018, S. 6 ff.)

```
1 npm install -g @angular/cli
```

Listing 3.1: Installation von AngularCLI

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/**
- ...

Abbildung 2.: Screenshot Angular-Beispielanwendung

3.2. Grundkonzepte

Wie setzt Angular das MVC Pattern um? Abbildung S.35 Freeman - kurze Beschreibung und dann lange Beschreibung durch Unterkapitel

Im Folgenden werden die in Angular verwendeten Konzepte näher erläutert. Die Beispielanwendung in Abbildung 2 wird zur Erklärung der Konzepte verwendet. Bei Änderung des Namens im Input-Feld wird dieser in der obigen Überschrift auch verändert.

3.2.1. Angular-Module

Eine Angular-Anwendung ist modular aufgebaut und kann demnach aus mehreren Angular-Modulen bestehen. 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.

Den Begriff Bootstrapping erwähnen!

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 3.2.

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 besitzt die Eigenschaft *bootstrap*. Diese Eigenschaft spezifiziert die Komponente, die beim Starten der Anwendung geladen werden soll.

Das Modul *AppModule* in Listing 3.2 importiert die Module *NgModule*, *BrowserModule* und *FormsModule* und deklariert die zugehörigen Komponenten *AppComponent* und *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 3.2: Das Root-Module in der Datei app.module.ts

Feature Modulen 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

Evtl. auf den Lifecycle von Komponenten eingehen.

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 2018c; 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 3.3) und *HelloComponent* (siehe Listing 3.4).

```
import { Component } from '@angular/core';

@Component({
selector: 'my-app',
  templateUrl: './app.component.html',
  styleUrls: [ './app.component.css' ]
})
export class AppComponent  {
  name: string;
}
```

Listing 3.3: Die Komponente AppComponent in der Datei app.component.ts

```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'hello',
  template: `<h1>Hello {{name}}!</h1>`,
  styles: [`h1 { font-family: Lato; }`]
})
export class HelloComponent  {
  @Input() name: string;
}
```

Listing 3.4: 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 2018c; 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

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 2018b) 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 3.5: 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 verwendeten 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

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

Evtl. noch einen kurzen Abschnitt zu Routen

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

4.2. Konzepte

4.2.1. Virtueller DOM

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.

Die Funktion muss genau ein React-Element zurückgeben. Die implementierte Komponente trägt den gleichen Namen wie die Funktion. Die Funktion Hello in Listing 4.1 implementiert die Komponente Hello.

```
import React from 'react';
funtion Hello(){
  return <h1>Hello World</h1>;
}
```

Listing 4.1: Beispiel einer Komponente als Funktion

Eine Klasse, die eine React-Komponente implementiert, muss von *React.Component* erben. Zudem muss die Klasse eine Methode *render()*, die ein React-Element zurückgibt, implementieren. Das Listing 4.2 zeigt die Implementierung der Komponente Hello als Klasse.(vgl. Zeigermann und Hartmann 2016, S. 80 ff.)

```
import React from 'react';
class Hello extends React.Component{
  render() {
    return <h1>Hello World</h1>;
  }
}
```

Listing 4.2: Beispiel einer Komponente als Klasse

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. Eine Veränderung der Properties durch die Komponente ist nicht möglich. 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 Komponentenklasse 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 KeyValue-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.)

Eine Änderung an den Properties oder am State führt zum erneuten Rendern einer Komponente. Dieser Vorgang erfolgt asynchron. Änderungen werden zum Teil

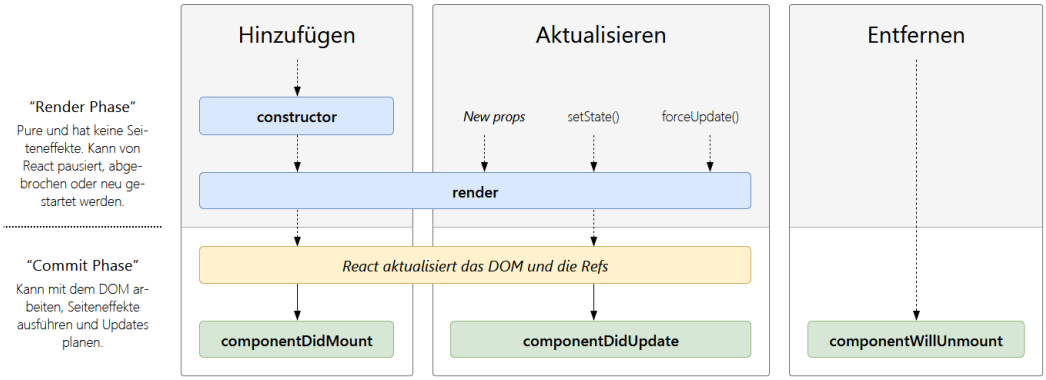

Abbildung 3.: Lebenszyklus einer React-Komponente
Quelle: Maj (2018)

zusammengefasst und nicht sofort angewandt. (vgl. Zeigermann und Hartmann 2016, S. 24 f., 90 f.)

Eine Komponente hat einen gewissen Lebenszyklus. In diesen Zyklus kann bei Verwendung einer Komponentenklasse durch Überschreiben von Lebenszyklus-Methoden (engl. lifecycle-methods) eingegriffen werden. Die Abbildung 3 zeigt die gebräuchlichsten Lebenszyklus-Methoden einer Komponente.(vgl. Zeigermann und Hartmann 2016, S. 96–100; vgl. Facebook 2018)

4.3. Verwendung

5. OpenUI5

6. Fazit

A. Anhang

Literatur

Facebook, Hrsg. (2018). *React.Component*. URL: https://reactjs.org/docs/react-component.html (besucht am 28. 12. 2018).

Freeman, Adam (2018). *Pro Angular 6*. 3. Aufl. Berkeley, CA: Apress.

Gackenheimer, Cory (2015). *Introduction to React*. The expert's voice in web development. New York, New York: Apress.

Google, Hrsg. (2018a). *Attribute Directives*. URL: https://angular.io/guide/attribute-directives (besucht am 12. 12. 2018).

– Hrsg. (2018b). *Introduction to components*. URL: https://angular.io/guide/architecture-components (besucht am 13. 12. 2018).

– Hrsg. (2018c). *Introduction to modules*. URL: https://angular.io/guide/architecture-modules (besucht am 15. 12. 2018).

– Hrsg. (2018d). *Template Syntax*. URL: https://angular.io/guide/template-syntax (besucht am 17. 12. 2018).

Maj, Wojciech (2018). *React Lifecycle Methods diagram*. URL: http://projects.wojtekmaj.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.