

Fakultät Wirtschaft
Studiengang Wirtschaftsinformatik

**Clientseitige Webframeworks
wie Angular und React**

Seminararbeit

Im Rahmen der Prüfung zum Bachelor of Science (B.Sc.)

Verfasser:	Marco Burkart, Tobias Pastrzig, Raphael Schmitt
Kurs:	WWI15B2, WWI15B4
Betreuer:	P. Pohl, D. Ratz, D. Schulmeister-Zimolong
Abgabedatum:	08.01.2018

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich meine Seminararbeit mit dem Thema: Clientseitige Webframeworks wie Angular und React 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, 08.01.2018

(Unterschrift)

(Unterschrift)

(Unterschrift)

Inhaltsverzeichnis

Abbildungsverzeichnis	IV
Listingverzeichnis.....	V
Abkürzungsverzeichnis	VI
1. Einleitung	1
1.1. Überblick.....	1
1.2. Aufbau.....	1
2. Technische Grundlagen	3
2.1. Webframeworks	3
2.1.1. Allgemein.....	4
2.1.1.1. Definition	4
2.1.2. Architekturen im Internet	5
2.1.3. Softwarebibliotheken und Frameworks.....	6
2.2. JavaScript	7
2.2.1. Hintergrund der Sprache	7
2.2.2. Aktueller Standard ECMAScript 2015 (ES6)	8
2.3. TypeScript.....	12
2.3.1. Ansatz	12
2.3.2. Typensystem	13
2.3.3. Vorteile gegenüber JavaScript	16
3. Das TypeScript-Framework Angular	18
3.1. Versionierung.....	18
3.2. Leitkonzepte.....	18
3.2.1. MVC/MVVM.....	19
3.2.2. Dependency Injection	19
3.3. Benötigte Software.....	20
3.3.1. Node.js und npm	20

3.3.2.	Entwicklungsumgebung.....	20
3.3.3.	Angular CLI	21
3.4.	Aufbau eines Projektes	21
3.4.1.	Der Stammordner	21
3.4.2.	Das Quellverzeichnis.....	23
3.5.	Architektur.....	25
3.5.1.	Komponenten	25
3.5.2.	Templates.....	26
3.5.3.	Data Binding.....	27
3.5.3.1.	Property-Bindings.....	27
3.5.3.2.	Event-Bindings	27
3.5.3.3.	Two-Way-Data-Binding	28
3.5.3.4.	Interpolation	29
3.5.4.	Direktiven	29
3.5.5.	Services.....	29
4.	Die JavaScript-Bibliothek React.....	30
4.1.	Komponenten.....	30
4.1.1.	Implementierung einer Komponente als Klasse oder als Funktion	30
4.1.2.	Eigenschaften und Zustand einer Komponente.....	31
4.1.3.	Event-Handling in React-Komponenten	33
4.2.	Virtueller DOM	34
4.3.	Rendern von Elementen	34
4.4.	JSX	35
4.5.	Mehrere Komponenten	35
4.6.	Routing.....	36
4.7.	Die Flux-Architektur.....	37
4.7.1.	Die Bestandteile der Flux-Architektur	37
4.7.2.	Die Struktur und der Datenfluss der Flux-Architektur	38

4.8. Bestandteile eines React-Projekts	38
4.9. Testen von React-Anwendungen	39
5. Fazit	41
Anhang.....	43
Anhang 1: Beigabenverzeichnis	44
Literaturverzeichnis	45

Abbildungsverzeichnis

Abbildung 1: TypeScript und ECMAScript.....	13
Abbildung 2: Architektur einer Angular-Anwendung.....	25
Abbildung 3: Two-Way-Data-Binding Ausgabe	28
Abbildung 4: Der unidirektionale Datenfluss von Flux	38
Abbildung 5: Der Datenfluss als Kreislauf	38

Listingverzeichnis

Listing 1: ES5: Geltungsbereich von var	9
Listing 2: Geltungsbereich von let	10
Listing 3: Prototypische Klasse in ES5.....	10
Listing 4: Klasse in ES6	11
Listing 5: TypeScript: Basistypen und Deklarationen	16
Listing 6: Einbindung der AppComponent in der index.html.....	26
Listing 7: AppComponent mit Selektor "app-root".....	26
Listing 8: Initialisierung von Variablen in der Komponentenklasse.....	27
Listing 9: Property-Binding zum Setzen einer Variable	27
Listing 10: Event-Binding	28
Listing 11: Two-Way-Data-Binding Quellcode.....	28
Listing 12: Verwendung der Interpolationssyntax.....	29
Listing 13: React-Komponente als JavaScript-Funktion.....	31
Listing 14: React-Komponente als ES6-Klasse.....	31
Listing 15: React-Komponente mit Eigenschaften.....	32
Listing 16: React-Komponente mit Zustand	32
Listing 17: Event-Handler in React.....	33
Listing 18: Zuweisung eines JSX-Befehls an eine Variable	35
Listing 19: Übersetzung von JSX in JavaScript.....	35
Listing 20: Komponente als Rückgabewert einer Komponente	36
Listing 21: Beispielverwendung des React-Routers	37

Abkürzungsverzeichnis

CLI	-	Command Line Interface
CSS	-	Cascading Styles Sheets
DI	-	Dependency Injection
DOM	-	Document Object Model
ECMA	-	European Computer Manufacturers Association
ES	-	ECMAScript
HTML	-	Hypertext Markup Language
MVC	-	Model View Controller
MVVM	-	Model View ViewModel
NPM	-	Node Package Manager
URL	-	Uniform Resource Locator

1. Einleitung

Autoren: Marco Burkart, Tobias Pastrzig & Raphael Schmitt

1.1. Überblick

Die Seminar- und Projektarbeit „Stochastische Unternehmensbewertung mittels Methoden der Zeitreihenanalyse“, kurz SUMZ, behandelt die zuverlässige Berechnung des Unternehmenswertes als Grundlage für eine wertorientierte Unternehmensführung. In den Vorgängerjahren wurden hierzu bereits verschiedene Ansätze entwickelt, um den Vorgang der Berechnung informations-technologisch in einer Software zu realisieren und betriebswirtschaftlich anzuwenden. Zum heutigen Stand liegt die dabei entwickelte Software „businesshorizon“ in zwei verschiedenen, aber unfertigen Versionen vor. Zum einen ist dies eine Java-Web-Applikation und zum anderen eine Java-Bibliothek mit Demo-Anwendung.

Im Rahmen des diesjährigen Projekts soll auf Grundlage der Java-Bibliothek eine neue Applikation als clientseitige Webanwendung entstehen. Daher werden in dieser Seminararbeit zwei der derzeit bekanntesten und meist genutzten Frameworks aus diesem Bereich vorgestellt. Es handelt sich hierbei um das von Google entwickelte Angular, sowie Facebooks React. Beide Frameworks sind durch die MIT-Lizenzierung kostenfrei zugänglich und ohne Einschränkungen bei der Verwendung nutzbar.

1.2. Aufbau

Das nachfolgende Kapitel 2 behandelt die technischen Grundlagen dieser Arbeit, sodass ein geeigneter Einstieg in die Thematik gewährleistet werden kann. In Kapitel 2.1 wird auf Webframeworks, ihre Verwendung sowie Ausprägungen eingegangen. Hierbei erfolgt in dem Unterkapitel 2.1.1 Allgemein ein genereller Überblick. Zudem werden verschiedene Vorteile erläutert, welche für die Entwickler durch den Einsatz eines Frameworks entstehen. Das nachfolgende Unterkapitel 2.1.2 Architekturen im Internet umfasst verschiedene Architekturmöglichkeiten von Webframeworks und deren Anwendungen. Hierbei werden die Themen

clientseitige Anwendungen und serverseitige Anwendungen erläutert. Anschließend wird in Kapitel 2.1.3 auf den Unterschied zwischen Softwarebibliothek und Frameworks eingegangen. Das darauffolgende Kapitel 2.2 stellt die Programmiersprache JavaScript vor. Es wird auf den Hintergrund der Sprache eingegangen und der aktuelle Standard ECMAScript 2015 (ES6) veranschaulicht. Anschließend wird in Kapitel 2.3 die Sprache TypeScript aufgeführt. Da diese eine Erweiterung von JavaScript ist, wird auf den Ansatz von TypeScript eingegangen, sowie dessen Typsystem behandelt. Den Abschluss von Kapitel 2.3 bildet die Gegenüberstellung der beiden Programmiersprachen.

Kapitel 3 beschäftigt sich mit dem TypeScript-Framework Angular. Abschnitt 3.2 und 3.3 beschäftigen sich mit einigen Grundlagen zum Framework. Darauf folgt ein Kapitel über den Aufbau der Ordnerstruktur einer Angular-Applikation. In Abschnitt 3.5 werden die wesentlichen Bausteine einer Angular-Anwendung beschrieben.

Anschließend folgt ein Kapitel, das sich mit der JavaScript-Bibliothek React beschäftigt. Dazu wird in Kapitel 4.1 zunächst das Komponenten-Konzept von React vorgestellt, um in den darauffolgenden Kapiteln wesentliche Konzepte und Bestandteile der JavaScript-Bibliothek aufzuführen und zu erläutern. Anschließend wird in Kapitel 4.7 die Flux-Architektur vorgestellt, welche extra von Facebook für React entwickelt wurde. Ebenfalls sind in Kapitel 4.8 die wesentlichen Bestandteile einer React-Anwendung aufgezeigt. Abschließend werden die verschiedenen Möglichkeiten genannt, mit denen eine React-Anwendung getestet werden kann.

Die Seminararbeit schließt in Kapitel 5 mit einem Fazit zu Angular und React ab.

2. Technische Grundlagen

Autor: Raphael Schmitt

Der Inhalt dieses Kapitels hat das Ziel, ein allgemeines Verständnis für die nachfolgenden Webframeworks und die zusammenhängenden Technologien zu vermitteln. Hierbei werden Webframeworks im Allgemeinen vorgestellt und Architekturen aufgezeigt, welche sich durch den Einsatz ebendieser ergeben können. Zudem wird der Unterschied zwischen Java-Bibliothek und Webframework aufgezeigt. Einen weiteren Themenpunkt bilden die Skriptsprachen JavaScript und TypeScript. Dem Leser soll ein Verständlicher Überblick der in der Webprogrammierung gängigen Programmiersprache JavaScript, sowie deren aktuellen Standard „ECMAScript 2015“ vermittelt werden. Anschließend wird das auf JavaScript aufsetzende TypeScript vorgestellt. Dabei wird geschildert, wie diese Sprache in das Umfeld von JavaScript einzuordnen ist. Ebenso wird das Typensystem erläutert und die beiden Programmiersprachen verglichen.

2.1. Webframeworks

In der heutigen Zeit kann immer mehr beobachtet werden, dass viele Desktopanwendungen teilweise oder auch vollständig in das Internet ausgelagert werden und dieses als Plattform nutzen (1&1 Internet SE, 2016; Tchoukio, 2004). Eine solche Anwendung ist über den Browser erreichbar und kann als Webanwendung bezeichnet werden (Nickel, 2010). Einige typisches Beispiele von Webanwendungen bilden unter anderem E-Mail-Dienste, Terminkalender oder Onlineshops. Der klassische Ansatz von Webanwendungen galt der Ausführung auf einem Server, wobei der Browser lediglich für die Darstellung und Benutzerinteraktion genutzt wurde (Hildebrandt, Luthiger, Stamm, & Yereaztian, 2012). Durch die heutzutage eingesetzte Technologie verschiebt sich ein großer Teil des ausführbaren Codes aber auch auf die Clientseite (Hildebrandt u. a., 2012). Der große Vorteil von Webanwendungen liegt darin, dass sie plattformunabhängig in jedem Browser lauffähig sind und keine zusätzliche Installation auf dem Computer benötigt wird (Nickel, 2010). Für die Entwicklung einer Webanwendung stehen heutzutage zahlreiche Frameworks zu Verfügung, auf welche die Programmierer zurückgreifen können (Hildebrandt u. a., 2012). Was es damit auf sich hat

und welche Vorteile sich durch die Verwendung ergeben wird in den nachfolgenden Abschnitten behandelt.

2.1.1. Allgemein

Ein Framework beschreibt in der Software-Architektur einen Entwicklungsrahmen als Grundlage für die Anwendungsprogrammierung (ITWissen, 2012). Als Grundidee von Frameworks kann die Wiederverwendung von Quellcode und Designmustern für spezifische Anwendungsfelder genannt werden (Posch, Birken, & Gerdorf, 2007, Kapitel 8.3.2). Im konkreten Anwendungsfall für Webanwendungen wird von Web Application Frameworks, beziehungsweise der Einfachheit wegen von Webframeworks gesprochen (1&1 Internet SE, 2016; ITWissen, 2009).

2.1.1.1. Definition

Für die Definition des Begriffs Webframework gibt es viele verschiedene Formulierungen, welche am Ende alle in eine sinngemäß ähnliche Bedeutung münden. Zusammenfassend kann ein Webframework wie folgt definiert werden:

Ein Webframework ist ein Software-Framework, welches eine grundlegende Struktur bereitstellt, die die Entwicklung von Webanwendungen vereinfacht und unterstützt. Es kann demnach als eine Art wiederverwendbares Baugerüst bezeichnet werden, um allgemeine Probleme und Architekturfragen bei der Entwicklung zu verringern. (Heinrich & Michelsen, 2011; Nickel, 2010; Shan & Hua, 2006; Techopedia, 2018)

Vorteile für Entwickler

Ein erster nennenswerter Vorteil, der sich durch den Einsatz von Webframeworks ergibt, ist dem Prinzip der Wiederverwendung von Quellcode geschuldet. Das Webframework beinhaltet von Haus aus viele verschiedene Grundfunktionalitäten, sowie eine Menge zusammenarbeitender Klassen und gibt in der Regel auch das Architekturmuster vor (Nickel, 2010). Somit wird dem Entwickler grundlegende Arbeit in Architektur- und Designfragen abgenommen, was zu Zeit- und Kostenreduktion führt (Heinrich & Michelsen, 2011).

Da diese Webframeworks für gewöhnlich von professionellen Entwicklerteams entworfen und von der Nutzergemeinschaft vorangetrieben werden, durchlaufen

Webframeworks verschiedene Entwicklungszyklen und Qualitätsprüfungen (Heinrich & Michelsen, 2011). Demnach kann bei der Verwendung eines Webframeworks der Fokus gezielt auf die Entwicklung der eigenen Geschäftslogik gelegt werden (1&1 Internet SE, 2016).

Des Weiteren ergibt sich aufgrund der vorgegebenen Richtung von Architektur und Designprinzipien oftmals auch ein sauberer Quellcode. Dies kann sich zum einen in eine saubere Trennung der Anwendungsschichten, beispielsweise in Präsentation, Geschäftslogik und Datenhaltung nach dem Model-View-Controller-Prinzip, auswirken. Aber auch andere bekannte Designprinzipien, wie „Don't Repeat yourself“ (Modularisierung) oder „Keep it small and simple“ (minimalistische und leicht verständliche Problemlösungen) sind möglich. (1&1 Internet SE, 2016)

2.1.2. Architekturen im Internet

Die Architektur im Internet lässt sich vereinfacht als das Zusammenspiel von Clients und Servern nach dem Client-Server-Prinzip beschreiben. Nach diesem Prinzip bildet der Client die Benutzerschnittstelle ab und ist demnach der agierende Teil, welcher die Interaktion des Benutzers an den Server als Anfrage weiterleitet. Der Server wiederum kann als reagierender Partner angesehen werden, da dieser die übermittelte Anfrage verarbeitet und die Antwort zurück an den Client übermittelt. (Tchoukio, 2004) Es ist aber auch denkbar gewisse Funktionalität auf die Clientseite zu verschieben. Dies kann zum Beispiel durch den Einsatz von JavaScript geschehen.

Für Webanwendungen ist es wichtig, dass die Funktionen nicht nur ausgeführt werden, sondern der Nutzer die Ergebnisse unmittelbar sehen kann. Um dies zu ermöglichen setzen Webframeworks auf clientseitige Web-Technologien wie Hypertext Markup Language und JavaScript. Dabei gibt es viele verschiedene Frameworks, die primär auf die Verlagerung der Logik auf die Clientseite zielen und somit der größte Teil des Quellcodes in der Webanwendung ausgeführt werden kann. (Thattil, 2016) Mit diesem Ansatz wird der Server beispielsweise nur noch für Datenbankabfragen benötigt. Unter die Rubrik der clientseitigen Webframeworks können die im weiteren Verlauf der Arbeit vorgestellten Frameworks Angular und ReactJS gezählt werden.

Die serverseitige Programmierung setzt sich hingegen nicht dem Ziel, einem Benutzer direktes Feedback zu geben, sondern viel mehr darauf quasi unsichtbar im Hintergrund mit der Datenbank zu interagieren (Thattil, 2016). Mit Technologien wie PHP, ASP.NET oder Java liegt der Fokus weniger auf der Benutzerschnittstelle, sondern primär bei der Verarbeitung von Benutzeranfragen im Backend und der Abfrage von Datenbanken (Thattil, 2016). Somit ist der Fokus auf die serverseitige Architektur. Als Webframeworks für die serverseitige Programmierung können beispielsweise Vaadin und ASP.NET aufgelistet werden.

2.1.3. Softwarebibliotheken und Frameworks

Für die Entwicklung von Softwaresystemen haben sich heutzutage sogenannte Softwarebibliotheken, auch „libraries“ genannt, etabliert. Solche Bibliotheken fassen beispielsweise mehrere Prozeduren oder Funktionen zu einer Einheit zusammen. Es ist auch möglich, zusammengehörende Klassen in einer Bibliothek zu sammeln, wie es in der objektorientierten Softwareentwicklung oft gemacht wird. Als Vorteile von Softwarebibliotheken können Strukturierung durch Zusammenfassung von Funktionalität, Wiederverwendung von Softwarebausteinen und Binärauslieferung genannt werden. Softwarebibliotheken zielen demnach auf die Wiederverwendung von Software, ohne auf ein spezielles Anwendungsgebiet ausgelegt zu sein. (Posch u. a., 2007)

Der Unterschied zu Webframeworks liegt einmal darin, dass ein Webframework aus einem umfangreichen Zusammenspiel von Strukturen, Klassen und Funktionalität besteht, mit deren Verwendung in kurzer Zeit eine lauffähige Anwendung erstellt werden kann (1&1 Internet SE, 2016). Eine Softwarebibliothek bildet vereinfacht gesagt einen kleinen Teil des Webframeworks oder zusätzliche Hilfsmodule. Des Weiteren ist der Verwendungszweck unterschiedlich. Während Webframeworks für ein spezifisches Anwendungsfeld ausgerichtet sind können Softwarebibliotheken vielseitig eingesetzt werden und sind zudem nicht von einem spezifischen Anwendungsgebiet abhängig (Posch u. a., 2007).

2.2. JavaScript

Dieses Kapitel soll dem Leser die Skriptsprache JavaScript und deren aktuellen Sprachstandard näherbringen. Die Erläuterungen werden die Thematik nur oberflächlich beleuchten, eine detaillierte Einführung in die Sprache ist nicht vorgesehen.

2.2.1. Hintergrund der Sprache

Die Programmiersprache JavaScript ist mit ihren vielseitigen Einsatzmöglichkeiten aus der heutigen Webtechnologie nicht mehr wegzudenken. Damit aber verstanden werden kann, warum das so ist, muss ein Blick auf die Geschichte der Sprache und des Internets geworfen werden. Zu Anfangszeiten des World Wide Web bestanden die Internetseiten aus einer statischen Anordnung von Elementen. Mit Hypertext Markup Language (HTML) wurde die Struktur dieser Elemente beschrieben und mittels Cascading Style Sheets (CSS) deren Erscheinungsbild verändert, was beides auch noch heute der Fall ist. Bei Benutzerinteraktionen oder Veränderungen im HTML oder CSS musste die Seite stets komplett neu geladen werden. JavaScript erweiterte die Webseiten mit dynamischer Funktionalität, sowie Interaktivität und kann ebenfalls für die gesamte Steuerungslogik verwendet werden. Somit wurde es auch möglich, einzelne HTML-Elemente im Nachhinein zu verändern oder hinzuzufügen, ohne die Seite neu aufrufen zu müssen. (Kereszturi, 2014)

Die Skriptsprache wurde im Jahre 1995 ursprünglich unter dem Namen „Mocha“ danach „LiveScript“ entwickelt. Nach einem Abkommen der Firmen Netscape und Sun wurde sie letztendlich als JavaScript veröffentlicht. (Krill, 2008) Wenig später übergab Netscape Ihre Entwicklung an die European Computer Manufacturers Association, kurz ECMA, um JavaScript als Industriestandard deklarieren zu lassen (Scharwies, 2017). Brendan Eich, Entwickler der Sprache, erklärt in einem Interview, dass der Begriff ECMAScript aus Marketinggründen entstanden sei, da sich die beteiligten Parteien auf keinen allgemein anerkannten Namen einigen konnten (Krill, 2008). Im weiteren Verlauf erklärt er, wozu JavaScript ursprünglich entwickelt wurde:

The idea was to make something that Web designers, people who may or may not have much programming training, could use to add a little bit of animation or a little bit of smarts to their Web forms and their Web pages. [...] JavaScript was just a little snippet you could write, you could copy somebody else's, you could learn as you went. You didn't have to learn the whole language to use it and you could buy it by the yard.

(Krill, 2008)

Die Idee war demnach, dass JavaScript eine kompakte, leicht zu erlernende Sprache sein soll, um die Webseiten mit mehr Funktionalität zu füllen. Dabei setzt sie auf eine intuitive Programmierung, sodass auch unerfahrene Personen damit arbeiten können, ohne zuvor die gesamte Sprachsyntax lernen zu müssen. Wie Eich schildert, gab es konkrete Vorgaben für die Entwicklung der Sprache, denen zufolge JavaScript viel mehr als der „dumme Bruder“ von Java bezeichnet werden kann (Severance, 2012).

2.2.2. Aktueller Standard ECMAScript 2015 (ES6)

Der Sprachstandard ECMAScript (ES), welcher bei der ECMA unter der Spezifikation ECMA-262 gelistet wird, erfuhr seit seiner ersten Veröffentlichung im Jahre 1997 insgesamt acht Versionen (ECMA International, 2017). Veröffentlicht wurden davon jedoch nur sieben, denn die Arbeit an einem vierten Standard wurde wegen Uneinigkeiten eingestellt (Scharwies, 2017). Der zuständige Arbeitskreis bei der ECMA ist das Technical Committee 93 (Braun, 2015). Mit der im Jahr 2015 veröffentlichten Version ECMAScript 2015, im Volksmund auch ES6 genannt, erhielt der Sprachstandard von JavaScript eine umfangreiche Überarbeitung (Kröner, 2012b). In Folge dessen wurden verschiedene Konzepte ausgebaut und neue implementiert, mit denen die Sprache um einige aus der objektorientierten Programmierung gewohnten Sprachmittel ergänzt wurde (Möllenbeck & Schwichtenberg, 2014). Der weitere Verlauf dieses Kapitels wird eine Auswahl von Änderungen des Standards näher beleuchtet.

Geltungsbereich von Variablen

In den Versionen ES5 und früher werden Variablen mit dem Schlüsselwort „var“ deklariert. Der Geltungsbereich einer Variable ist dabei immer auf die gesamte Funktion bezogen, in welcher die Variable deklariert wurde (Braun, 2015; Möllenbeck & Schwichtenberg, 2014). Grund hierfür ist, dass bei der Interpretation des Codes die Variablendeklaration an den Funktionsbeginn gezogen wird, selbst wenn die Variable innerhalb eines weiteren Blocks deklariert wurde. Dieses Verhalten führt oftmals zu unerwünschten Nebeneffekten. (Kröner, 2012a) In Listing 1 wird vereinfacht die Problematik des übergreifenden Geltungsbereiches skizziert. Angenommen der Code ist einer Funktion entnommen, so wird die Variable in Zeile 3 interpretiert, als wäre sie außerhalb von dem zugehörigen Block (Zeilen 2-4) deklariert worden. Aufgrund des gleichen Variablennamens wird demnach nur eine Variable interpretiert und der Wert aus Zeile 1 beim Erreichen von Zeile 3 unbeabsichtigt überschrieben.

```
1  var x = 1;
2  {
3    var x = 2;
4  }
5  console.log(x); // logs 2
```

Listing 1: ES5: Geltungsbereich von var

(Scholz, 2017)

In ES6 wurde zur Verbesserung dieses Problems die Variablenbezeichnung „let“ eingeführt (Möllenbeck & Schwichtenberg, 2014). Somit ist es möglich den Geltungsbereich einer Variablen auf einen bestimmten Block zu begrenzen (Möllenbeck & Schwichtenberg, 2014), sodass diese außerhalb ihrer Kontrollstruktur nicht sichtbar ist (Kröner, 2012a). Listing 2 auf der nächsten Seite zeigt das gleiche Codebeispiel wie zuvor, unter Verwendung der Variablenbezeichnung „let“. Dies hat zur Folge, dass die Variable in Zeile 3 auf ihren umschließenden Block beschränkt ist und nicht den Wert aus Zeile 1 überschreibt.

```
1  let x = 1;
2  {
3    let x = 2;
4  }
5  console.log(x); // logs 1
```

Listing 2: Geltungsbereich von let

(Scholz, 2017)

Klassensystem

Wie bereits im vorangehenden Kapitel angesprochen wurde, bestanden für die Entwicklung von JavaScript gewisse Vorgaben, welche unter anderem nicht vorsahen ein Klassenkonzept umzusetzen, da die Sprache dann der Programmiersprache Java zu sehr ähnlich werde (Severance, 2012). Die Frage nach einem solchen Konzept wurde aber mit steigender Popularität der Sprache immer größer. In ES gibt es für die Umsetzung von Objekten und Vererbung die sogenannten Prototypen (Braun, 2015). Damit in ES eine Klasse erzeugt werden kann, wird der Code des Prototypen erweitert und somit eine Kapselung simuliert (Catuhe, 2015).

```
1  var Animal = (function () {
2    function Animal(name) {
3      this.name = name;
4    }
5    // Methods
6    Animal.prototype.doSomething = function () {
7      console.log("I'm a " + this.name);
8    };
9    return Animal;
10 })();
11 var lion = new Animal("Lion");
12 lion.doSomething();
```

Listing 3: Prototypische Klasse in ES5

(Catuhe, 2015)

In Listing 3 ist zu erkennen, dass die „Klasse“ Animal als Funktion definiert ist, welche einen Prototypen beinhaltet. Der Funktion sind weitere Funktionen zugefügt, wie beispielsweise „function Animal(name)“ in Zeile 2, welche den Konstruktor der Klasse darstellt. Der Umgang mit Prototypen ist allerdings kein einfaches Konzept und kann daher für Programmierer einer klassenbasierten Sprache schnell verwirrend sein (Catuhe, 2015).

Mit dem Sprachstandard ES6 wurde die Erstellung von Klassen überarbeitet und das Schlüsselwort „class“ eingeführt (Braun, 2015). Eine solche Bezeichnung gab es bereits vorher, jedoch hatte diese nichts, beziehungsweise nicht das gewünschte Verhalten bewirkt (Catuhe, 2015). Als Zusatz für die Klassenbezeichnung steht nun auch die Funktion „constructor()“ zu Verfügung, welche das Objekt initialisiert (Braun, 2015).

```
1  class AnimalES6 {  
2      constructor(name) {  
3          this.name = name;  
4      }  
5  
6      doSomething() {  
7          console.log("I'm a " + this.name);  
8      }  
9  }  
10  
11  var lionES6 = new AnimalES6("Lion");  
12  lionES6.doSomething();
```

Listing 4: Klasse in ES6

(Catuhe, 2015)

Listing 4 zeigt den selben Quellcode wie bereits Listing 3, jedoch in ES6-Schreibweise. Es wird deutlich, dass die Schreibweise von Klassen in ES6 vereinfacht wurde und sehr einer objektorientierten Sprache ähnelt. Für das Ergebnis sind jedoch beide Codeschnipsel identisch, denn auch eine Klasse in ES6 wird letztlich als Funktion mit Prototyp abgebildet (Catuhe, 2015).

2.3. TypeScript

Die Programmiersprache TypeScript (Microsoft, 2017b) ist eine von Microsoft entwickelte Technologie, welche den Sprachstandard von JavaScript erweitert und eine echte Obermenge ebendieser Sprache bildet (Dienst, 2017). Chefentwickler ist Anders Hejlsberg, welcher unter anderem auch an den Sprachen Pascal und C# gearbeitet hat (Springer, 2017). Aufgrund der engen Verbundenheit beider Sprachen ist es möglich, dass jeglicher lauffähige JavaScript-Code, ohne weitere Anpassungen an die geänderte Logik, in TypeScript ausgeführt werden kann (Woiwode, Malcher, Koppenhagen, & Hoppe, 2017). Grund hierfür ist, dass sämtliche TypeScript-Erweiterungen während des Übersetzungsvorganges durch den Compiler entfernt werden und ein reiner JavaScript-Quellcode verbleibt (Roden, 2012). Dies ermöglicht somit die Lauffähigkeit des Programms in allen gängigen Browsern, die auch JavaScript unterstützen (Dienst, 2017).

2.3.1. Ansatz

TypeScript wurde dafür entwickelt, verschiedene Lücken von JavaScript zu füllen und eine einfachere objektorientierte Programmierung zu ermöglichen (Roden, 2012). Zu diesem Ansatz wurden unter Anderem Konzepte wie Klassen, Interfaces und Module eingeführt. Ziel war es, auch große Anwendungen, die letzten Endes in JavaScript vorliegen, einfacher und besser skalierbar programmieren zu können. (Maharry, 2013, Kapitel 1) Die womöglich größte Änderung von TypeScript birgt ein optionales, statische Typsystem für Variablen, wie es auch von Java oder C# bekannt ist (Woiwode u. a., 2017, Kapitel 4). Da dieses Konzept jedoch nicht von JavaScript unterstützt wird, ist die Typsicherheit auf den Zeitraum beschränkt, bis der Übersetzungsvorgang beendet wurde. Die Vorteile entstehen aber dennoch daraus, dass der Compiler durch Typdifferenzen auftretende Fehler schon vor der Programmlaufzeit erkennen kann. (Springer, 2017) In Abbildung 1 auf der nächsten Seite wird aufgezeigt, wie TypeScript in das Umfeld von ECMAScript als Obermenge eingeordnet werden kann.

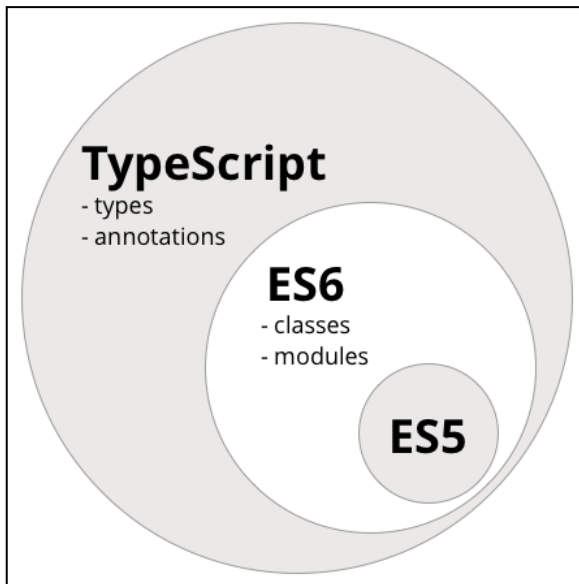


Abbildung 1: TypeScript und ECMAScript

(Murray, Coury, Lerner, & Taborda, 2017)

Den Designzielen kann entnommen werden, dass TypeScript vielmehr als „syntaktischer Zusatz“ (Vogel, 2015) für JavaScript zu verstehen ist, was vor allem die Punkte 2 und 6 der aufgelisteten Ziele verdeutlichen. Diese besagen zum einen, dass TypeScript Mechanismen für eine bessere Struktur, auch bei großer Codebasis, bereitstellt und zum anderen an den Vorgaben und Neuerungen von ECMAScript orientiert ist. (Turner, 2014) Folglich kann TypeScript auch als Werkzeug für das Programmieren bezeichnet werden, um die Qualität von JavaScript zu verbessern (Maharry, 2013, Kapitel 1). Das Ergebnis von kompiliertem TypeScript soll stets sauberes und klares JavaScript sein (Turner, 2014), als wäre von Beginn an in dieser Sprache programmiert worden (Maharry, 2013, Kapitel 1).

2.3.2. Typensystem

Das Hauptmerkmal von TypeScript ist seine statische Typisierung für Variablen (Dienst, 2017). Die Verwendung ist jedoch optional, wodurch auch in TypeScript und dessen Syntax dynamisch typisierter Code möglich ist (Roden, 2012). Aus den Abgrenzungen der Designziele von TypeScript geht hervor, dass gegen ein vollständig korrektes Typsystem entschieden wurde (Turner, 2014). Grund hierfür ist, dass JavaScript kein Typsystem unterstützt und das Zusammenspiel (Interoperabilität) beider Sprachen nicht gebrochen werden soll (Zuvic & Roelofsen, 2016).

TypeScript bietet eine Reihe von Datentypen, die auch aus anderen Sprachen, wie beispielsweise C# und Java, bekannt sind. Die Umsetzung und die Reihenfolge der Typdeklaration birgt jedoch manchen Unterschied. Anders als bei C# oder Java, bei welchen der Datentyp immer vor dem Variablennamen angegeben wird, ist die Reihenfolge bei TypeScript quasi gespiegelt. Zuerst wird die Variable stets mit dem Schlüsselwort „let“ deklariert und im Anschluss erfolgt die Typzuweisung eingeleitet durch einen Doppelpunkt.

TypeScript bietet die folgenden Basistypen an:

- Boolean
- Number
- String
- Array
- Tuple
- Enum
- Any
- Void

Im Folgenden werden die verschiedenen Basistypen erläutert und auf der übernächsten Seite mit dem Codebeispiel Listing 5 veranschaulicht. Die Verweise erfolgen im Text unter Angabe der Zeilennummer.

Der erste Datentyp ist Boolean (Zeile 2), mit welchem einfache wahr-falsch-Werte abgebildet werden können (Microsoft, 2017a).

Mit dem Typ String (Z. 9) kann jegliche Form von Texten abgebildet werden (Microsoft, 2017a).

Eine Besonderheit birgt der Typ Number (Z. 5f.), welcher, wie auch die Zielsprache JavaScript, keine Unterscheidung verschiedener Zahlenformate vorsieht (Microsoft, 2017a). Dies geht darauf zurück, dass in ECMAScript alle Zahlen ausschließlich als Gleitkommazahlen nach „IEEE 754-2008“ definiert sind (ECMA International, 2015, Kapitel 4.3.20). Eine Abweichung von diesem Standard würde einen Bruch in der zuvor angesprochenen Interoperabilität zwischen JavaScript und TypeScript verursachen.

Mit dem Typ Array lassen sich variable Listen erzeugen, welche wahlweise auf zwei Arten definiert werden können (Dienst, 2017). So kann ein Array als solches eines bestimmten Typs (Z. 12) oder als Objekt von Typ Array mit generischem Typ (Z. 13) deklariert werden (Microsoft, 2017a).

Für den Fall, dass bei einem Array die Anzahl der Elemente bekannt ist oder dieses verschiedene Typen enthalten soll, besteht in TypeScript die Möglichkeit ein Tuple zu definieren (Microsoft, 2017a). Die bei der Deklaration angegebenen

Typen und deren Reihenfolge müssen bei der ersten Zuweisung strikt eingehalten werden (Z. 16f. und Z. 19). Für jeden nachfolgenden Tuple-Eintrag (Z. 18) gilt dann lediglich die Prämisse, dass dessen Typ in der Deklaration des Tuples enthalten sein muss (Šuta, 2017).

Ein weiterer Basistyp ist das Enum, welches auch in C# Verwendung findet (Microsoft, 2017a). Hiermit ist es möglich, eine Aufzählung von benannten Konstanten zu definieren, denen eine Ganzzahl zugrunde liegt (Z. 23). Somit kann, beispielsweise bei einer Liste von Zahlenwerten, jedem Zahlenwert eine sprechende Bezeichnung zugewiesen werden. (Wagner, 2017)

Der Typ Any ist ein der dynamischen Typisierung geschuldeter Datentyp (Dienst, 2017). Dieser kann für Variablen genutzt werden, bei denen der zugewiesene Datentyp noch nicht bekannt ist oder dieser sich durch dynamischen Inhalt ergibt (Z. 26f.), und zur Compilezeit nicht geprüft werden soll (Microsoft, 2017a). Der Typ any ist zugleich auch Standardtyp in TypeScript, wodurch jeder Variablen ohne explizite Angabe dieser Typ implizit zugewiesen wird (Roden, 2012).

Der letzte aufgelistete Basistyp ist Void. Dieser kann als das genaue Gegenteil des Typ Any beschrieben werden, nämlich wenn kein Typ vorhanden ist. In den meisten Fällen wird void bei Funktionen ohne Rückgabewert verwendet. (Microsoft, 2017a)

```
1 // Boolean
2 let awnser: boolean = true;
3
4 // Number
5 let num1: number = 25;
6 let num2: number = 1.75;
7
8 // String
9 let str: string = "Hallo Welt";
10
11 // Array
12 let list1: number[] = [1, 23, 45];
13 let list2: Array<number> = [1, 23, 45];
14
15 // Tuple
16 let tuple: [number, string];
17 tuple = [12, "Hallo"]; // OK
18 tuple = [3, "Hallo", "Welt", 2]; // OK
19 tuple = ["Hallo", 12]; // Error:
20     //Type '[string, number]' is not assignable to type '[number, string]'
21
22 // Enum
23 enum weekday { Mon, Tue, Wed, Thu, Fri, Sat, Sun };
24
25 // Any
26 let flex: any = 2;
27 flex = "Hallo Welt" // OK
28
```

Listing 5: TypeScript: Basistypen und Deklarationen

(Quelle: Eigene Darstellung)

2.3.3. Vorteile gegenüber JavaScript

Die Vorteile von TypeScript sind seit der Veröffentlichung und flächendeckenden Unterstützung des Sprachstandards ES5 geringer geworden. Konnte die Sprache zu den Anfangszeiten noch mit dem Klassenkonstrukt und objektorientierter Codestruktur für sich überzeugen, so sind nach dem heutigen Stand „lediglich“ das statische Typensystem und Annotationen, welche automatisiert in Quellcode umgewandelt werden, als Vorteile zu nennen. Nichtsdestotrotz ist die Daseinsberechtigung von TypeScript weiterhin begründet. Es muss sich allerdings zu Beginn der Entwicklung die Frage gestellt werden, in welche Richtung die entstehende Anwendung gehen wird.

Für kleine Anwendungen, bei denen der endgültige Rahmen bereits bekannt ist und diese nach der Fertigstellung nicht mehr erweitert werden und keinen aufwändigen Wartungsprozess benötigen, kann JavaScript bedenkenlos verwendet werden. Ist das Endergebnis, beziehungsweise die Größe der Anwendung unbekannt oder handelt es sich um eine aufwändige Geschäftsanwendung mit wiederkehrendem Wartungsprozess, so sollte aus Gründen der einfacheren Codestrukturierung und leichteren Skalierbarkeit einer großen Codebasis die Entscheidung für TypeScript gefällt werden. (Huber, 2017)

3. Das TypeScript-Framework Angular

Autor: Tobias Pastrzig

Angular ist ein Web-Framework, welches in TypeScript geschrieben ist (Höller, 2017, S. 29). Angular-Applikationen werden in der Regel auch in TypeScript entwickelt. Die Entwicklung mit JavaScript ist jedoch ebenfalls möglich.

3.1. Versionierung

Die erste Version (AngularJS) wurde im Jahr 2009 von den Entwicklern Miško Hevery und Adam Abrons vorgestellt (Hevery, 2009, o.S.). Im Jahr 2016 wurde die Version 2.0.0 vorgestellt. Dabei wurde das Framework von Grund auf neu geschrieben, was dazu führt, dass beide Versionen nicht miteinander kompatibel sind. Seit der Version 2.0.0 spricht man deshalb auch nur noch von Angular (Minar, Ok.. let me explain: it's going to be Angular 4.0, or simply Angular, 2016, o.S.). Die Versionsbezeichnung gliedert sich in drei Nummern, die jeweils durch einen Punkt voneinander separiert werden. Somit entsteht das Muster X.Y.Z. Die Ziffer X (Major) bezeichnet dabei die Hauptversion. Ein Wechsel der Hauptversion ist immer mit sogenannten *Breaking Changes* verbunden. Breaking Changes sind Änderungen, die Codeanpassungen erfordern. Änderungen, die keine Anpassung des Quellcodes erfordern (*Minor Changes*), werden mit der Ziffer Y durchnummeriert, während mit Z (*Patches*) Fehlerbehebungen vorgenommen werden. Zum Zeitpunkt der Erstellung dieser Arbeit liegt Angular in der Version 5.1.3 vor (Fluin, 2017, o.S.).

Die Entwicklung des Frameworks wird von einer Community, angeführt von Google, durchgeführt. Angular ist unter der MIT-Lizenz verfügbar (Google Inc., 2018a, o.S.).

3.2. Leitkonzepte

Dieses Kapitel gibt eine Übersicht über zwei der grundlegenden Konzepte des Frameworks. Angular nutzt einige bewährte Entwurfsmuster und *Best Practices*. Die Konzepte müssen verstanden werden, damit man Angular verstehen kann.

3.2.1. MVC/MVVM

Das Pattern *Model-View-Controller* ist eines der grundlegendsten Entwurfsmuster in der Softwaretechnik. Es bietet eine klare Trennung zwischen Verarbeitungskontrolle, Datenmodell und der Darstellung der Daten indem das Projekt in drei logische Einheiten aufgeteilt wird. Das *Model* stellt das Datenmodell für die Verwaltung der Anwendungsdaten dar. Es gibt auf Anfrage der *View* Daten zurück oder ändert diese durch Befehle des Controllers. Die *View* stellt die Präsentationsschicht dar, die dafür verantwortlich ist, dass der Nutzer stets die aktuellen Daten des Models angezeigt bekommt. Der *Controller* stellt die Geschäftslogik dar. Er nimmt die Interaktion des Benutzers mit der *View* entgegen und manipuliert die Daten des *Models*.

Einen ähnlichen Ansatz verfolgt das Pattern *Model-View-ViewModel*, das als eine Variante von MVC angesehen werden kann. Dabei stellt ein Webserver mit dem *Model* die Daten zu Verfügung. Der Client bekommt im Gegensatz zu MVC zusätzliche Funktionalität durch das *ViewModel*. Es dient als eine Art Proxy-Schicht, die dem aktuellen *View* nur die Daten liefert, die es momentan benötigt und diese im Vorhinein ggf. aufbereitet. Zusätzlich enthält es auch die Funktionalität, die innerhalb der *View* benötigt wird.

Nach welchem der oben genannten Muster eine Angular-App nun aufgebaut ist, führte im Internet zu vielen Diskussionen (AnonDCX, 2016, o.S.). Grundsätzlich kann gesagt werden, dass eine Angular-Applikation eine strikte Trennung zwischen der Datenhaltung (*Model*), der Präsentation (*View*) und der Geschäftslogik (*Controller* oder *ViewModel*) vorsieht. Ein führender Entwickler von Angular, Igor Minar, bezeichnete das Architekturmuster von Angular-Apps als MVW (Model-View-Whatever) und ergänzte, dass Angular jedem Entwickler die Flexibilität gäbe selbst zu entscheiden, was geeigneter für ihn ist (Minar, MVC vs MVVM vs MVP, 2012, o.S.).

3.2.2. Dependency Injection

Zum Erfolg von AngularJS trug unter anderem die Unterstützung von *Dependency Injection* (DI) wesentlich bei (Höller, 2017, S. 249). Benötigt ein Objekt eine Instanz eines anderen Objektes, so erzeugt es diese unter Verwendung

dieses Entwurfsmusters nicht selbst, sondern bekommt die Instanz von einem anderen (oft übergeordneten) Objekt übermittelt (Culp, 2011, o.S.).

DI wird in Angular bei der Verwendung von *Services* (siehe Abschnitt 3.5.5) genutzt. Der Vorteil liegt unter anderem in einer loseren Kopplung der Objekte, wodurch diese unabhängig voneinander getestet werden können.

3.3. Benötigte Software

Bevor ein Angular-Projekt erstellt werden kann müssen die benötigten Werkzeuge installiert werden.

3.3.1. Node.js und npm

Node.js ist eine Plattform die in einer JavaScript-Laufzeitumgebung läuft (Node.js Foundation, 2017, o.S.). Sie wurde ursprünglich dafür entwickelt serverseitig JavaScript-Code auszuführen. Node.js arbeitet als Webserver für die Angular Applikation.

Node.js kommt mit dem zugehörigen *Node Package Manager (npm)*, einem Paketmanager zum einfachen Einbinden von JavaScript-Bibliotheken und zur Verwaltung von Abhängigkeiten in einer Node.js-Applikation (npm, Inc., 2017, o.S.). Über dieses Tool können Frameworks wie Angular, Bibliotheken wie Bootstrap und jQuery und CSS-Präprozessoren wie Less und Sass installiert werden. Der zugehörige Befehl in der Kommandozeile für die Installation eines Paketes lautet wie folgt:

```
npm install <Bibliothek> --save
```

Node.js und npm stehen als Installationspaket unter <https://nodejs.org/en/download/> zum Download zur Verfügung.

3.3.2. Entwicklungsumgebung

Als IDE empfiehlt sich auf eine Software zurückzugreifen die Unterstützung für TypeScript bietet. Als kostenlose Variante ist hier der von Microsoft entwickelte Editor *VisualStudio Code* zu nennen. Er steht unter <https://code.visualstudio.com/> zum Download zur Verfügung.

Eine kostenpflichtige Alternative bietet JetBrains mit dem JavaScript-Editor *Webstorm*. Er kann unter <https://www.jetbrains.com/webstorm/> heruntergeladen werden. Für Studenten steht dieser Editor jedoch auch kostenlos zur Verfügung (siehe <https://www.jetbrains.com/student/>).

3.3.3. Angular CLI

Das Angular *command line interface* (CLI) ist ein Tool, das den Umgang mit Angular-Projekten erheblich erleichtert (Arora, 2017, o.S.). Es ist nicht zwingend notwendig, die Verwendung empfiehlt sich jedoch, weil es Aufgaben, wie das Erstellen eines Projektes und Hinzufügen von Dateien vereinfacht. Des Weiteren werden Aufgaben, wie Testen, Packen und Deployment der Angular-App automatisiert.

Angular CLI kann über *npm* mit dem folgenden Befehl installiert werden (Google Inc., 2017a, o.S.):

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

3.4. Aufbau eines Projektes

Generiert man über das Angular CLI ein neues Projekt, können einen die Dateien und Verzeichnisse zuerst einmal abschrecken. Deshalb soll in diesem Kapitel ein Überblick über die wichtigsten Verzeichnisse und deren Inhalt gegeben werden.

3.4.1. Der Stammordner

Der Stammordner enthält, bis auf den Ordner *src/*, Dateien, die dem Bauen, Testen, Warten, Dokumentieren und Bereitstellen der Applikation dienen (Google Inc., 2018b, o.S.).

Tabelle 1: Aufbau Stammordner

(in Anlehnung an Google Inc., 2018b, o.S.)

Datei/Ordner	Zweck
e2e/	Dieser Ordner enthält Ende-zu-Ende Tests, also Tests, die das Gesamtsystem testen. Sie haben einen eigenen Ordner, weil diese Tests als eigene Applikation laufen.
node_modules/	Dieser Ordner wird von <i>Node.js</i> bzw. <i>npm</i> generiert und enthält alle Bibliotheken von Drittanbietern.
.angular-cli.json	Konfigurationsdatei für das Angular CLI. In dieser Datei können verschiedene Standardwerte gesetzt werden. Außerdem wird hier konfiguriert, welche Dateien bei einem Build miteinbezogen werden.
.editorconfig	Konfigurationsdatei für die IDE um sicherzustellen, dass jeder, der an dem Projekt entwickelt, die gleichen Einstellungen verwendet. ¹
.gitignore	Konfigurationsdatei für das Git-Repository. Damit lassen sich Dateien vom Repository ausschließen.
karma.conf.js	Konfigurationsdatei für Unit-Tests mit dem <i>Karma test runner</i> . ²
package.json	NPM-Konfiguration, die die vom Projekt verwendeten Drittanbieterpakete auflistet (liegen im Ordner <i>node_modules/</i>). Hier können auch eigene benutzerdefinierte Skripte hinzugefügt werden.
protractor.conf.js	Konfigurationsdatei für Ende-zu-Ende Tests mit dem Testframework Protractor. ³ Die zugehörigen Tests befinden sich im Ordner <i>e2e/</i> .

¹ *WebStorm* unterstützt die *.editorconfig*-Datei standardmäßig, bei *VisualStudio Code* muss ein entsprechendes Plugin installiert werden. Weitere Informationen sind unter <http://editorconfig.org/> verfügbar.

² Für weitere Informationen siehe <https://karma-runner.github.io/2.0/index.html>

³ Für weitere Informationen siehe <http://www.protractortest.org>

README.md	Dokumentationsdatei für das Projekt.
tsconfig.json	Konfigurationsdatei für den TypeScript-Compiler.
tslint.json	Konfigurationsdatei für <i>TSLint</i> und <i>Codelyzer</i> zur Überprüfung von <i>TypeScript</i> -Code hinsichtlich Lesbarkeit, Wartbarkeit und dem Auffinden von funktionalen Fehlern (Palantir Technologies, 2018, o.S.). ⁴

3.4.2. Das Quellverzeichnis

Die eigentliche Angular Applikation befindet sich im `src`-Ordner. Hier sind alle Komponenten, Vorlagen, Bilder, usw. enthalten.

Tabelle 2: Aufbau Quellverzeichnis

(in Anlehnung an Google Inc., 2018b, o.S.)

Datei/Ordner	Zweck
app/app.component.{ts,html,css,spec.ts}	Die <i>AppComponent</i> ist die Wurzelkomponente unter der später weitere Blattkomponenten eingefügt werden können. Zu jeder Komponente gehört ein HTML-Template, ein CSS-Stylesheet und eine Datei für Unit-Tests.
app/app.module.ts	Das <i>AppModule</i> ist das Wurzelmodul. Es enthält Informationen wie Angular die Applikation bauen muss. Hier sind alle Komponenten, aus denen die Applikation besteht, eingetragen.
assets/*	Ordner für Dateien wie z.B. Bilder, die beim Bauen der Anwendung mit ausgeliefert werden.
environments/*	Ordner mit Konfigurationsdateien je Laufzeitumgebung für die jeweilige Laufzeitumgebung.

⁴ Für weitere Informationen siehe <https://palantir.github.io/tslint/> und <http://codelyzer.com/>

<code>favicon.ico</code>	Favicon zur Anzeige in der Adresszeile des Webbrowsers
<code>index.html</code>	HTML-Datei, die aufgerufen wird, wenn jemand die Applikation im Webbrowser öffnet. Hier werden dann die einzelnen Module geladen. Die <code>index.html</code> muss in der Regel nicht manuell bearbeitet werden. Die Einbindung von JavaScript- und CSS-Dateien erfolgt automatisch über das CLI.
<code>main.ts</code>	Haupteinstiegspunkt der Applikation. Kompiliert die Applikation zur Ausführung im Browser.
<code>polyfills.ts</code>	Nicht jeder Browser unterstützt die aktuellsten Webtechnologien. Diese Datei enthält Skripte um Unterschiede zwischen Funktionalitäten verschiedener Webbrowser und den Anforderungen der Angular-Applikation zu umgehen.
<code>styles.css</code>	Globale CSS-Datei mit Formatierungen, die das ganze Projekt betreffen.
<code>test.ts</code>	Haupteinstiegspunkt beim Ausführen von Unit-Tests.
<code>tsconfig.{app spec}.json</code>	Konfigurationsdatei des TypeScript Compilers für die Angular Anwendung (<code>tsconfig.app.json</code>) und die Unit-Tests (<code>tsconfig.spec.json</code>).

3.5. Architektur

Nachfolgend werden die wesentlichen Bausteine einer Angular-Anwendung beschrieben.

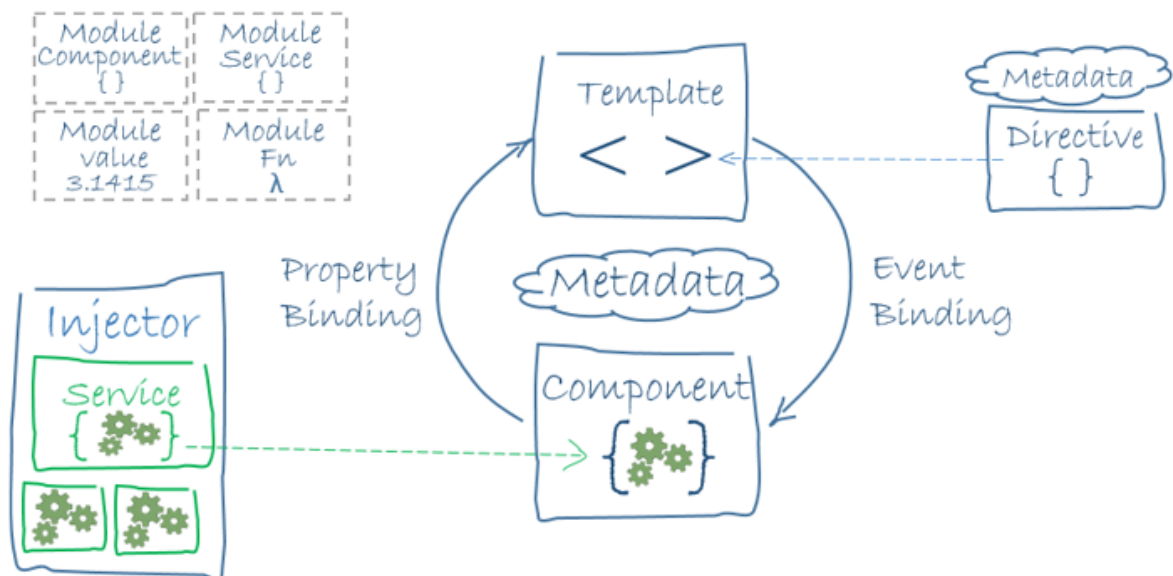


Abbildung 2: Architektur einer Angular-Anwendung

(Google Inc., 2018c, o.S.)

3.5.1. Komponenten

Komponenten sind Elemente, die einen eigenen Sub-DOM aufbauen (Höller, 2017, S. 86). Im Endeffekt ist eine Angular-Anwendung ein hierarchischer Baum aus Komponenten. Die Wurzel bildet dabei die AppComponent. Sie unterscheidet sich nicht von anderen Komponenten. Lediglich der Eintrag in der bootstrap-Eigenschaft der Datei `app.module.ts` führt dazu, dass diese Komponente zur Wurzelkomponente wird.

Jede Komponente verknüpft ein Template mit einer TypeScript-Klasse und kann weitere Sub-Komponenten enthalten. Die hierarchische Organisation von Komponenten ist eines der wichtigsten Architekturprinzipien von Angular. Jede Komponente bildet dabei eine Aufgabe im Projekt ab. Der Vorteil darin liegt unter anderem in der Wiederverwendbarkeit. So werden Komponenten möglichst generisch entwickelt um sie an verschiedenen Stellen im Projekt einsetzen zu können.

In den DOM-Baum eingebunden werden Komponenten über ein eigenes Tag (siehe Listing 6), welches zur Laufzeit mit dem Markup der Komponente gefüllt wird. Der Name des Tags wird in der Komponente durch den Selektor festgelegt (siehe Listing 7).

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>MyApp</title>
6   <base href="/">
7
8   <meta name="viewport" content="width=device-width, initial-scale=1">
9   <link rel="icon" type="image/x-icon" href="favicon.ico">
10 </head>
11 <body>
12   <app-root></app-root>
13 </body>
14 </html>
```

Listing 6: Einbindung der AppComponent in der index.html

```
1 import { Component } from '@angular/core';
2 import { currency } from './currency';
3
4 @Component({
5   selector: 'app-root',
6   templateUrl: './app.component.html',
7   styleUrls: ['./app.component.css']
8 })
9 export class AppComponent {
10   title = 'AppComponent';
11
12   constructor() { }
13 }
```

Listing 7: AppComponent mit Selektor "app-root"

3.5.2. Templates

Jede Komponente hat ein eigenes Template. Templates bilden die View bei MVC bzw. MVVM und sind reguläre html-Dateien, die um Angular spezifische Zusätze erweitert werden können (Google Inc., 2018c, o.S.).

3.5.3. Data Binding

Data-Bindings stellen die Verbindung zwischen Templates und Komponenten dar. Je nach Kommunikationsrichtung unterscheidet man zwischen Property-Bindings und Event-Bindings.

3.5.3.1. Property-Bindings

Property-Bindings ermöglichen es den Wert eines DOM-Elements auf einen Ausdruck (z.B. eine Variable) zu setzen (Höller, 2017, S. 93). Die Syntax sieht dabei wie folgt aus:

```
<elem [property]="Ausdruck">...</elem>
```

Im folgenden Beispiel ist das Value-Property eines Input-Feldes mit Eigenschaft `currency.eur` aus der zugehörigen Komponente verknüpft:

```
12 currency: currency;
13 constructor() {
14   this.currency = {
15     eur: 10,
16     usd: 0,
17     cny: 0
18   }
19 }
```

Listing 8: Initialisierung von Variablen in der Komponentenklasse

(in Anlehnung an Höller, 2017, S. 93)

```
1 <input [value]="currency.eur" />
```

Listing 9: Property-Binding zum Setzen einer Variable

(in Anlehnung an Höller, 2017, S. 93)

Ändert sich nun der Wert `currency.eur` in der Komponente, wird automatisch auch der Wert im Input-Feld aktualisiert.

3.5.3.2. Event-Bindings

Event-Bindings bilden das Gegenstück zu den *Property-Bindings*. Durch sie wird es möglich auf Benutzerinteraktionen mit dem DOM zu reagieren (Höller, 2017, S. 99). Die Syntax sieht dabei wie folgt aus:

```
<elem (eventname)="“aktion“"></elem>
```

Das folgende Beispiel (siehe Listing 10) zeigt ein einfaches Anwendungsszenario. Hier wird bei einer Änderung des Input-Feldes die Variable *currency.eur* in der Komponentenklasse auf den Wert des Input-Feldes gesetzt.

```
1 <input (change)="currency.eur = $event.target.value"/>
```

Listing 10: Event-Binding

(in Anlehnung an Höller, 2017, S. 99)

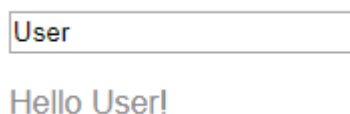
Angular bietet für das Feststellen von Maus-und Tastatureingaben eine Vielzahl von vorkonfigurierten Events an (siehe Höller, 2017, S. 102).

3.5.3.3. Two-Way-Data-Binding

Ein Mechanismus, der bereits für AngularJS populär war und sich auch in Angular wiederfindet, ist die Zwei-Wege-Datenbindung. Sie ist eine logische Kombination von *Property-Binding* und *Event-Binding*. Dabei wird das Datenmodell aktualisiert, wenn der Nutzer eine Eingabe macht. Anders herum wird auch die Benutzeroberfläche aktualisiert, wenn sich das Datenmodell ändert (Höller, 2017, S. 120). Listing 11 und Abbildung 3 zeigen eine stark vereinfachte Umsetzung dieses Konzeptes. Beim Ändern des Eintrags des Input-Feldes fällt dabei auf, dass die Aktualisierung der Ausgabe umgehend erfolgt. Weiterhin fällt hier auf, dass dieses Beispiel ohne viel Boilerplate-Code auskommt, was sich positiv auf die Lesbarkeit auswirkt.

```
1 <input [ngModel]="username" (ngModelChange)="username = $event">
2 <p>Hello {{username}}!</p>
```

Listing 11: Two-Way-Data-Binding Quellcode



User

Hello User!

Abbildung 3: Two-Way-Data-Binding Ausgabe

3.5.3.4. Interpolation

Eine weitere Form des Data-Bindings, die in Angular häufig verwendet wird, ist die Interpolationssyntax. Sie dient der Ausgabe von Strings im HTML-View. Dabei werden die auszugebenden Variablen, Berechnungen oder Methoden in geschweifte Klammern gefasst (siehe Listing 12).

```
1  <div>
2    <p>Verfügbares Guthaben:</p>
3    <p>
4      {{currency.eur}} Euro <br />
5      {{currency.usd}} US-Dollar <br />
6      {{currency.cny}} Renminbi <br />
7    </p>
8  </div>
```

Listing 12: Verwendung der Interpolationssyntax

(in Anlehnung an Höller, 2017, S. 98)

3.5.4. Direktiven

Direktiven sind Komponenten ohne eigenes Template (Höller, 2017, S. 149). Man verwendet sie, wenn man die Anwendung lediglich um Verhalten erweitern möchte. Sie werden mit dem Decorator `@Directive` anstatt `@Component` ausgezeichnet (vgl. Listing 7).

3.5.5. Services

Der dritte Baustein für eine Angular Anwendung sind neben Komponenten und Direktiven sogenannte Services. Ein Service ist eine Klasse mit einer genau definierten und abgegrenzten Aufgabe (Höller, 2017, S. 252). Beispiele dafür sind Protokolldienste, Nachrichtenbus oder Steuerrechner (Google Inc., 2018c, o.S.).

Services können auch als Datendienste zur Anbindung von Ressourcen wie Webdiensten oder Datenbanken ausprogrammiert werden (Google Inc., 2018d, o.S.). Diese Datendienste entsprechen dann bei MVC bzw. MVVM dem Model.

4. Die JavaScript-Bibliothek React

Autor: Marco Burkart

Die JavaScript-Bibliothek React wurde von Facebook entwickelt und steht seit 2013 als Open-Source-Software zur Verfügung (Zeigermann & Hartmann, 2016, S. 3). Nach Facebook (2018, o.S.) dient React der Erstellung von User Interfaces. Das Framework wird bereits von einigen namhaften Webseiten wie z.B. Instagram, Netflix und AirBnB eingesetzt (Zeigermann & Hartmann, 2016, S. 3).

React unterstützt den Anwender bei der Erstellung von komponentenbasierten und funktionalen Interfaces für Web- und Mobile-Applikationen (Alpert, 2017, o.S.). Diese komponentenbasierten Interfaces, vereinfacht Komponenten genannt, stellen den Kern von React dar (vgl. Kapitel 4.1) (Zeigermann & Hartmann, 2016, S. 3).

Da React eine JavaScript-Bibliothek ist, müssen die Anwendungen in JavaScript geschrieben sein, wobei auch eine Übersetzung nach JavaScript möglich ist (Zeigermann & Hartmann, 2016, S. 3).

4.1. Komponenten

Komponenten stellen in React ein wesentliches Element dar (Zeigermann & Hartmann, 2016, S. 3). Sie ermöglichen, die Benutzeroberfläche in unabhängige und wiederverwendbare Stücke zu trennen (Facebook Inc., 2017b, o.S.). Durch das Komponieren, also das Verbinden von Komponenten, entsteht in React eine Anwendung (Zeigermann & Hartmann, 2016, S. 6).

4.1.1. Implementierung einer Komponente als Klasse oder als Funktion

Die Implementierung von Komponenten kann entweder als JavaScript-Funktion oder als Klasse geschehen (Facebook Inc., 2017b, o.S.). Die unterschiedlichen Möglichkeiten sollen im Folgenden anhand der Beispielausgabe „`<h1>Hello SUMZ-World</h1>`“ erläutert werden.

Die Implementierung dieser Beispielausgabe als Komponente in Form einer JavaScript-Funktion ist in Listing 13 dargestellt. Mit der Funktion *createElement(type, props, children)* wird ein neues HTML-Element erzeugt (Facebook

Inc., 2017f, o.S.). Der erste Parameter *type* definiert den Tag-Name des Elements, im Fall der Beispielausgabe also `<h1>` (Zeigermann & Hartmann, 2016, S. 5). Da es keine weiteren Properties gibt, ist anschließend ein *null* aufgeführt, gefolgt von dem Text *Hello SUMZ-World* als einziges Kinder-Element (Zeigermann & Hartmann, 2016, S. 5).

```
function HelloMessage() {  
  return React.createElement('h1', null, 'Hello SUMZ-World');  
}
```

Listing 13: React-Komponente als JavaScript-Funktion

(in Anlehnung an Zeigermann & Hartmann, 2016, S. 4)

Die Beispielausgabe als ES6-Klasse ist in Listing 14 dargestellt. Hier wird ebenfalls die *createElement()*-Funktion zurückgegeben, allerdings innerhalb der *Render*-Methode (vgl. Kapitel 4.3). Wichtig ist, dass die Klasse von *React.Component* erben muss.

```
class HelloMessage extends React.Component {  
  render() {  
    return react.createElement('h1', null, 'Hello SUMZ-World');  
  }  
}
```

Listing 14: React-Komponente als ES6-Klasse

(in Anlehnung an Zeigermann & Hartmann, 2016, S. 5)

4.1.2. Eigenschaften und Zustand einer Komponente

In React gibt es zwei Möglichkeiten, um das Verhalten und das Aussehen von Komponenten zu beeinflussen (Zeigermann & Hartmann, 2016, S. 24). Die Eigenschaften (Properties) einer Komponente werden dieser beim Erzeugen von außen durch das übergeordnete Element als Attribut übergeben (Zeigermann & Hartmann, 2016, S. 24; Facebook Inc., 2017c, o.S.). Danach ist die Eigenschaft für die gesamte Lebenszeit der Komponente gesetzt und kann nicht mehr geändert werden (Facebook Inc., 2017c, o.S.). In Listing 15 ist wieder das „Hello SUMZ-World“-Beispiel zu sehen. Diesmal wird allerdings der Text „SUMZ-World“ als Eigenschaft übergeben. Dadurch erhält die Komponente eine Wiederverwendbarkeit, da sie nur die Art der Darstellung kennt, der Grußtext allerdings variabel ist (Zeigermann & Hartmann, 2016, S. 83). In diesem Beispiel können

also nicht nur das SUMZ-Projekt, sondern auch andere Personen bzw. Organisationen begrüßt werden.

```
function HelloMessage(props) {  
  return React.createElement('h1', null, 'Hello ' + props.hello);  
}  
  
function App() {  
  return (  
    <HelloMessage hello = 'SUMZ-World' />  
  );  
}
```

Listing 15: React-Komponente mit Eigenschaften

(in Anlehnung an Facebook Inc., 2017b, o.S.)

Der Zustand (State) einer Komponente ist ein Objekt, welches nur innerhalb der Komponente sichtbar ist (Zeigermann & Hartmann, 2016, S. 24). Soll das Aussehen und Verhalten auch von der Komponente selbst geändert werden können, wird in React der Zustand verwendet (Facebook Inc., 2017c, o.S.). Dieser stellt also die Daten dar, die beim Rendern von der Komponente verwendet werden (Stefanov, 2017, S. 17). Eine Änderung des Zustands sowie der Eigenschaften führt zu einem neuen Rendern der Komponente (Zeigermann & Hartmann, 2016, S. 25). Wichtig ist, Komponenten mit einem Zustand als Klasse zu implementieren, da Funktionen dies nicht unterstützen (Zeigermann & Hartmann, 2016, S. 89). In Listing 16 ist das Beispiel „Hello SUMZ-World“ mit einem Zustand dargestellt.

```
class HelloMessage extends React.Component {  
  constructor(props) {  
    super();  
    this.state = {comment: props.title};  
  }  
  
  do() { this.setState({comment: 'SUMZ-World'}); }  
  
  render () {  
    this.do();  
    return (React.createElement('h1', null, 'Hello ' + this.state.comment));  
  }  
}  
  
ReactDOM.render(  
  <HelloMessage title = 'BPM 4.0' />, document.getElementById('root')  
);
```

Listing 16: React-Komponente mit Zustand

(Eigene Darstellung)

Der Zustand einer Komponente kann mit *this.state* abgefragt und mit der Methode *this.setState()* gesetzt werden (Zeigermann & Hartmann, 2016, S. 89-90). Der Zustand kann auf zwei Arten gesetzt werden: entweder im Konstruktor mit *this.state={}* oder mit *this.setState()* (Zeigermann & Hartmann, 2016, S. 92). Wurde der Zustand nicht im Konstruktor gesetzt, ist er bis zum Setzen durch *this.setState()* undefined (Zeigermann & Hartmann, 2016, S. 92).

Listing 16 zeigt nicht nur das Setzen des Zustands, sondern verdeutlicht auch den Unterschied gegenüber den Eigenschaften. Die Klasse enthält als Eigenschaft (*props*) den Titel „BPM 4.0“ und initialisiert den Zustand damit im Konstruktor. Die Ausgabe wäre in diesem Fall „Hello BPM 4.0“. Da aber eigentlich das SUMZ-Team begrüßt werden soll, wird der Zustand innerhalb der *do()-Methode* zu „SUMZ-World“ verändert. Dadurch wird die Ausgabe zu „Hello SUMZ-World“ korrigiert. Eine Änderung der Eigenschaft innerhalb der Klasse wäre nicht möglich gewesen.

4.1.3. Event-Handling in React-Komponenten

Die Behandlung von Events ist in React ähnlich der Behandlung von Events in HTML, es gibt aber kleine syntaktische Unterschiede (Facebook Inc., 2017e, o.S.). Wichtig ist, dass die an den Handler übergebenen Event-Objekte keine Events des nativen Document Object Model (DOM) sind, sondern eine Eigenheit von React (Zeigermann & Hartmann, 2016, S. 129). Diese synthetischen Events dienen der Kompatibilität zwischen verschiedenen Browsern und verhindern somit Inkonsistenzen (Stefanov, 2017, S. 23). Ein Beispiel für einen Event-Handler ist in Listing 17 dargestellt.

```
function Hello({hello}) {  
  return <h1 onClick={e => console.log('Event: ' + e)}>  
    {hello}  
  </h1>;  
}
```

Listing 17: Event-Handler in React

(in Anlehnung an Zeigermann & Hartmann, 2016, S. 129)

4.2. Virtueller DOM

Gibt es eine Änderung an einer React-Komponente, wird durch die Render-Methode (vgl. Kapitel 4.3) die gesamte Benutzeroberfläche neu erzeugt (Zeigermann & Hartmann, 2016, S. 23). Da der DOM-Baum einer Anwendung meistens sehr groß ist, ist eine neue Erzeugung des DOM nach jeder Änderung nicht performant (Voutilainen, Mikkonen, & Systä, 2016, S. 146).

React hat mit der neuen Technologie des virtuellen DOM darauf reagiert (Voutilainen, Mikkonen, & Systä, 2016, S. 146). Der virtuelle DOM (VDOM) ist ein Programmierkonzept, bei dem eine abstrakte Kopie der Benutzeroberfläche im Speicher gehalten wird, um diese mit dem nativen DOM zu synchronisieren (Facebook Inc., 2017j, o.S.; Voutilainen, Mikkonen, & Systä, 2016, S. 146). Der neu erzeugte VDOM wird mit der vorherigen Version verglichen und somit die Unterschiede festgestellt, die im realen DOM geändert werden müssen (Zeigermann & Hartmann, 2016, S. 24). React ändert den echten DOM durch geeignete Operationen so ab, dass er dem Zustand des virtuellen DOM entspricht (Zeigermann & Hartmann, 2016, S. 24). Es muss also nur der Zustand beschrieben werden, in welchem sich die Benutzeroberfläche befinden soll, React sorgt dann für die Umsetzung (Facebook Inc., 2017j, o.S.; Zeigermann & Hartmann, 2016, S. 24). Außerdem ist React durch die minimalen Änderungen am realen DOM sehr schnell (Zeigermann & Hartmann, 2016, S. 24).

4.3. Rendern von Elementen

Mit Hilfe der Render-Methode *React.DOM.render(parameter1, parameter2)* kann ein React-Element in den realen DOM eingefügt werden (Zeigermann & Hartmann, 2016, S. 66). Als ersten Eingabeparameter erwartet die Methode das Element, das in den realen DOM eingefügt werden soll (Zeigermann & Hartmann, 2016, S. 67). Unter *parameter2* soll der Container angegeben werden, in den das Element einzufügen ist (Facebook Inc., 2017g, o.S.). Dieser gibt auch eine Referenz auf das DOM-Element zurück (Facebook Inc., 2017g, o.S.).

Beim ersten Aufruf der Render-Methode werden alle Elemente innerhalb des Containers im realen DOM durch den Inhalt des Containers im virtuellen DOM ersetzt (Zeigermann & Hartmann, 2016, S. 67). Wenn die Methode ein weiteres

Mal aufgerufen wird und sich *parameter1* und *parameter2* nicht geändert haben, wird der in Kapitel 4.2 geschriebene Abgleich zwischen nativem DOM und virtuellem DOM durchgeführt, wodurch nur die Änderungen in den realen DOM übernommen werden (Zeigermann & Hartmann, 2016, S. 67).

4.4. JSX

In React wird die Darstellung einer Komponente nicht in HTML, sondern in JavaScript geschrieben (Zeigermann & Hartmann, 2016, S. 59). Dafür gibt es eine Spracherweiterung von JavaScript namens JSX, welche durch einen Compiler in JavaScript übersetzt wird (Zeigermann & Hartmann, 2016, S. 59). Die Erweiterung ist zwar optional, bietet aber mehr Übersichtlichkeit (Stefanov, 2017, S. 65). Betrachtet man beispielsweise das „Hello SUMZ-World“-Beispiel aus den vorherigen Kapiteln, kann man den HTML-Befehl `<h1>Hello SUMZ-World</h1>` mit Hilfe von JSX einfach einer JavaScript-Variablen zuweisen (Zeigermann & Hartmann, 2016, S. 59). Die komplette Zuweisung ist in Listing 18 zu sehen.

```
cons helloSUMZ = <h1>Hello SUMZ-World</h1>;
```

Listing 18: Zuweisung eines JSX-Befehls an eine Variable

(in Anlehnung an Zeigermann & Hartmann, 2016, S. 59)

Aus dieser Zuweisung generiert der Compiler den JavaScript-Code in Listing 19. Dieser ist deutlich länger und unübersichtlicher als in der JSX-Notation.

```
var helloSUMZ = React.createElement("h1", null, "Hello SUMZ-World");
```

Listing 19: Übersetzung von JSX in JavaScript

(in Anlehnung an Zeigermann & Hartmann, 2016)

4.5. Mehrere Komponenten

Komponenten können auch auf andere Komponenten verweisen bzw. in ihnen verwendet werden (Facebook Inc., 2017b, o.S.). Dabei behandelt man die Komponente wie ein normales HTML-Element, d.h. man schreibt den Namen der Komponente im JSX-Syntax (Zeigermann & Hartmann, 2016, S. 111). Ein Beispiel dafür ist in Listing 20 gegeben, in dem eine Komponente zweimal als Rückgabewert einer anderen Komponente ausgegeben wird.

Dadurch zeigt sich das Entwicklungskonzept von React. Elemente wie z.B. Buttons oder auch Eingabefelder für Texte werden in React als Komponenten geschrieben (Facebook Inc., 2017b, o.S.). Ob die Komponente als Funktion oder als Klasse implementiert ist hat dabei keine Bedeutung (Zeigermann & Hartmann, 2016, S. 112).

```
function HelloWorld(props) {  
  return <h1>Hello {props.name}</h1>;  
}  
  
function Greet() {  
  const member = <HelloWindow name = "SUMZ-member" />;  
  const lecturer = <HelloWindow name = "SUMZ-lecturer"/>;  
  return (  
    <div>  
      {member}  
      {lecturer}  
    </div>  
  );  
}
```

Listing 20: Komponente als Rückgabewert einer Komponente

(in Anlehnung an Facebook Inc., 2017b, o.S.; Zeigermann & Hartmann, 2016, S. 112)

Wie in Listing 20 ebenfalls zu erkennen ist, kann eine Komponente auch einer Variablen zugewiesen werden (Zeigermann & Hartmann, 2016, S. 112).

4.6. Routing

Um über Uniform Resource Locators (URL) auf spezifische Komponenten zu verweisen oder auch den Zustand einer Komponente in einer URL zu speichern, benötigt man in React eine zusätzliche Bibliothek, da dies nicht standardmäßig unterstützt wird (Zeigermann & Hartmann, 2016, S. 183). Als De-facto-Standard hat sich dafür der React Router⁵ durchgesetzt (Zeigermann & Hartmann, 2016, S. 183). In Listing 21 ist ein Beispiel dargestellt, in dem die Router-Komponente über die URL an andere Komponenten verweist.

⁵ <https://reacttraining.com/react-router/>

```
import { BrowserRouter as Router, Route, Link } from 'react-router-dom';

const Routing = () => (
  <Router>
    <div>
      <ul>
        <li><Link to="/">Home</Link></li>
        <li><Link to="/sumz-beschreibung">SUMZ-Beschreibung</Link></li>
      </ul>
      <Route exact path="/" component={Home} />
      <Route path="/sumz-beschreibung" component={SUMZ} />
    </div>
  </Router>
);
```

Listing 21: Beispielverwendung des React-Routers

(in Anlehnung an Facebook Inc., 2017d, o.S.)

4.7. Die Flux-Architektur

Flux ist eine Anwendungsarchitektur für die Entwicklung clientseitiger Webanwendungen, die zusammen mit React bei Facebook entstanden ist (Facebook Inc., 2015, o.S.; Zeigermann & Hartmann, 2016, S. 229). Bei der Anwendung von React ist Flux der Quasistandard und bietet aufgrund der einheitlichen Architektur bei großen Entwicklungen bessere Wartbarkeit und Übersichtlichkeit (Zeigermann & Hartmann, 2016, S. 229). Dabei ist Flux allerdings rein optional und steht für eine Idee, wie man die Daten in einer React-Anwendung organisieren und strukturieren kann (Stefanov, 2017, S. 167).

4.7.1. Die Bestandteile der Flux-Architektur

Die Flux-Architektur besteht aus einem Dispatcher sowie einem oder mehreren Stores, Views und Actions (Facebook Inc., 2015, o.S.). In den Stores werden die Daten der Anwendung gehalten, welche von den Views – also den React-Komponenten – ausgelesen und gerendert werden (Stefanov, 2017, S. 168). Die Stores enthalten also den Anwendungsstatus und die gesamte Logik der Anwendung (Facebook Inc., 2015, o.S.). Außerdem bietet jeder Speicher eine callback-Funktion, mit der er sich beim Dispatcher registriert (Facebook Inc., 2015, o.S.). Der Dispatcher ist die zentrale Verwaltung für alle Datenflüsse in einer Flux-Architektur (Facebook Inc., 2015, o.S.). Die Verwaltung besteht darin, dass der Dispatcher die Actions an alle registrierten Stores weiterleitet. Die Actions legen fest,

was im Store geändert wird und werden meistens als Reaktion auf Benutzerinteraktionen von den Views ausgelöst (Stefanov, 2017, S. 177).

4.7.2. Die Struktur und der Datenfluss der Flux-Architektur

Der Datenfluss der Flux-Architektur ist in Abbildung 4 dargestellt. Auffällig ist, dass alle Daten nur in eine Richtung fließen, was auch als unidirektionaler Datenfluss bezeichnet wird (Facebook Inc., 2015, o.S.). Dieser unidirektionale Datenfluss erleichtert das logische Folgern des Datenflusses als auch das Debuggen (Stefanov, 2017, S. 168).

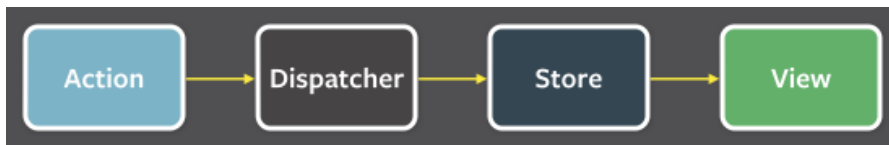


Abbildung 4: Der unidirektionale Datenfluss von Flux

(Facebook Inc., 2015, o.S.)

Erzeugt die View als Ergebnis einer Benutzerinteraktion eine neue Action, wird diese wieder an den Dispatcher weitergeleitet (Facebook Inc., 2015, o.S.). Dadurch entsteht ein vollständiger Kreislauf (Stefanov, 2017, S. 183), der in Abbildung 5 dargestellt ist.

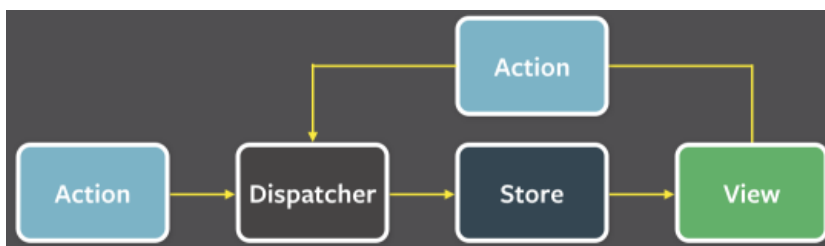


Abbildung 5: Der Datenfluss als Kreislauf

(Facebook Inc., 2015, o.S.)

4.8. Bestandteile eines React-Projekts

Im Folgenden soll aufgezeigt werden, aus welchen Teilen ein React-Projekt im Optimalfall bestehen sollte. Ein React-Projekt sollte mindestens aus drei Teilen bestehen:

- Einem Übersetzer (Compiler):

Wie in Kapitel 4.4 beschrieben, ist für die Übersetzung von JSX in JavaScript ein Compiler notwendig. Die React-Entwickler empfehlen dazu

die Verwendung des Übersetzers Babel⁶ (Facebook Inc., 2017a, o.S.). Babel unterstützt nicht nur die Übersetzung von JSX in JavaScript, sondern übersetzt auch ES6 nach ES5 (Zeigermann & Hartmann, 2016, S. 8).

- Einem Modul-Lader (Bundler):

In React wird eine Anwendung durch die Verwendung von ES6-Modulen gegliedert (Zeigermann & Hartmann, 2016, S. 9). Dadurch sollen Abhängigkeiten aufgelöst und in einer Ausgabedatei gesammelt werden (Zeigermann & Hartmann, 2016, S. 9). React empfiehlt hierfür die Verwendung von Webpack⁷ oder Browserify⁸ (Facebook Inc., 2017a, o.S.).

- Einem Paket-Manager (Package Manager):

Ein Paket-Manager unterstützt bei der Verwaltung der Pakete von Drittanbietern (Facebook Inc., 2017a, o.S.). Die Entwickler von React empfehlen die Benutzung von Yarn⁹ oder npm¹⁰ (Facebook Inc., 2017a, o.S.).

Bei großen Anwendungen ist außerdem empfehlenswert, auch ein geeignetes Architekturmuster (vgl. Kapitel 4.7) sowie das Routing (vgl. Kapitel 4.6) zu bedenken (Zeigermann & Hartmann, 2016, S. 8).

4.9. Testen von React-Anwendungen

Im Folgenden sollen die verschiedenen Möglichkeiten zum Testen von React-Anwendungen betrachtet werden. Als erstes muss entschieden werden, was und wie getestet werden soll. Möchte man Unit-Tests ohne Rendering durchführen, also Code außerhalb von Komponenten testen, ist das mit normalen JavaScript-Testmethoden und -Testwerkzeugen möglich (Zeigermann & Hartmann, 2016, S. 276).

Sollen Komponenten getestet werden, müssen diese gerendert werden (Zeigermann & Hartmann, 2016, S. 276). Eine Methode, die dafür keinen DOM benötigt, ist das Shallow Rendering (Facebook Inc., 2017h, o.S.). Dabei wird eine Komponente nur eine Ebene tief gerendert, wodurch das Verhalten von Kind-

⁶ <https://babeljs.io/>

⁷ <https://webpack.js.org/>

⁸ <http://browserify.org/>

⁹ <https://yarnpkg.com/lang/en/>

¹⁰ <https://www.npmjs.com/>

Komponenten keine Rolle spielt (Facebook Inc., 2017h, o.S.). Sollen Komponenten in einen nativen DOM gerendert werden, kann bspw. das *jsdom*-Framework¹¹ verwendet werden (Zeigermann & Hartmann, 2016, S. 277). Das Testen der Interaktion der Anwendung mit einem echten Browser ist durch einen Testrunner wie *Karma*¹² möglich (Zeigermann & Hartmann, 2016, S. 277).

Weiterhin bringt React selbst einige Möglichkeiten zum Testen einer Anwendung mit. Die Testfunktionen sind unter *ReactTestUtils* zu finden und können zusammen mit jedem geeigneten Testframework für React verwendet werden (Facebook Inc., 2017i, o.S.). Als Testframework empfehlen Zeigermann & Hartmann (2016, S. 271) *mocha*¹³, da dieses Framework bei React-Projekten am häufigsten Verwendung findet. Aber auch hierfür hat Facebook eine Eigenentwicklung im Zusammenhang mit React, nämlich das *Jest*-Framework¹⁴ (Zeigermann & Hartmann, 2016, S. 271).

¹¹ <https://github.com/tmpvar/jsdom>

¹² <https://karma-runner.github.io/2.0/index.html>

¹³ <https://mochajs.org/>

¹⁴ <https://facebook.github.io/jest/>

5. Fazit

Autoren: Marco Burkart & Tobias Pastrzig

Das Framework Angular und die JavaScript-Bibliothek React unterscheiden sich in einigen wesentlichen Punkten, wovon die drei größten nachfolgend betrachtet werden.

Ein großer Unterschied ist die Entwicklungssprache. Angular ist in TypeScript geschrieben und auch die Angular-Applikationen werden in i.d.R. in TypeScript entwickelt. React-Anwendungen werden in JavaScript entwickelt. TypeScript bietet dabei einige Vorteile, wie beispielsweise Typsicherheit und eine bessere Lesbarkeit des Codes, muss jedoch - im Gegensatz zu JavaScript - oft erst erlernt werden.

Ein weiterer Hauptunterschied liegt in den Architekturmustern für die Angular und React entwickelt werden. Wie in Abschnitt 3.2.1 aufgeführt, können Angular-Applikationen sowohl nach dem MVC- als auch nach dem MVVM-Muster aufgebaut sein, wonach der Controller oder das ViewModel aufgrund von Benutzerinteraktionen mit der View die Datenmanipulation im Model ansteuern. Im Gegensatz dazu arbeitet React mit dem Flux-Architekturmuster, wie in Abschnitt 4.7 erläutert. Der eingesetzte Dispatcher leitet lediglich die Actions an die verschiedenen Stores weiter. Die Anwendungslogik steckt hingegen in den Stores. Weiterhin arbeiten MVC und MVVM mit einem bidirektionalen Datenfluss. Im Gegensatz dazu arbeitet das Flux-Architekturmuster aus React mit einem unidirektionalen Datenfluss. Dadurch gibt es weniger Abhängigkeiten im Programmcode, wodurch Fehler einfacher nachzuvollziehen sind. Genau diese Wartbarkeitsprobleme will Facebook mit der Flux-Architektur mindern (Zeigermann & Hartmann, 2016, S. 270).

Der dritte große Unterschied liegt in Umfang der beiden Frameworks. Angular kann man als ein Rundum-Sorglos-Paket sehen, das für alle Aufgaben im Projekt die passenden Werkzeuge zur Verfügung stellt. Der Preis für so ein mächtiges Framework besteht jedoch darin, dass es gerade am Anfang viel Einarbeitungszeit benötigt um selbst kleine Projekte umsetzen zu können. So kommt Angular standardmäßig unter anderem mit eigenen Frameworks für Unit-Tests, Ende-zu-Ende-Tests, Animationen sowie für mehrsprachige und geschlechterspezifische

Anwendungen (wurden in dieser Arbeit nicht behandelt). React wirkt dagegen wesentlich leichtgewichtiger. Der Einstieg erfolgt schnell, weitere Bibliotheken müssen jedoch manuell gesucht und installiert werden.

Abschließend kann man sagen, dass sich für Applikationen im Browser der Einsatz eines Web-Frameworks heutzutage empfiehlt, da sie die Umsetzung und Wartung erheblich erleichtern.

Anhang

Anhang 1: Beigabenverzeichnis

1. Verzeichnis „Seminararbeit“
2. Verzeichnis „Elektronische Quellen“

Literaturverzeichnis

- 1&1 Internet SE. (2016). Webframeworks – ein Überblick. Von <https://hosting.1und1.de/digitalguide/websites/web-entwicklung/webframeworks-ein-ueberblick/> abgerufen
- Alpert, S. (2017). *React 16: A look inside an API-compatible rewrite of our frontend UI library*. Von <https://code.facebook.com/posts/1716776591680069/react-16-a-look-inside-an-api-compatible-rewrite-of-our-frontend-ui-library/> abgerufen
- AnonDCX. (2016). *Angular2: MVC, MVVM or MV*?* Von stack overflow: <https://stackoverflow.com/questions/36950582/angular2-mvc-mvvm-or-mv> abgerufen
- Arora. (2017). *Angular CLI*. Von Github: <https://github.com/angular/angular-cli/wiki> abgerufen
- Braun, H. (2015). JavaScript wird erwachsen. *c't Magazin, Ausgabe 2*, 168–174.
- Catuhe, D. (2015). Die neue JavaScript-Standardisierung: So funktionieren Klassen und Vererbung mit ECMAScript 6. Von <https://t3n.de/news/ecmascript-6-verstehen-klassen-657062/> abgerufen
- Culp, A. (2011). *The Dependency Injection Design Pattern*. Von Microsoft Developer Network: [https://msdn.microsoft.com/en-us/library/hh323705\(v=vs.100\).aspx](https://msdn.microsoft.com/en-us/library/hh323705(v=vs.100).aspx) abgerufen
- Dienst, J. (2017). TypeScript Tutorial: Grundlagen und Typisierung für JavaScript. Von <https://jaxenter.de/tutorial-typescript-typisierung-59192> abgerufen
- ECMA International. (2015). ECMAScript® 2015 Language Specification. Von <http://www.ecma-international.org/ecma-262/6.0/> abgerufen
- ECMA International. (2017). Standard ECMA-262. Von <http://www.ecma-international.org/publications/standards/Ecma-262-arch.htm> abgerufen

- Facebook Inc. (2015). *Flux: In Depth Overview*. Von facebook.github.io:
<https://facebook.github.io/flux/docs/in-depth-overview.html#content>
abgerufen
- Facebook Inc. (2017a). *Add React to an Existing Application*. Von
<https://reactjs.org/docs/add-react-to-an-existing-app.html> abgerufen
- Facebook Inc. (2017b). *Components and Props*. Von
<https://reactjs.org/docs/components-and-props.html> abgerufen
- Facebook Inc. (2017c). *Components, Props and State*. Von
<https://facebook.github.io/react-vr/docs/components-props-and-state.html>
abgerufen
- Facebook Inc. (2017d). *Context*. Von
<https://reactjs.org/docs/context.html#parent-child-coupling> abgerufen
- Facebook Inc. (2017e). *Handling Events*. Von <https://reactjs.org/docs/handling-events.html> abgerufen
- Facebook Inc. (2017f). *React Top-Level API*. Von <https://reactjs.org/docs/react-api.html#createelement> abgerufen
- Facebook Inc. (2017g). *ReactDOM*. Von <https://reactjs.org/docs/react-dom.html#render> abgerufen
- Facebook Inc. (2017h). *Shallow Renderer*. Von <https://reactjs.org/docs/shallow-renderer.html> abgerufen
- Facebook Inc. (2017i). *Test Utilities*. Von <https://reactjs.org/docs/test-utils.html> abgerufen
- Facebook Inc. (2017j). *Virtual DOM and Internals*. Von reactjs.org:
<https://reactjs.org/docs/faq-internals.html#what-is-the-virtual-dom>
abgerufen
- Facebook Inc. (2018). *React*. Von
<https://code.facebook.com/projects/176988925806765/react/> abgerufen

- Fluin, S. (6.. Dezember 2017). *Angular 5.1 & More Now Available*. Von blog.angular.io: <https://blog.angular.io/angular-5-1-more-now-available-27d372f5eb4e> abgerufen
- Google Inc. (2017a). *CLI tool for Angular*. Von GitHub: <https://github.com/angular/angular-cli> abgerufen
- Google Inc. (2018a). *The MIT License*. Von Angular: <https://angular.io/license> abgerufen
- Google Inc. (2018b). *QuickStart*. Von angular.io: <https://angular.io/guide/quickstart> abgerufen
- Google Inc. (2018c). *Architecture Overview*. Von Angular: <https://angular.io/guide/architecture> abgerufen
- Google Inc. (2018d). *Services*. Von Angular: <https://angular.io/tutorial/toh-pt4> abgerufen
- Heinrich, A., & Michelsen, B. (2011). *Web-Technologien - Grundlagen, ausgewählte Systeme und Frameworks*. Otto-Friedrich-Universität Bamberg.
- Hevery, M. (28. September 2009). *Hello World, <angular/> is here*. Von Miško Hevery - The Testability Explorer Blog: <http://misko.hevery.com/2009/09/28/hello-world-angular-is-here/> abgerufen
- Hildebrandt, A., Luthiger, J., Stamm, C., & Yereaztian, C. (2012). Eine Kategorisierung mobiler Applikationen. Von <https://irf.fhnw.ch/handle/11654/17875> abgerufen
- Höller, C. (2017). *Angular - Das umfassende Handbuch*. Bonn: Rehinwerk Verlag GmbH.
- Huber, T. C. (2017). *Einstieg in TypeScript Grundlagen für Entwickler*. entwickler.press.
- ITWissen. (2009). Web-Framework. Von <http://www.itwissen.info/Web-Framework-web-framework.html> abgerufen

- ITWissen. (2012). Framework. Von <http://www.itwissen.info/Framework-framework.html> abgerufen
- Kereszturi, A. (2014). Aller guten Dinge sind 3: HTML, CSS und JavaScript im Zusammenspiel. Von <https://news.digicomp.ch/de/2014/08/21/aller-guten-dinge-sind-3-html-css-und-javascript-im-zusammenspiel/> abgerufen
- Krill, P. (2008). JavaScript creator ponders past, future. Von <https://www.infoworld.com/article/2653798/application-development/javascript-creator-ponders-past--future.html> abgerufen
- Kröner, P. (2012a). ECMAScript 6: Block Scope und Konstanten. Von <http://www.peterkroener.de/ecmascript-6-block-scope-und-konstanten/> abgerufen
- Kröner, P. (2012b). ECMAScript 6: Die nächste Ausbaustufe von JavaScript. Von <http://www.peterkroener.de/ecmascript-harmony-die-naechste-ausbaustufe-von-javascript/> abgerufen
- Maharry, D. (2013). *TypeScript revealed*. Apress.
- Microsoft. (2017a). Basic Types · TypeScript. Von <https://www.typescriptlang.org/docs/handbook/basic-types.html> abgerufen
- Microsoft. (2017b). TypeScript. Von <https://www.typescriptlang.org/> abgerufen
- Minar, I. (2012). *MVC vs MVVM vs MVP*. Von Google+: <https://plus.google.com/+AngularJS/posts/aZNVhj355G2> abgerufen
- Minar, I. (13. Dezember 2016). *Ok.. let me explain: it's going to be Angular 4.0, or simply Angular*. Von juristr.com: <https://juristr.com/blog/2016/12/let-me-explain-angular-release-cycles/> abgerufen
- Möllenbeck, M., & Schwichtenberg, D. H. (2014). Neue Sprachfeatures im ECMAScript-6-Entwurf – Teil 1. Von <https://www.heise.de/developer/artikel/Neue-Sprachfeatures-im-ECMAScript-6-Entwurf-Teil-1-2398267.html?seite=all> abgerufen
- Murray, N., Coury, F., Lerner, A., & Taborda, C. (2017). ng-book. Von <https://www.ng-book.com/images/ng2/previews/typescript/es5-es6->

typescript-circle-diagram.png abgerufen

Nickel, H. (2010). *Analyse und Vergleich von Web-Frameworks im Bereich der agilen Webentwicklung anhand einer Beispielanwendung*. Hochschule für Technik und Wirtschaft Dresden.

Node.js Foundation. (2017). *About Node.js®*. Von nodejs.org:
<https://nodejs.org/en/about/> abgerufen

npm, Inc. (2017). *Features*. Von npm: <https://www.npmjs.com/features>
abgerufen

Palantir Technologies. (2018). *TSLint*. Von GutHub:
<https://palantir.github.io/tslint/> abgerufen

Posch, T., Birken, K., & Gerdorf, M. (2007). *Basiswissen Softwarearchitektur : verstehen, entwerfen, wiederverwenden* (2., übera.). Heidelberg: Dpunkt-Verl.

Roden, G. (2012). TypeScript: Microsofts neues typisiertes JavaScript. Von
<https://www.heise.de/developer/artikel/TypeScript-Microsofts-neues-typisiertes-JavaScript-1723407.html?seite=all> abgerufen

Scharwies, M. (2017). JavaScript/Entstehung und Standardisierung. Von
https://wiki.selfhtml.org/wiki/JavaScript/Entstehung_und_Standardisierung#Versionsgeschichte_von_JavaScript abgerufen

Scholz, F. (2017). block - JavaScript. Von <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/block> abgerufen

Severance, C. (2012). JavaScript: Designing a Language in 10 Days. Von
<https://www.computer.org/csdl/mags/co/2012/02/mco2012020007.html>
abgerufen am 29.12.2017

Shan, T., & Hua, W. (2006). Taxonomy of Java Web Application Frameworks. In *2006 IEEE International Conference on e-Business Engineering (ICEBE'06)* (S. 378–385). Washington DC: IEEE Computer Society.
<https://doi.org/10.1109/ICEBE.2006.98>

Springer, S. (2017, Mai). Modernes JavaScript Handgemacht. *iX Magazin*.

- Stefanov, S. (2017). *Durchstarten mit React: Web-Apps einfach und modular entwickeln* (1. Auflage). Heidelberg: dpunkt.verlag.
- Šuta, I. (2017). Tuples in TypeScript.
 Von <https://codingblast.com/typescript-tuples/> abgerufen
- Tarasiewicz, P., & Böhm, R. (2014). *AngularJS - Eine praktische Einführung in das JavaScript-Framework*. Heidelberg: dpunkt.verlag GmbH.
- Tchoukio, G. (2004). Aufbau einer Web-Anwendung (JSP o. ASP o. PHP), 1–6.
- Techopedia. (2018). What is Application Framework? Von <https://www.techopedia.com/definition/6005/application-framework> abgerufen
- Thattil, S. (2016). Die wichtigsten Technologien für die Entwicklung von Webanwendungen. Von <http://www.yuhiro.de/technologien-fuer-die-entwicklung-von-webanwendungen/> abgerufen
- Turner, J. (2014). TypeScript Design Goals. Von <https://github.com/Microsoft/TypeScript/wiki/TypeScript-Design-Goals> abgerufen
- Vogel, P. (2015). TypeScript – TypeScript-Grundlagen. Von <https://msdn.microsoft.com/de-de/magazine/dn890374.aspx> abgerufen
- Voutilainen, J.-P., Mikkonen, T., & Systä, K. (2016). Synchronizing Application State Using Virtual DOM Trees. In S. Casteleyn, P. Dolog, & C. Pautasso (Hrsg.), *Current Trends in Web Engineering: ICWE 2016 International Workshops, DUI, TELERISE, SoWeMine, and Liquid Web, Lugano, Switzerland, June 6-9, 2016. Revised Selected Papers* (S. 142-154). Cham: Springer International Publishing.
- Wagner, B. (2017). Enumerationstypen (C#-Programmierhandbuch). Von <https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/enumeration-types> abgerufen
- Woiwode, G., Malcher, F., Koppenhagen, D., & Hoppe, J. (2017). *Angular*. Heidelberg: Dpunkt.Verlag.

Zeigermann, O., & Hartmann, N. (2016). *React: Die praktische Einführung in React, React Router und Redux* (1. Auflage). Heidelberg: dpunkt.verlag.

Zuvic, D., & Roelofsen, R. (2016). Der pragmatische Ansatz von TypeScript.
Von <https://www.informatik-aktuell.de/entwicklung/programmiersprachen/der-pragmatische-ansatz-von-typescript.html> abgerufen