Angular Programmieren für Einsteiger

Der leichte Weg zum Angular-Experten

Sebastian Conrad

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Informationen sind im Internet über http://dnb.d-nb.de abrufbar.

©2020 BMU Media GmbH
www.bmu-verlag.de
info@bmu-verlag.de

Lektorat: Dr. Iris Seemann
Einbandgestaltung: Pro ebookcovers Angie
Druck und Bindung: Wydawnictwo Poligraf sp. zo.o. (Polen)

Taschenbuch-ISBN: 978-3-96645-105-5
Hardcover-ISBN:978-3-96645-050-8
E-Book-ISBN: 978-3-96645-042-3

Dieses Werk ist urheberrechtlich geschützt.
Alle Rechte (Übersetzung, Nachdruck und Vervielfältigung) vorbehalten. Kein Teil des Werks darf ohne schriftliche Genehmigung des Verlags in irgendeiner Form – auch nicht für Zwecke der Unterrichtsgestaltung- reproduziert, verarbeitet, vervielfältigt oder verbreitet werden.

Dieses Buch wurde mit größter Sorgfalt erstellt, ungeachtet dessen können weder Verlag noch Autor, Herausgeber oder Übersetzer für mögliche Fehler und deren Folgen eine juristische Verantwortung oder irgendeine Haftung übernehmen. Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären.

Angular Programmieren für Einsteiger

DANKSAGUNG

Sabrina & Carlotta

Ich danke euch, für eure Geduld und für die Zeit, die ich mir für dieses Buch nehmen durfte.

Ihr seid eine große Unterstützung gewesen. Ich liebe euch ...

Inhaltsverzeichnis

1. Schnellübersicht – Bausteine und Befehle 12

1.1 CLI ... 13
1.2 Angular-Template-Syntax ... 14
1.3 Lifecycle-Hooks ... 15
1.4 Pipes ... 16

2. Einleitung 18

2.1 An wen richtet sich das Buch? ... 18
2.2 Was sollten Sie mitbringen? ... 19
2.3 Ziel des Buches ... 20
2.4 Angular-Version ... 20
2.5 Vorgehen ... 20

3. Angular – Überblick 23

3.1 JavaScript und ECMAScript ... 24
3.1.1 ES6 ... 25
3.1.2 ES7 ... 26
3.1.3 ES8 ... 26
3.2 JavaScript-Frameworks ... 26
3.2.1 Bibliothek ... 26
3.2.2 Framework ... 26
3.3 AngularJS versus Angular ... 28
3.4 Angular ... 28
3.4.1 Versionierung ... 28
3.4.2 Vorteile ... 29
3.4.3 Nachteile ... 32

4. Werkzeuge 34

4.1 Entwicklungsumgebung ... 34
4.2 Konsole ... 35
4.2.1 Windows ... 35
4.2.2 Linux ... 37
4.3 Node.js ... 38
4.3.1 Installieren und einrichten – Windows ... 38
4.3.2 Installieren und einrichten – Linux ... 41
4.4 npm ... 42
4.5 Editor – IDE ... 44
4.5.1 Visual Studio Code ... 46
4.5.2 Atom ... 47
4.6 Plug-ins ... 49
4.6.1 Plug-in-Installation unter Visual Studio Code ... 49
4.6.2 Plug-in-Installation unter Atom ... 50
4.6.3 Empfehlungen ... 51
4.7 FTP-Client ... 52

5. CLI 56

5.1 Installation ... 57
5.2 Befehle ... 57

5.2.1 Projekt anlegen: new ... 58
5.2.2 Anwendung starten: serve ... 62
5.2.3 Dateien generieren: generate ... 63
5.2.4 Anwendung bauen: build ... 65
5.3 Verzeichnisstruktur der CLI ... 70
5.3.1 Workspace-Konfigurationsdateien ... 72
5.3.2 Projektdateien ... 73
5.3.3 Statische Daten einbinden ... 75
5.3.4 Externe Bibliotheken einbinden ... 77
5.4 Übungsaufgabe ... 78

6. Exkurs – TypeScript 82

6.1 Überblick ... 83
6.1.1 Transpilieren ... 83
6.1.2 Transpilieren versus Kompilieren ... 84
6.1.3 Nützliches Wissen ... 84
6.1.4 Installation ... 85
6.2 Identifier ... 88
6.3 var, let, const ... 89
6.3.1 var ... 90
6.3.2 let ... 90
6.3.3 const ... 91
6.4 Basistypen ... 91
6.4.1 number ... 92
6.4.2 string ... 92
6.4.3 boolean ... 93
6.4.4 Array ... 94
6.4.5 enum ... 94
6.4.6 null und undefined ... 95
6.4.7 any ... 95
6.4.8 void ... 95
6.5 Interfaces ... 96
6.5.1 Optionale Eigenschaften ... 97
6.5.2 Erweiterbare Interfaces ... 98
6.5.3 Interfaces für Funktionen ... 99
6.6 Export und Import ... 99
6.7 Klassen ... 100
6.7.1 Eigenschaften von Klassen ... 100
6.7.2 Methoden von Klassen ... 101
6.7.3 Statische Methoden ... 102
6.7.4 Interfaces für Klassen ... 103
6.7.5 Vererbung ... 105
6.7.6 Sichtbarkeit von Eigenschaften und Methoden ... 106
6.8 Generische Typdefinitionen ... 109
6.9 Übungsaufgaben ... 109
6.9.1 Typen deklarieren ... 109
6.9.2 Interfaces erstellen ... 110
6.9.3 Klassen erstellen ... 111
6.9.4 Klassen mit Vererbung ... 112
6.9.5 Generische Typen erstellen ... 113

7. Angular-Bausteine und -Abläufe 116

7.1 Dekoratoren ... 118
7.2 Bootstrap-Prozess ... 119
7.2.1 index.html ... 121
7.2.2 main.ts ... 121
7.3 Environment Variables (Umgebungsvariablen) ... 123

7.4 Projektstruktur 127
7.5 Beispielanwendung 128

8. Erste Anwendung – Hallo Welt 131

8.1 Neues Projekt anlegen 131
8.2 Hallo Welt 132
8.3 Anwendung starten 132
8.4 Browser starten 132
8.5 Anwendung anpassen 132

9. Die Beispielanwendung 134

9.1 Aufbau 134
9.2 Funktionalitäten 135
9.2.1 Produkte 135
9.2.2 Suche 137
9.2.3 Warenkorb 139
9.3 Projektgenerierung 141
9.4 Installation von Bibliotheken 142
9.4.1 Bootstrap 142
9.4.2 Font Awesome 143
9.5 Backend- und Mock-Daten 145
9.6 Starten der Beispielanwendung 147

10. Komponenten und Komponentenarchitektur 149

10.1 Komponentenbaum 152
10.2 Komponenten implementieren 152
10.2.1 Template 153
10.2.2 Style 154
10.2.3 Logik 155
10.3 Komponenten einbinden 157
10.4 Template-Syntax *{{}}[]() 157
10.4.1 Interpolation 158
10.4.2 Property-Binding 160
10.4.3 Interpolation versus Property-Binding 165
10.4.4 Event-Binding 165
10.4.5 Two-Way-Databinding 169
10.4.6 Safe-Operator 171
10.4.7 Elementreferenz 172
10.4.8 Beispiel: Alles in allem 175
10.5 Interaktion zwischen Komponenten 177
10.5.1 Eingehender Datenfluss 180
10.5.2 Ausgehender Datenfluss 181
10.5.3 Dumme und kluge Komponenten 183
10.5.4 Datenaustausch 184
10.6 Lebenszyklus einer Komponente 185
10.6.1 ngOnChanges 191
10.6.2 ngOnInit 192
10.6.3 ngDoCheck 193
10.6.4 ngAfterContentInit 193
10.6.5 ngAfterContentChecked 193
10.6.6 ngAfterViewInit 193
10.6.7 ngAfterViewChecked 193
10.6.8 ngOnDestroy 193
10.7 Auf Änderungen reagieren 194
10.7.1 Datenauswertung über OnChanges 194
10.7.2 Datenauswertung über Setter und Getter 195

10.7.3 Vor- und Nachteile 196
10.8 Beispielanwendung 197
10.9 Übungsaufgaben 200
10.9.1 Formular NewBookComponent 202
10.9.2 Liste BookListComponent 203
10.9.3 Vorgehen 204
10.9.4 Lösung 206

11. Direktiven 211

11.1 Komponente versus Direktive 211
11.2 Strukturdirektiven 211
11.2.1 NgIf-Direktive – Teile der Ansicht bedingt anzeigen 212
11.2.2 NgIf-Direktive – mit else-Block 212
11.2.3 NgSwitch-Direktive – Teile der Ansicht bedingt anzeigen 213
11.2.4 NgFor-Direktive – Listen anzeigen 215
11.2.5 NgFor-Direktive mit trackBy 217
11.3 Attributdirektiven 219
11.3.1 NgClass 219
11.3.2 NgStyle 220
11.4 Eigene Direktiven implementieren 222
11.5 Dekoratoren 223
11.5.1 HostBinding 223
11.5.2 HostListener 225
11.6 Beispielanwendung 227
11.7 Übungsaufgaben 229

12. Services 237

12.1 Injector und Dependency Injection 238
12.2 Service erstellen 240
12.3 Service registrieren 240
12.4 Service nutzen 242
12.5 Beispielanwendung 247
12.6 Übungsaufgabe 249

13. Module 253

13.1 Arten von Modulen 254
13.1.1 Hauptmodul 254
13.1.2 Routing-Modul 255
13.1.3 Shared-Modul 255
13.1.4 Feature-Modul 255
13.2 Modul-Aufbau 255
13.2.1 AppModule 255
13.2.2 Eigene Module 256
13.2.3 NgModule 257
13.3 Häufig genutzte Module 258
13.4 Modularisierung anhand der Beispielanwendung 259
13.4.1 Vorgehen 260
13.4.2 Umsetzung anhand des Moduls FeaturesModule 262
13.5 Übungsaufgaben 265

14. Observables 268

14.1 Observables erzeugen 270
14.2 Methoden 271
14.2.1 subscribe 272
14.2.2 unsubscribe 273

14.2.3 pipe ... 274
14.3 Operatoren ... 274
14.3.1 of ... 275
14.3.2 map ... 275
14.3.3 filter ... 276
14.3.4 catchError ... 277
14.4 Beispielanwendung ... 278
14.5 Übungsaufgabe ... 281

15. Navigation 286

15.1 Router ... 287
15.2 Routing einrichten ... 288
15.2.1 RouterOutlet – Inhalt platzieren ... 289
15.2.2 Routen einrichten ... 290
15.2.3 Verschachtelte Routen einrichten ... 291
15.2.4 RouterModule ... 293
15.3 Routing mit Parametern ... 296
15.4 Navigieren mit dem Router ... 299
15.4.1 Template ... 299
15.4.2 Klasse ... 300
15.5 Lazy Loading ... 301
15.6 Beispielanwendung ... 303
15.7 Übungsaufgaben ... 304

16. Benutzereingaben – Formulare 308

16.1 Template-Driven-Formulare ... 310
16.1.1 Implementierung ... 311
16.1.2 Validierungen ... 315
16.2 Reaktive Formulare ... 317
16.2.1 Implementierung ... 318
16.2.2 Validierung/Prüfungen ... 327
16.2.3 Datenänderungen ... 332
16.3 Beispielanwendung ... 334
16.4 Übungsaufgaben ... 337
16.4.1 Formular erstellen ... 337
16.4.2 Auf Änderungen reagieren ... 337

17. Pipes – Daten vor dem Rendern transformieren 343

17.1 Überblick ... 344
17.2 UpperCasePipe ... 344
17.3 LowerCasePipe ... 345
17.4 SlicePipe ... 345
17.5 JSON-Pipe ... 348
17.6 DecimalPipe ... 349
17.7 DatePipe ... 350
17.8 PercentPipe ... 352
17.9 CurrencyPipe ... 353
17.10 AsyncPipe ... 354
17.11 Eigene Pipes ... 355
17.12 Beispielanwendung ... 357
17.13 Übungsaufgaben ... 359

18. Kommunikation mit dem Server 362

18.1 HTTP-Methoden ... 362
18.2 Online-REST-API –PhotoAdmin ... 364
18.2.1 REST-API ... 367
18.2.2 Interface ... 368
18.2.3 Templategestaltung ... 369
18.2.4 Warum ein Service? ... 370
18.3 HTTPClient ... 371
18.4 Eigene Header ... 372
18.5 Daten abrufen ... 373
18.6 Daten anlegen ... 374
18.7 Daten aktualisieren ... 375
18.8 Daten löschen ... 376
18.9 Fehlerbehandlung ... 376
18.10 Übungsaufgaben ... 377
18.10.1 Interface ... 378
18.10.2 Template ... 378
18.10.3 Daten abrufen ... 380
18.10.4 Daten anlegen ... 381
18.10.5 Daten aktualisieren ... 382
18.10.6 Daten löschen ... 382

19. Debugging – Fehlersuche 384

19.1 Developer-Tools ... 384
19.1.1 Elements ... 386
19.1.2 Console ... 388
19.1.3 Network ... 391
19.2 Haltepunkte ... 394
19.2.1 Haltepunkte im Browser setzen ... 395
19.2.2 Haltepunkte im Editor setzen ... 399
19.3 Augury ... 405
19.4 Übungsaufgaben ... 407
19.4.1 Developer-Tools ... 407

20. Testen 410

20.1 Unit-Tests ... 410
20.2 Jasmine ... 413
20.2.1 Suite ... 413
20.2.2 Spezifikationen (Specs) ... 414
20.2.3 Expectations und Matcher ... 414
20.2.4 Setup und Teardown ... 415
20.2.5 TestBed ... 416
20.2.6 ComponentFixture ... 417
20.3 Komponenten testen ... 419
20.4 Services testen ... 423
20.5 Exkurs – Tests mit Protractor ... 426
20.6 Übungsaufgaben – Tests schreiben ... 428

21. Weiteres Wissenswertes 432

21.1 Webpack ... 432
21.2 TSLint ... 432
21.3 Angular-Styleguide ... 433
21.4 Change Detection ... 433
21.5 Sprachraum einstellen ... 434
21.6 CDK ... 435

21.6.1 Drag and Drop 435
21.6.2 Scrolling und Virtual Scrolling 436
21.6.3 Tree 437
21.7 Angular – Entwicklungsgeschichte 438
21.7.1 Support 438
21.7.2 Angular 2 438
21.7.3 Angular 4 439
21.7.4 Angular 5-6 439
21.7.5 Angular 7 439
21.7.6 Angular 8 439
21.7.7 Angular 9 439
21.8 Git 439

22. Glossar **444**

23. Index **446**

Alle Programmcodes aus diesem Buch sind über GitHub zum Download verfügbar. Dadurch müssen Sie sie nicht abtippen:
https://bmu-verlag.de/angular

Außerdem erhalten Sie die eBook Ausgabe zum Buch im PDF Format kostenlos auf unserer Website:

https://bmu-verlag.de/angular
Downloadcode: siehe Kapitel 21

Kapitel **1**

Schnellübersicht – Bausteine und Befehle

Baustein	**Beschreibung**
Komponente	Die Komponente ist der zentrale Baustein einer Angular-Webapplikation. Jede Komponente ist eine eigenständige und in sich geschlossene Einheit, die wiederverwendet werden kann. Eine Komponente hat ein *User Interface*, das im Browser dargestellt wird. Die dafür benötigte Struktur ist durch HTML-Elemente im `Template` abgelegt. Zusätzlich kann es noch eine separate Layout-Datei geben, die die Stylings für das `Template` liefert. Schließlich enthält die Klasse die Logik und das Verhalten der Komponente. Siehe genauer dazu Kapitel 10.
Direktive	Direktiven sind zur Steuerung von DOM-Elementen gedacht. Mit ihnen lassen sich HTML-Elemente sowie Angular-Komponenten erzeugen, verändern oder entfernen. Siehe genauer dazu Kapitel 11.
Pipe	Eine Pipe nimmt Daten als Eingabe entgegen und wandelt diese in eine gewünschte Ausgabe um. Siehe genauer dazu Kapitel 17.
Service	Ein Service ist der Ort für die Logik und Datenverwaltung. Siehe genauer dazu Kapitel 12.
Modul	Ein Modul ist eine Art Verwaltungsdatei. Es ermöglicht das Gliedern und Strukturieren einer Webapplikation in verschiedene Funktionsblöcke. Siehe genauer dazu Kapitel 13.
Interface	Mit Interfaces kann eine erwartete Objektstruktur mit dem jeweiligen Typ definiert werden. Siehe genauer dazu Kapitel 6.5.

Tabelle 1.1 Übersicht über die Angular-Bausteine

1.1 CLI

Befehl	Alias	Beschreibung
`ng new <name> [Optionen]`	`n`	Erstellt ein neues Projekt mit einer vordefinierten Verzeichnisstruktur.
`ng serve [Optionen]`	`s`	Baut die App und zeigt diese im Browser an.
`ng generate [Optionen]`	`g`	Erzeugt basierend auf den Angular-Schemas neue Dateien.
`ng build [Optionen]`	`b`	Baut die App und kopiert alle kompilierten Sourcen in den Ordner `/dist`.
`ng lint [Optionen]`	`l`	Führt eine statische Codeanalyse aus und zeigt eventuelle Schwachstellen im Code auf.
`ng test [Optionen]`	`t`	Führt Tests mit unterschiedlichen Optionen aus.
`ng update [Optionen]`		Ermöglicht die automatische Aktualisierung der Abhängigkeiten und des Angular-Frameworks.

Tabelle 1.2 Übersicht über die wichtigsten Angular-CLI-Befehle

Generator	Befehl	Kurzform
Komponente	`ng g component <name>`	`ng g c <name>`
Direktive	`ng g directive <name>`	`ng g d <name>`
Pipe	`ng g pipe <name>`	`ng g p <name>`
Service	`ng g service <name>`	`ng g s <name>`
Modul	`ng g module <name>`	`ng g m <name>`

Tabelle 1.3 Übersicht über die generate-Befehle

1.2 Angular-Template-Syntax

Syntax	Bezeichnung	Beschreibung
`{{ expression }}`	Interpolation	Zeigt Daten aus der Logik-Klasse im Template an.
`[target]="expression"` `bind-target="expression"`	Property-Binding	Bindet eine Eigenschaft an einen Ausdruck.
`(target)="statement"` `on-target="statement"`	Event-Binding	Bindet ein Angular-Ereignis an eine Aktion.
`[(target)]="expression"` `bindon-target="expression"`	Two-Way-Binding	Liest Eigenschaften und verarbeitet Ereignisse.
`*`	Strukturdirektiven	Direktiven, die den DOM-Baum manipulieren, siehe Kapitel 11.2.
`#`	Elementreferenz	Direktzugriff auf ein DOM-Element.

Tabelle 1.4 Übersicht über die Template-Syntax

1.3 Lifecycle-Hooks

Zur Übersicht über die Ablauffolge der Lebenszyklen einer Komponente (sogenannte Lifecycle-Hooks) eignet sich sehr gut die folgende Grafik aus Kapitel 10.6.

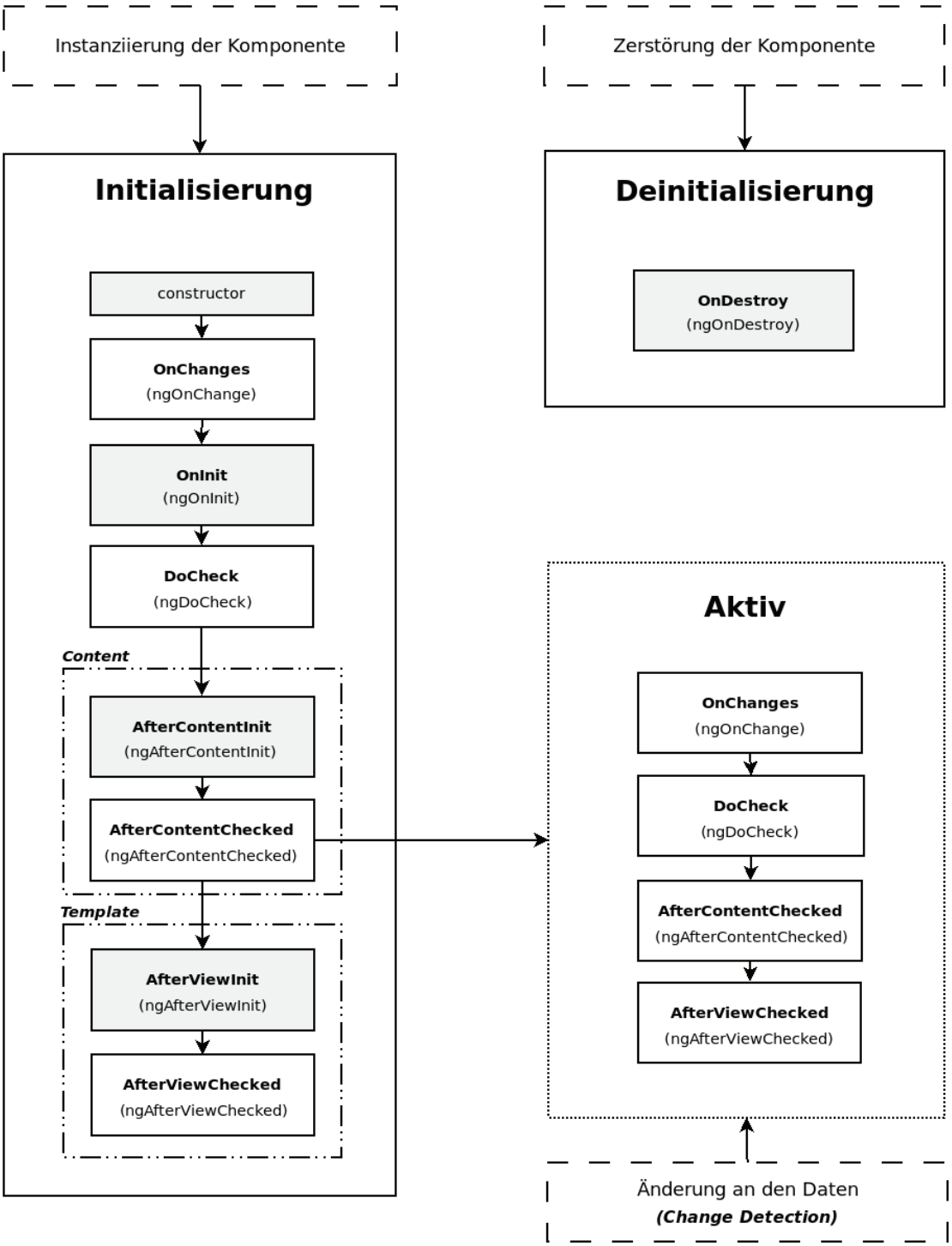

Abb. 1.1 Übersicht über die und Zuordnung der Lifecycle-Hooks

1.4 Pipes

Pipe	Beschreibung
`uppercase`	Transformiert den Text in Großbuchstaben, siehe Kapitel 17.2.
`lowercase`	Transformiert den Text in Kleinbuchstaben, siehe Kapitel 17.3.
`slice`	Erstellt ein neues Array oder eine neue Zeichenfolge, die eine Teilmenge der Elemente enthält, siehe Kapitel 17.4.
`json`	Gibt einen Wert in einer formatierte JSON-Darstellung aus, siehe Kapitel 17.5.
`decimal`	Transformiert eine Zahl in eine Zeichenfolge, die nach lokalen Regeln formatiert ist, siehe Kapitel 17.6.
`date`	Formatiert einen Datumswert nach den Regeln des Gebietsschemas, siehe Kapitel 17.7.
`percent`	Transformiert eine Zahl in eine prozentuale Zeichenfolge, die nach lokalen Regeln formatiert ist, siehe Kapitel 17.8.
`currency`	Wandelt eine Zahl in eine Währungszeichenfolge um, die nach lokalen Regeln formatiert ist, siehe Kapitel 17.9.
`async`	Zeigt die letzten Werte aus einem `Observable` oder aus einem `Promise` im Template an, siehe Kapitel 17.10.

Tabelle 1.5 Übersicht über die wichtigsten Pipes

Alle Programmcodes aus diesem Buch sind über GitHub zum Download verfügbar. Dadurch müssen Sie sie nicht abtippen:
https://bmu-verlag.de/angular

Außerdem erhalten Sie die eBook Ausgabe zum Buch im PDF Format kostenlos auf unserer Website:

https://bmu-verlag.de/angular
Downloadcode: siehe Kapitel 21

Kapitel 2

Einleitung

Noch vor zehn Jahren war es undenkbar, dass Unternehmen Teile ihrer Software auf JavaScript umstellen – JavaScript galt als viel zu fehleranfällig (Stichwort: keine Typisierung und keine automatischen Tests) und war als Skriptsprache verschrien. Aufgrund des unbändigen Drangs zur Digitalisierung ist JavaScript jedoch heute mit seinen zahlreichen Tools und vor allem mit dem Aufsatz TypeScript eine ernstzunehmende Technologie im Unternehmensumfeld geworden.

Angular spielt als JavaScript-Framework ganz vorne mit. Es handelt sich dabei um eine leistungsfähige Open-Source-Software aus dem Hause Google, die vorrangig zur Entwicklung von Webapplikationen beziehungsweise Single-Page-Applications (SPA) dient.

Das Angular-Framework zeigt seine Stärken in der standardisierten Modulpalette und den mitgelieferten Tools. Es unterstützt Entwickler in der Entwicklung selbst, bei der Fehlersuche, beim Testen, beim Bauen der Anwendung – und vor allem durch die einheitliche Struktur. Beim Wechsel von einem Angular-Projekt in ein anderes Angular-Projekt kann davon ausgegangen werden, dass Komponenten die standardisierten Angular-Module (wie das Navigations- oder HTTP-Modul) nutzen und dass die Bestandteile von Angular einheitlich strukturiert sind.

TypeScript als Erweiterung von JavaScript erleichtert die Entwicklung komplexer und stabiler Applikationen. Mit Angular werden komponentenbasierte Webanwendungen entwickelt, die in Bezug auf Apps und native Desktop-Anwendungen portierbar sind.

Dieses Buch ermöglicht Ihnen einen ersten Einstieg in Angular – mit praxisnahen Beispielen und nützlichen Tipps zur Umsetzung Ihres eigenen Angular-Projekts.

Ich wünsche Ihnen beim Lesen und Bearbeiten des Inhalts viel Spaß und Erfolg. Verzweifeln Sie nicht, einige Zusammenhänge und Konzepte versteht man oft erst im Detail, wenn man komplexe Anwendungen über einen längeren Zeitraum begleitet hat.

2.1 An wen richtet sich das Buch?

Das Buch richtet sich an alle, die Lust haben, mit Angular Projekte umzusetzen – ganz gleich, ob privat oder beruflich oder ob es um eine Kleinstanwendung wie eine Homepage mit zwei Komponenten oder um eine komplexe Unternehmenssoftware mit

über 120.000 Zeilen Code geht. Dabei spielt es keine Rolle, ob Sie schon Projekte mit AngularJS umgesetzt haben oder Neuling in der Webentwicklung sind. In jedem Fall ist es für das Verständnis des Buches von Vorteil, wenn Sie schon über Kenntnisse in HTML, CSS und JavaScript verfügen.

In diesem Buch wird ein praxisorientierter Ansatz verfolgt. Sie werden Schritt für Schritt anhand einer Beispielanwendung das Framework und den Tech-Stack kennenlernen und die Nutzung erproben.

2.2 Was sollten Sie mitbringen?

Angular vereint zahlreiche Webtechnologien, daher wäre es von Vorteil, wenn Sie, wie eingangs erwähnt, über folgende Kenntnisse verfügen:

HTML
Sie sollten die gängigsten HTML-Tags sowie das HTML-Grundgerüst kennen. Außerdem sollten Sie schon einmal etwas vom DOM-Baum[1] gehört haben.

CSS
Begriffe wie Selektor oder Eigenschaften (wie `display` oder `color`) sollten Ihnen bekannt sein.

JavaScript
Ein großer Vorteil sind auch grundlegende Programmierkenntnisse in JavaScript, wie der sichere Umgang mit Datentypen (zum Beispiel *string*, *integer* oder *Array*), das Erstellen von eigenen Funktionen, die Funktionalität von asynchronen Berechnungen mit Promises[2] sowie die Behandlung von Ereignissen (sogenanntes Event-Handling).

Der BMU Verlag bietet Ihnen eine große Auswahl an Einsteiger-Fachliteratur an, falls Sie hier schon merken, dass Wissenslücken vorhanden sind.

Was Sie nicht benötigen, sind Kenntnisse in Angular oder TypeScript selbst. Zudem müssen Sie sich vorab nicht mit den nötigen Tools und Entwicklungsumgebungen für die Entwicklung von Angular-Applikationen vertraut machen. Das nötige Wissen darüber wird Ihnen in diesem Buch nähergebracht.

[1] https://javascript.info/dom-nodes
[2] weitere Informationen unter https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_ Objects/Promise

2.3 Ziel des Buches

Dieses Buch ermöglicht den Einstieg in das Angular-Framework. Es werden viele Bausteine und Technologien im Detail vorgestellt. Die Leser werden am Ende in der Lage sein, eigene Projekte zu planen, zu entwickeln, Fehler zu erkennen und zu beseitigen sowie eine lauffähige Applikation zu erstellen.

Der Fokus liegt dabei auf der Entwicklung einer Webapplikation. Mit Angular ist es ebenfalls möglich, Desktop- oder App-Anwendungen zu entwickeln. Dieses Ziel wird hier jedoch nicht verfolgt und es wird auch nicht weiter darauf eingegangen.

Das Buch erhebt keinerlei Anspruch auf Vollständigkeit. Der Fokus wird auf die wichtigsten Bausteine, Zusammenhänge und Technologien gelegt, sodass Sie in der Lage sein werden, eine Angular-Anwendung zu verstehen und selbstständig zu entwickeln. Das Angular-Framework ist sehr komplex und es werden dazu viele externe Bibliotheken und Technologien angeboten. Je nach den Anforderungen eines Projekts macht es daher Sinn, sich weiter mit Technologien zu beschäftigen, die um Angular herum entstanden und gewachsen sind.

2.4 Angular-Version

Für die Beispiele und die Beispielanwendung `sebcon24` wurde die Angular-Version 8 verwendet. Sie sind jedoch nicht explizit auf diese Version festgelegt. In diesem Buch werden Techniken und Implementierungen vermittelt, die unabhängig von einer bestimmten Angular-Version sind. Die Bausteine von Angular sowie der Entwicklungsprozess sind seit der Angular-Version 4 im Großen und Ganzen identisch geblieben.

Kurz vor dem Erscheinen des Buches ist Angular-Version 9 erschienen. Diese Version bringt Features wie den Ivy-Compiler (siehe Kapitel 21.7.7) mit. Die Beispiele bauen zwar alle auf Angular 8 auf, sind aber mit und unter Angular 9 genauso kompatibel und lauffähig.

2.5 Vorgehen

Zum Buch gehört die Beispielanwendung `sebcon24`, die Schritt für Schritt in den einzelnen Kapiteln aufgebaut und erweitert wird – bis schließlich eine fertige abgespeckte Online-Handelsplattform entstanden ist.

Neben der Beispielanwendung `sebcon24` werden in den meisten Kapiteln die Methoden und Implementierungsvorgänge von zahlreichen Codebeispielen (`Examples`) begleitet. An den betreffenden Stellen wird jeweils ein Hinweis auf eine Downloadmöglichkeit oder einen Dateipfad des betroffenen Angular-Bausteins gegeben.

Am Dateipfad lässt sich erkennen, ob es sich um die Codebeispiele der `Examples` (Pfad beginnend mit `examples/src/app/...`) oder um die Beispielanwendung `sebcon24` (Pfad beginnend mit `sebcon24/src/app/...`) handelt.

Die Projekte können hier geklont[3] und heruntergeladen werden:
https://bmu-verlag.de/angular

Die Benennung von Variablen und Funktionen erfolgt im Buch durchgehend in der sogenannten camelCase-Notation beziehungsweise in der lowerCamelCase-Notation. Das bedeutet, dass jedes neue Wort mit einem Großbuchstaben begonnen wird. Die Wörter werden direkt aneinandergereiht, es werden keine Leerzeichen oder Unterstriche verwendet. Der erste Buchstabe des Gesamtworts beginnt mit einem kleinen Buchstaben. Das hat den Vorteil, dass aussagekräftige Bezeichnungen aus mehreren Wörtern zusammengesetzt werden können.

Beispiel: `meine erste Variable` wird in der Notation zu `meineErsteVariable`.

Die Benennung von Variablen, Funktionen und Angular-Bestandteilen erfolgt in Englisch, was in der Softwareentwicklung üblich ist. Im Browser werden in der gerenderten Ausgabe alle Info-Texte, Werte und Hinweise in Deutsch ausgegeben.

Beispiel: `meineErsteVariable` wird zu `myFirstVariable`.

Die Codebeispiele sollen bestimmte Implementierungsmöglichkeiten in den Kapiteln verdeutlichen. Dabei wurde aus Übersichtlichkeitsgründen auf Fehlerprüfungen verzichtet. Zum Beispiel wird bei Argumenten in Funktionen immer davon ausgegangen, dass die Werte vom richtigen Typ sind. Prüfungen auf `null` oder `undefined` entfallen ebenfalls.

[3] siehe Git-Befehle in Kapitel 21.8

Alle Programmcodes aus diesem Buch sind über GitHub zum Download verfügbar. Dadurch müssen Sie sie nicht abtippen:
https://bmu-verlag.de/angular

Außerdem erhalten Sie die eBook Ausgabe zum Buch im PDF Format kostenlos auf unserer Website:

https://bmu-verlag.de/angular
Downloadcode: siehe Kapitel 21

Kapitel 3

Angular – Überblick

Angular (Ängulah ausgesprochen) ist ein Framework von Google, das seit 2010 stetig weiterentwickelt wird und derzeit als Version 9 veröffentlicht ist. Das Framework hat die *MIT-Lizenz* und darf somit sowohl für den privaten als auch für den gewerblichen Gebrauch genutzt werden.

Das Framework wurde für die Entwicklung von Webapplikationen für den Desktop sowie für den mobilen Bereich konzipiert. Mit Angular wird komponentenbasiert entwickelt, das heißt, dass die geplante Applikation in wiederverwendbare Komponenten heruntergebrochen und in Module gebündelt wird. Dadurch erhalten Sie eine wart-, test- und erweiterbare Software.

Die Komponenten und anderen Bestandteile werden in TypeScript entwickelt. Die TypeScript-Dateien werden letztendlich in JavaScript-Code transpiliert. Der Standard für JavaScript wird durch die sogenannte ECMAScript Spezifikation vorgegeben.

Laut der Stack-Overflow-Umfrage von 2019 in Abbildung 3.1 wird Angular am zweithäufigsten von professionellen Entwicklern verwendet. React von Facebook holt seit Jahren stetig auf, Angular steht aber immer noch an der Spitze der Webframeworks. Was für die Entwicklung mit Angular spricht, wird in Kapitel 3.4.2 näher beschrieben.

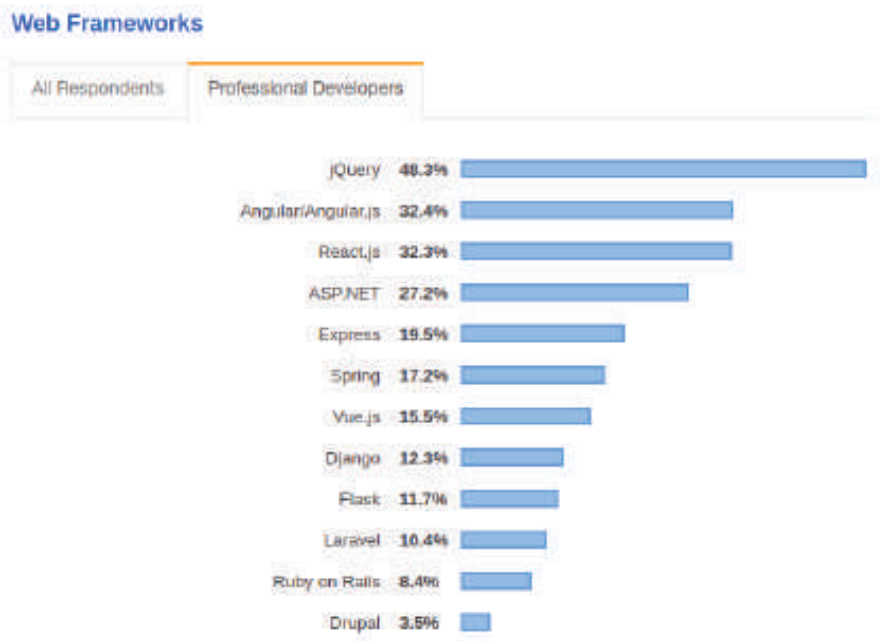

Abb. 3.1 Stack-Overflow-Umfrage zu Webframeworks 2019

3.1 JavaScript und ECMAScript

„JavaScript (kurz JS) ist eine Skriptsprache, die 1995 von Netscape für dynamisches HTML in Webbrowsern entwickelt wurde […]."[4] Heutzutage wird JavaScript auch auf Servern und in Mikrocontrollern eingesetzt.

ECMAScript (oder ES) ist eine von der Ecma International standardisierte Spezifikation für Skriptsprachen. Die Spezifikation ist festgehalten unter `ECMA-262` und `ISO/IEC 16262`. Sie wurde zur Standardisierung von JavaScript erstellt, um mehrere unabhängige Implementierungen zu fördern. In Tabelle 3.1 werden die Editionen mit ihrem jeweiligen Namen aufgelistet. Momentan ist ECMAScript 2019 beziehungsweise ES10 die aktuellste Version.

Edition	Veröffentlichung	Name
1	1997	
2	1998	
3	1999	
4	Abgebrochen	
5	2009	
5.1	2011	
6	2015	ECMAScript 2015 (ES2015)
7	2016	ECMAScript 2016 (ES2016)
8	2017	ECMAScript 2017 (ES2017)
9	2018	ECMAScript 2018 (ES2018)
10	2019	ECMAScript 2019 (ES2019)
	ES.Next	ES.Next[5]

Tabelle 3.1 Übersicht über die ECMAScript-Versionen, Stand: Januar 2020

Seit 2010 werden Standardtests für die ECMAScript-Spezifikation unter dem Namen Test262 entwickelt. Test262 ist eine „Konformitätstest-Test-Suite". Das Ziel dieser Tests ist die Prüfung, wie nah die JavaScript-Implementierung der jeweiligen ECMAScript-Spezifikation folgt beziehungsweise wie weit sie mit dem jeweiligen Standard übereinstimmt. Die Test-Suite enthält Tausende von individuellen Tests, die im offiziellen

[4] Seite „JavaScript", in: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 18. Mai 2020, 08:09 UTC. URL: https://de.wikipedia.org/w/index.php?title=JavaScript&oldid=200059690 (abgerufen am: 20. Mai 2020).

[5] ES.Next ist ein Platzhalter für die jetzige Version, welche gerade in Bearbeitung ist. ES.Next Features sind mehr Vorschläge und noch nicht finalisiert.

GitHub Repository[6] zu finden sind. Die Tests können online auf github.io[7] ausgeführt werden.

Die Tabelle 3.2 wurde aus den Daten der Homepage https://kangax.github.io/compat-table zusammengestellt. Die aktuellsten Browser mit ihren JavaScript-Engines wurden auf ihre Konformität in Bezug auf die einzelnen ECMAScript-Versionen getestet. Es zeigt sich, dass ES5 bei allen die Konformitätsrate von 100 % erreicht, die aufgelisteten Browser unterstützen demnach alle Features und Funktionen von ES5. Es zeigt sich aber auch, dass sogar die neusten Browser-Versionen nicht zu 100 % ES6 und höher abdecken.

Scripting Engine	**Browser**	**Versionen**		
		ES5	**ES6**	**Neuer (2016+)**
Chakra Microsoft	Edge 18	100 %	96 %	44 %
SpiderMonkey	Firefox 69	100 %	98 %	82 %
Chrome V8	Google Chrome 77, Opera 64	100 %	98 %	98 %
JavaScriptCore (Nitro)	Safari 13	99 %	99 %	86 %

Tabelle 3.2 Übersicht über ECMAScript – aktueller Browser-Support, Stand: Februar 2020

Im Folgenden werden die wichtigsten Features aufgelistet, entsprechend danach, in welcher ECMAScript-Version sie enthalten sind.

3.1.1 ES6

- Arrow-Funktionen
- Klassen und Vererbung
- Getter/Setter in Klassen
- Template-Strings
- let und const
- Promises

[6] https://github.com/tc39/test262
[7] https://v8.github.io/test262/website/default.html

- Modules und Module loaders

3.1.2 ES7

- Array.prototype.includes()
- Exponierungsoperator

3.1.3 ES8

- Async-Funktionen
- Shared Memory and Atomics

3.2 JavaScript-Frameworks

Was genau ist ein Framework? Und was ist der Unterschied zu einer Bibliothek? Oft werden diese beiden Begriffe synonym verwendet. Aber auch wenn der Übergang fließend ist, gibt es grundlegende Unterschiede zwischen den beiden Begriffen.

3.2.1 Bibliothek

Eine Bibliothek enthält abgestimmte Hilfsfunktionen, die einen bestimmten Teilaspekt in der Programmierung unterstützen sollen. Beispielsweise dient die JavaScript-Bibliothek `D3.js` der Datenvisualisierung – mit ihr lassen sich sowohl Diagramme und Statistiken als auch komplexe grafische Darstellungen wie Animationen umsetzen.

Bibliotheken werden explizit von einer Software eingebunden, die auf die entsprechenden Funktionen einer Programmbibliothek zugreift. Bibliotheken dienen der Programmierunterstützung und können nicht eigenständig ausgeführt werden.

3.2.2 Framework

Ein Framework (englisch für „Rahmenstruktur", „Ordnungsrahmen") ist eine spezielle Form einer Bibliothek. Es stellt die Softwarearchitektur einer Anwendung dar und bestimmt wesentlich den Entwicklungsprozess. Frameworks besitzen bestimmte Entwurfsmuster mit verschiedenen Funktionen und dienen der Entwicklung neuer eigenständiger Anwendungen. Zudem wird der Entwicklungsprozess durch Tools und weitere Bibliotheken unterstützt, wie bei Angular zum Beispiel durch die `CLI`[8] oder durch *Webpack*.

[8] siehe Kapitel 5

Im Folgenden werden die wichtigsten JavaScript-Frameworks neben Angular aufgelistet und kurz beschrieben, um einen kleinen Eindruck von den Unterschieden zu bekommen.

3.2.2.1 React

React fand erstmals 2011 im Newsfeed von Facebook Verwendung und wurde 2013 dann auf Open-Source-Basis veröffentlicht. Es handelt sich um eine weitere JavaScript-Bibliothek, mit der Benutzeroberflächen erstellt werden können. Das Besondere dabei ist, dass sich React nicht nur im Webseiten-Client einsetzen lässt, sondern auch auf dem Server oder bei der Appentwicklung zum Einsatz kommt. Das ist auf die Verwendung eines virtuellen DOMs zurückzuführen, wodurch auch das Testen von Webanwendungen einfacher wird. Darüber hinaus überzeugt die JavaScript-Bibliothek viele Entwickler durch `One-direction-data-flow`: Diese Technik sorgt für einen stabilen Code, indem Änderungen in hierarchisch tieferliegendem Code den höherstehenden nicht beeinflussen können. Nur in die andere Richtung können Änderungen Auswirkungen haben.

3.2.2.2 Ember.js

Auch Ember.js ist ein clientseitiges Framework, das für die Umsetzung von Single-Page-Webanwendungen eingesetzt wird und ebenfalls Desktopanwendungen erzeugen kann. Ein Unterscheidungsmerkmal zu Angular ist, dass die Macher von Ember.js die Community noch intensiver in die Entwicklungsprozesse des Frameworks miteinbeziehen und bedeutende Änderungen am Framework mit der Community öffentlich diskutieren, bevor sie umgesetzt werden. Ember.js wird als Framework zum Erstellen ambitionierter Webapplikationen vermarktet – dementsprechend richtet es sich in erster Linie an Entwickler, die bereits Erfahrung mit der Konzeption von Webapplikationen haben.

3.2.2.3 Vue.js

Auch bei Vue.js handelt es sich um ein JavaScript-Framework zur Entwicklung von Single-Page-Webanwendungen, das an Angular und React erinnert. Die Entwickler des relativ jungen und aufstrebenden Projekts haben Vue.js bewusst so gestaltet, dass gerade Anfängern der Einstieg vergleichsweise leicht fällt. So ist es zum Beispiel möglich, Templates in HTML einzubinden. Darüber hinaus soll Vue.js auch sehr viel flexibler sein als viele andere Frameworks, die für gewöhnlich die Art und Weise, wie mit dem Framework umzugehen ist, starr vorgeben.

3.3 AngularJS versus Angular

Die Version 1.x.x wird AngularJS genannt. Sie wird nur noch bedingt weiterentwickelt. Die aktuellste Version ist die Version 1.7.8 vom März 2019. 2016 wurde Angular, Version 2 veröffentlicht. Das Framework wird seit diesem Zeitpunkt nur noch Angular genannt und wurde von Grund auf neu geschrieben und konzipiert. Der große Unterschied zwischen den beiden Versionen ist, dass in AngularJS mit JavaScript und in Angular mit TypeScript entwickelt wird. Zudem erfolgt die Entwicklung in Angular komponentenbasiert und die komplette Projekt- und Codestruktur wurde darauf ausgelegt. Das ist auch der Hauptgrund, warum AngularJS-Anwendungen nicht kompatibel mit Angular sind. Eine Migration von AngularJS zu Angular ist möglich, aber bei komplexen Applikationen kann diese sehr umfangreich und zeitintensiv ausfallen. Das ist auch der Grund, warum noch viele Webapplikationen in AngularJS existieren.

3.4 Angular

In diesem Kapitel geht es um wissenswerte Fakten zur Versionierung, zur Entwicklungsgeschichte sowie zu den Vor- und Nachteilen von Angular.

3.4.1 Versionierung

Angular nutzt eine semantische Versionierung[9] , die sich aus drei Teilen zusammensetzt:

Major.Minor.Patch

Zum Beispiel gibt die Version 8.2.11 an, dass es sich um die Major-Version 8, die Minor-Version 2 und den Patch-Level 11 handelt.

HINWEIS In der Softwareentwicklung ist ein Release eine fertige und veröffentlichte Version einer Software. Damit verbunden ist die Erhöhung der Versionsnummer.

Ein Major-Release beinhaltet signifikante Änderungen und Features, die gegebenenfalls einen Zeitaufwand für das Refactoring, für zusätzliche Tests und für die API-Anpassungen[10] nach sich ziehen. Ein Minor-Release hingegen beinhaltet nur kleinere Features, die vollständig rückwärts kompatibel sind und keine zusätzlichen Anpas-

[9] siehe https://semver.org/

[10] (Abkürzung für Application Programming Interface) stellt eine wohldefinierte standardisierte Schnittstelle eines Programms zur Verfügung

sungen benötigen. Ein Patch-Release schließt kleinere Bugs ein, wobei auch hier keine zusätzlichen Anpassungen nötig sind.

3.4.2 Vorteile

Angular ist ein ideales Werkzeug, um komplexe Webanwendungen zu entwickeln. Das Framework bietet viele Vorteile, einige davon nimmt der Benutzer direkt gar nicht wahr, weil diese quasi „unter der Haube“ passieren. Im Folgenden werden die wichtigsten Vorteile kurz angerissen, um einen Eindruck der Stärken zu vermitteln.

3.4.2.1 Komponentenbasierte Entwicklung

Der Hauptvorteil wurde bereits an mehreren Stellen erwähnt: die komponentenbasierte Entwicklung. Hierbei werden wiederverwendbare, in sich geschlossene und testbare Komponenten entwickelt, die zusammengesetzt unter Umständen miteinander agieren und reagieren können. Im Detail wird diese Konzept in Kapitel 10 erläutert.

3.4.2.2 TypeScript

Auch wenn für einige Anfänger TypeScript eine Hürde beim Einstieg in die Webentwicklung darstellt, ist es im Endeffekt eine große Unterstützung, um sauberen, wartbaren und testbaren Code zu erzeugen. Gerade bei großen Projekten in Unternehmen mit einer großen Anzahl von Entwicklern macht sich die Einarbeitung in TypeScript bezahlt. Zumal es sich bei TypeScript, wie in Kapitel 6 erläutert wird, nicht um eine neue Programmiersprache handelt, sondern lediglich um ein SuperSet von JavaScript.

3.4.2.3 RxJS: effiziente und asynchrone Programmierung

Bei RxJS[11] handelt es sich um eine Bibliothek, die Funktionen für die asynchrone Datenverarbeitung bereitstellt. Es lassen sich damit Ereignisse unabhängig voneinander parallel behandeln und die Ausführung kann fortgesetzt werden, ohne auf das Eintreten eines Ereignisses warten zu müssen.

Die Bibliothek arbeitet mit Observables (siehe Kapitel 14), eine Art Blaupause, die beschreiben, wie Datenströme kombiniert werden und wie die Anwendung auf Variablen in diesen `Streams` reagiert. Sobald Sie die Observables begriffen haben, können Sie diese problemlos wiederverwenden, verwalten und als Lego-Blöcke kombinieren. Dies reduziert die Komplexität von Programmiervorgängen und das Verarbeiten großer Datenmengen in Blöcken.

[11] siehe https://angular.io/guide/rx-library, siehe https://rxjs-dev.firebaseapp.com/

3.4.2.4 Plattformunabhängigkeit

Angular wurde unter Berücksichtigung des Mobile-First-Ansatzes entwickelt. Die Idee dabei ist, die Codebasis und letztendlich die technischen Fähigkeiten über das Web sowie über iOS- und Android-Anwendungen zu teilen. Um diese ehrgeizige Positionierung zu verwirklichen, haben die Angular-Entwickler 2015 mit dem Team zusammengearbeitet, das das NativeScript-Framework[12] entwickelt hat.

3.4.2.5 Hohe Performance

Angular verwendet im Vergleich zu AngularJS eine verbesserte hierarchische Abhängigkeitsinjektion. Die Technik entkoppelt Komponenten von ihren Abhängigkeiten, indem diese parallel zueinander ausgeführt werden. Angular erstellt dabei einen separaten Baum von Abhängigkeitsinjektoren, die geändert werden können, ohne dass die Komponenten neu konfiguriert werden müssen.

In Angular wurden verschiedene Tools und Techniken implementiert, die zu einer Leistungssteigerung führen. Dazu zählen:

- Angular Universal
- Ivy Renderer
- Differential Loading

Angular Universal ist ein Dienst, der das Rendern der Anwendungsansicht auf einem Server anstelle von Client-Browsern ermöglicht. Google stellt eine Reihe von Tools zur Verfügung, mit denen Sie Ihre Anwendung vorab rendern oder für jede Anforderung eines Benutzers erneut rendern können. Derzeit ist das Toolset auf serverseitige Node.JS-Frameworks zugeschnitten und unterstützt ASP.NETCore.

Ivy Renderer: Angular-Komponenten und -Vorlagen werden in TypeScript und HTML geschrieben, das eigentliche HTML wird jedoch nicht direkt im Browser verwendet. Es ist ein zusätzlicher Schritt erforderlich, wenn HTML und TypeScript in JavaScript-Anweisungen interpretiert werden. Ein Renderer ist eine Engine, die Vorlagen und Komponenten in JavaScript und HTML übersetzt, die Browser verstehen und anzeigen können. Ivy ist, nach dem ursprünglichen Compiler und Renderer 2, die dritte Iteration des Angular-Renderers.

Differential Loading: Das Differential Loading beziehungsweise das differentielle Laden ist eine Möglichkeit, Inhalte zu laden und die Bündelgröße zu optimieren. Tatsächlich lassen sich zwei verschiedene Pakete für ältere und für neue Browser erstellen. Angular verwendet die neueste Syntax und Polyfills für die neueren Browser und erstellt ein separates Paket mit stabiler Syntax für ältere Browser. Auf diese Weise

[12] https://www.nativescript.org/

reduziert das Differential Loading die Paketgröße und damit die Ladegeschwindigkeit für die entsprechenden Browser.

3.4.2.6 Lang-Zeit-Support

Ein großer Vorteil ist natürlich, dass ein Konzern wie Google mit seinem Know-how dahintersteht. Der Name Google allein reicht jedoch nicht aus. Deshalb zeigt die Einführung von einem Lang-Zeit-Support (LTS), dass Google daran interessiert ist, Unternehmen mit deren Webanwendungen auch langfristig zu unterstützen und mit Updates zu versorgen.

3.4.2.7 Material Design von Angular

Das Angular-Team hat sein Framework mit Material-Design-Komponenten aktualisiert. Mit Angular-Material[13] erhalten Sie nach Designrichtlinien vorgefertigte und konsistente Komponenten wie Formularsteuerelemente, Navigationselemente, Layouts, Schaltflächen und Indikatoren, Pop-ups, modale Fenster oder auch Datentabellen. Die Komponenten sind an Angular angepasst und lassen sich mühelos in die Projekte integrieren.

3.4.2.8 Updates mit der Angular-CLI

Seit der Version 6 ist der Befehl `ng update` beziehungsweise `ng update <package>` integriert. Dieser überprüft das angegebene Paket und gibt Empfehlungen für Updates, die möglicherweise für alle Abhängigkeiten erforderlich sind, einschließlich Loader und Plug-ins. Mit `ng update @angular/cli@angular/core` wird Angular selbst aktualisiert und bietet Unterstützung bei der Migration auf eine neue Version.

3.4.2.9 Leistungsstarkes Ökosystem

Angular ist schon seit 2010 am Markt. Daher gibt es eine große Community, die Plugins, Pakete und Bibliotheken bereitstellt.

3.4.2.10 Angular-Elemente

Beim Ausführen von mehreren Projekten, von denen einige nicht mit Angular umgesetzt sind, lassen sich Angular-Elemente wiederverwenden und in anderen Frameworks nutzen, in dem diese als DOM-Element „gewrappt“ werden. Das funktioniert zum Beispiel in Vue.js, React oder JQuery. Diese Funktion ist sehr praktisch, wenn zwischen verschiedenen Frameworks gewechselt wird.

[13] https://material.angular.io/

3.4.3 Nachteile

Wie jedes Framework hat auch Angular einige Nachteile, die hier nicht ungenannt bleiben sollen.

3.4.3.1 Migration

Wie schon erwähnt, gibt es einen großen Unterschied zwischen AngularJS und Angular. Eine Migration[14] ist daher nicht einfach und kann sehr zeitaufwendig sein. Vor allem bei komplexen Anwendungen kann die Umstellung ein langer und mühsamer Weg sein.

3.4.3.2 Komplexität

Obwohl die komponentenbasierte Architektur als Vorteil von Angular erwähnt wurde, ist die Art und Weise, wie Komponenten verwaltet werden, sehr komplex. Beispielsweise werden je nach Entwicklungsart bis zu fünf Dateien für eine einzelne Komponente benötigt, es müssen Abhängigkeiten eingefügt und die Komponenten in Modulen hinzugefügt werden. Infolgedessen wird ein Großteil der Entwicklungszeit in Angular für sich wiederholende Aufgaben aufgewendet.

3.4.3.3 Steile Lernkurve

Das Spektrum der zu behandelnden Themen in Angular ist groß: Module, Abhängigkeitsinjektion, Komponenten, Services und Templates. Dies kann am Anfang sehr viel sein und erst einmal etwas erschlagend wirken. Eine weitere Barriere kann RxJS darstellen, eine reaktive Programmierbibliothek für die asynchrone Programmierung. Das Erlernen ist für die Verwendung von Angular zumindest in der Grundstufe obligatorisch. Vor allem gibt es aber noch Beschwerden über die kryptischen, oftmals nichtssagenden Fehlermeldungen.

3.4.3.4 CLI-Dokumentation

Die CLI-Dokumentation war bis zur siebten Angular-Version nicht auf dem aktuellsten Stand und in einigen Bereichen ungenügend. Mit dem Angular 7-Update wurde die CLI-Dokumentation jedoch mit Verweislinks und Richtliniendetails aktualisiert.

---

[14] https://angular.io/guide/upgrade

Alle Programmcodes aus diesem Buch sind über GitHub zum Download verfügbar. Dadurch müssen Sie sie nicht abtippen:
https://bmu-verlag.de/angular

Außerdem erhalten Sie die eBook Ausgabe zum Buch im PDF Format kostenlos auf unserer Website:

https://bmu-verlag.de/angular
Downloadcode: siehe Kapitel 21