

Inhalt

Geleitwort des Fachgutachters zur zweiten Auflage	25
Einleitung	27

1 Angular-Kickstart: Ihre erste Angular-Webapplikation 29

1.1 Installation der benötigten Software	29
1.1.1 Node.js und npm	29
1.1.2 Visual Studio Code – eine kostenlose Entwicklungsumgebung für TypeScript und Angular	30
1.1.3 Alternative: Webstorm – perfekte Angular-Unterstützung	30
1.2 Hallo Angular	31
1.2.1 Komponenten konfigurieren	34
1.2.2 Die Komponenten-Klasse	36
1.2.3 Das Applikationsmodul: das Hauptmodul der Anwendung konfigurieren	37
1.2.4 main.ts: Wahl der Ausführungsplattform und Start des Applikationsmoduls	39
1.2.5 index.html – Einbinden der Bootstrap-Komponente und Start der Anwendung	40
1.3 Die Blogging-Anwendung	41
1.3.1 Start der Applikation	45
1.3.2 Einige Tipps zur Fehlersuche	45
1.3.3 Die Formulkomponente: Daten aus der View in den Controller übertragen	46
1.3.4 Das Applikationsmodell	49
1.3.5 Darstellung der Liste in der View	52
1.3.6 Modularisierung der Anwendung	54
1.4 Zusammenfassung und Ausblick	56

2 Das Angular-CLI: professionelle Projektorganisation für Angular-Projekte 59

2.1 Das Angular-CLI installieren	60
---	----

2.2	ng new: ein Grundgerüst für die Applikation erstellen	60
2.2.1	Konfigurationsoptionen für die Projekt-Generierung	63
2.2.2	Das generierte Projekt im Detail	64
2.3	ng serve: die Anwendung starten	67
2.3.1	Die Proxy-Konfiguration	68
2.3.2	ng serve-Default-Optionen über die angular.json einstellen	69
2.4	npm start: Start über die lokale CLI-Version	71
2.5	ng generate: Komponenten generieren	72
2.5.1	Konfigurationsoptionen bei der Komponentengenerierung	74
2.5.2	Weitere Generatoren	75
2.6	ng update: Angular und weitere Abhängigkeiten auf die neueste Version updaten	76
2.7	ng lint: Linting und der Angular-Style-Guide	78
2.8	Komponenten- und Ende-zu-Ende-Tests ausführen	80
2.8.1	ng test – Unit- und Komponententests ausführen	80
2.8.2	ng e2e – Ende-zu-Ende-Tests ausführen	82
2.9	CSS-Präprozessoren verwenden	83
2.10	Drittanbieter-Bibliotheken einbinden	84
2.10.1	Bibliotheken über die index.html einbinden	85
2.11	ng add: Angular-spezifische Abhängigkeiten zu Ihrer Anwendung hinzufügen	86
2.12	ng build: deploybare Builds erstellen	88
2.12.1	Konfigurationsoptionen für die Ausführung des Builds	88
2.12.2	Produktions-Builds mit dem Angular-CLI erzeugen	89
2.13	Configurations: Konfiguration unterschiedlicher Build- und Ausführungsumgebungen	90
2.13.1	File-Replacements: Dateien abhängig von der Konfiguration austauschen	91
2.13.2	Eigene Build-Konfigurationen anlegen und aktivieren	92
2.13.3	Konfigurationen für den ng serve-Befehl	94
2.14	Der AOT-Modus	96
2.15	Zusammenfassung und Ausblick	97

3 Komponenten und Templating: der Angular-Sprachkern

99

3.1	Etwas Theorie: der Angular-Komponentenbaum	99
3.2	Selektoren: vom DOM-Element zur Angular-Komponente	103
3.2.1	Tag-Selektoren	103
3.2.2	Attribut-Selektoren	104
3.2.3	Klassen-Selektoren	105
3.2.4	not()-Selektoren	105
3.2.5	Verknüpfung von Selektoren	105
3.3	Die Templating-Syntax: Verbindung zwischen Applikationslogik und Darstellung	106
3.3.1	Fallbeispiel: Timepicker-Komponente	106
3.3.2	Property-Bindings	107
3.3.3	Sonderfälle: Attribute, Klassen und Styles setzen	110
3.3.4	Interpolation – Darstellung von Werten im View	112
3.3.5	Event-Bindings	113
3.3.6	Two-Way-Data-Bindings mit NgModel	118
3.3.7	Lokale Template-Variablen	120
3.3.8	Die *-Templating-Microsyntax – neue DOM-Elemente dynamisch einfügen	121
3.3.9	Templating-Syntax-Spickzettel	125
3.4	Komponentenschnittstellen definieren: von der einzelnen Komponente zur vollständigen Applikation	126
3.4.1	Input-Bindings – Werte in Ihre Komponenten hineinreichen	127
3.4.2	Output-Bindings – andere Komponenten über Datenänderungen informieren	131
3.4.3	Two-Way-Data-Bindings – syntaktischer Zucker für Ihre Komponentenschnittstelle	134
3.4.4	Auf Änderungen von Bindings reagieren	135
3.4.5	Lokale Komponentenvariablen – Zugriff auf die API Ihrer Kind-Elemente im HTML-Code	137
3.5	ViewChildren: Zugriff auf Kind-Elemente aus der Komponentenklasse ...	137
3.6	Content-Insertion: dynamische Komponentenhierarchien erstellen	140
3.6.1	Einfachen HTML-Code injizieren	140
3.6.2	ContentChildren: Erzeugung von dynamischen Komponenten- bäumen am Beispiel einer Tabs-Komponente	146
3.7	Der Lebenszyklus einer Komponente	150
3.7.1	Der Konstruktor: Instanziierung der Komponente	153

3.7.2	ngOnInit – Initialisierung der eigenen Komponente	154
3.7.3	ngOnChanges – auf Änderungen reagieren	155
3.7.4	ngAfterContentInit – auf die Initialisierung von Content-Children reagieren	156
3.7.5	ngAfterViewInit – auf die Initialisierung von ViewChildren reagieren	157
3.7.6	ngOnDestroy – Aufräumarbeiten vornehmen	157
3.7.7	ngAfterContentChecked, ngAfterViewChecked – den ChangeDetection-Mechanismus verfolgen	158
3.7.8	ngDoCheck – den ChangeDetection-Mechanismus verändern	160
3.8	Zusammenfassung und Ausblick	161

4 Direktiven: Komponenten ohne eigenes Template 163

4.1	ElementRef und Renderer2: Manipulation von DOM-Eigenschaften eines Elements	164
4.1.1	Die Renderer2-Klasse: das native Element plattform- unabhängig manipulieren	167
4.2	HostBinding und HostListener: Auslesen und Verändern von Host-Eigenschaften und -Events	168
4.2.1	Kanonisches Host-Binding	170
4.3	Anwendungsfall: Einbinden von Drittanbieter-Bibliotheken	171
4.3.1	Two-Way-Data-Binding für die Slider-Komponente	174
4.4	Anwendungsfall: Accordion-Direktive – mehrere Kind-Komponenten steuern	176
4.5	exportAs: Zugriff auf die Schnittstelle einer Direktive	179
4.6	Zusammenfassung und Ausblick	181

5 Fortgeschrittene Komponentenkonzepte 183

5.1	Styling von Angular-Komponenten	183
5.1.1	Styles an der Komponente definieren	184
5.1.2	ViewEncapsulation – Strategien zum Kapseln Ihrer Styles	185

5.2	TemplateRef und NgTemplateOutlet: dynamisches Austauschen von Komponenten-Templates	195
5.2.1	NgFor mit angepassten Templates verwenden	196
5.2.2	NgTemplateOutlet: zusätzliche Templates an die Komponente übergeben	198
5.3	ViewContainerRef und ComponentFactory: Komponenten zur Laufzeit hinzufügen	202
5.3.1	ViewContainerRef und entryComponents: Komponenten zur Laufzeit hinzufügen	203
5.3.2	ComponentRef: Interaktion mit der dynamisch erzeugten Komponente	206
5.3.3	Komponenten an einer bestimmten Stelle einfügen	207
5.3.4	Komponenten innerhalb des ViewContainers verschieben und löschen	208
5.3.5	createEmbeddedView: Templates dynamisch einbinden	209
5.4	NgComponentOutlet: dynamisch erzeugte Komponenten noch einfacher verwalten	213
5.4.1	Übergabe von dynamischen Eigenschaften an NgComponentOutlet	215
5.5	ChangeDetection-Strategien: Performance-Boost für Ihre Applikation	218
5.5.1	Die Beispielapplikation	218
5.5.2	Veränderungen des Applikationsstatus	221
5.5.3	ChangeDetection-Strategien: Optimierung des Standardverhaltens	224
5.5.4	ChangeDetectorRef: die vollständige Kontrolle über den ChangeDetector	227
5.6	Zusammenfassung und Ausblick	230
6	Standarddirektiven und Pipes: wissen, was das Framework an Bord hat	233

6.1	Standarddirektiven	234
6.1.1	NgIf: Elemente abhängig von Bedingungen darstellen	235
6.1.2	NgSwitch: Switch-Case-Verhalten implementieren	236
6.1.3	NgClass: CSS-Klassen dynamisch hinzufügen und entfernen	237
6.1.4	NgStyle: das style-Attribut manipulieren	241

6.1.5	NgFor: Komfortabel über Listen iterieren	242
6.1.6	NgNonBindable-Auswertung durch die Templating-Syntax verhindern	246
6.2	Pipes: Werte vor dem Rendern transformieren	247
6.2.1	UpperCasePipe und LowerCasePipe: Strings transformieren	248
6.2.2	Die SlicePipe: nur bestimmte Bereiche von Arrays und Strings darstellen	248
6.2.3	Die JSON-Pipe: JavaScript-Objekte als String ausgeben	251
6.2.4	KeyValuePipe: über Objekte und Maps iterieren	252
6.2.5	DecimalPipe: Zahlenwerte formatieren	254
6.2.6	Kurzexkurs: lokalisierbare Pipes – Werte der aktuellen Sprache entsprechend formatieren	255
6.2.7	DatePipe: Datums- und Zeitwerte darstellen	257
6.2.8	Percent- und CurrencyPipe: Prozent- und Währungswerte formatieren	259
6.2.9	Die AsyncPipe: auf asynchrone Werte warten	260
6.2.10	Pipes im Komponentencode verwenden	263
6.2.11	Eigene Pipes implementieren	264
6.2.12	Pure vs. Impure Pipes: Pipe, ändere dich!	268
6.3	Zusammenfassung und Ausblick	271

7 Services und Dependency-Injection: lose Kopplung für Ihre Business-Logik 273

7.1	Grundlagen der Dependency-Injection	274
7.2	Services in Angular-Applikationen	276
7.3	Das Angular-Dependency-Injection-Framework	277
7.3.1	Injector- und Provider-Konfiguration: das Herz der DI	278
7.3.2	Vereinfachungen bei der Provider-Definition	280
7.4	Injection by Type: Vereinfachungen für TypeScript-Nutzer	281
7.4.1	Den @Inject-Decorator vermeiden	281
7.4.2	Der @Injectable-Decorator: TypeScript-optimierte Injektion in Services	282
7.4.3	Member-Injection – automatische Erzeugung von Membervariablen	283
7.5	Weitere Provider-Formen	284
7.5.1	Injection-Tokens: kollisionsfreie Definition von DI-Schlüsseln	287

7.6	Der hierarchische Injector-Baum: volle Flexibilität bei der Definition Ihrer Abhängigkeiten	288
7.6.1	Der Injector-Baum	288
7.6.2	Registrierung von globalen Services: der UserService	290
7.6.3	Registrieren von komponentenbezogenen Services: MusicSearchService und VideoSearchService	293
7.7	TreeShakable-Providers: der DI-Mechanismus auf den Kopf gestellt	297
7.8	Sichtbarkeit und Lookup von Dependencies	298
7.8.1	Sichtbarkeit von Providern beschränken	298
7.8.2	Den Lookup von Abhängigkeiten beeinflussen	301
7.9	Zusammenfassung und Ausblick	305

8 Template-driven Forms: einfache Formulare auf Basis von HTML 307

8.1	Grundlagen zu Formularen: template-driven oder reaktiv?	308
8.2	Das erste Formular: Übersicht über die Forms-API	309
8.2.1	Einbinden des Formular-Moduls	309
8.2.2	Implementierung des ersten Formular-Prototyps	310
8.2.3	NgModel, NgForm, FormControl und FormGroup: die wichtigsten Bestandteile der Forms-API	314
8.3	NgModel im Detail: Two-Way-Data-Binding oder nicht?	315
8.3.1	One-Way-Binding mit NgModel	315
8.4	Kurzexkurs: Verwendung von Interfaces für die Definition des Applikationsmodells	319
8.5	Weitere Eingabelemente	322
8.5.1	Auswahllisten	322
8.5.2	Checkboxes	326
8.5.3	Radio-Buttons	327
8.6	Verschachtelte Eigenschaften definieren	328
8.6.1	Verschachtelte Eigenschaften mit NgModelGroup	328
8.7	Validierungen	330
8.7.1	Vom Framework mitgelieferte Validierungsregeln	330
8.7.2	Validierungen im Formular darstellen	331
8.7.3	Implementierung einer generischen ShowError-Komponente	333

8.7.4	Eigene Validierungsregeln definieren	338
8.7.5	Asynchrone Validierungen	340
8.7.6	Feldübergreifende Validierungen	344
8.8	Implementierung der Tags-Liste: wiederholbare Strukturen mit Template-driven Forms	346
8.9	updateOn: steuern, wann Änderungen übernommen werden	350
8.10	Zusammenfassung und Ausblick	351

9 Reactive Forms: Formulare dynamisch in der Applikationslogik definieren 353

9.1	Aktivierung von Reactive Forms für Ihre Applikation	354
9.2	Das Task-Formular im reaktiven Ansatz	354
9.2.1	Definition des Formulars im TypeScript-Code	355
9.2.2	Verknüpfung des Formulars mit dem HTML-Code	356
9.2.3	FormArray im Detail: wiederholbare Strukturen definieren	358
9.2.4	Verbindung des Formulars mit dem Applikationsmodell	362
9.2.5	Der FormBuilder – komfortable Definition von Formularen	366
9.2.6	Validierungen von Reactive Forms	367
9.2.7	updateOn in Reactive Forms	375
9.3	Formulare und Kontrollelemente auf Änderungen überwachen	376
9.4	Fallbeispiel: Umfragebogen – Formulare komplett dynamisch definieren	377
9.5	ControlValueAccessor: eigene Eingabeelemente für die Forms-API implementieren	384
9.5.1	Das neue Eingabeelement bei der Forms-API registrieren	387
9.5.2	Verwendung des neuen Eingabeelements in Formularen	388
9.6	Die Forms-API im Überblick	390
9.6.1	AbstractControl: die Basis für alle Forms-API-Basisklassen	391
9.6.2	FormControl: Eigenschaften und Methoden für einzelne Kontrollelemente	393
9.6.3	FormGroup: API zur Verwaltung von Gruppen und Formularen	393
9.6.4	FormArray: wiederholbare Strukturen managen	393
9.7	Zusammenfassung und Ausblick	394

10 Routing: Navigation innerhalb der Anwendung 397

10.1 Project-Manager: die Beispielanwendung	398
10.2 Die erste Routenkonfiguration: das Routing-Framework einrichten	399
10.3 Location-Strategien:	
»schöne URLs« vs. »Routing ohne Server-Konfiguration«	404
10.3.1 PathLocation-Strategie – schöne URLs	405
10.3.2 HashLocation-Strategie – Routing ohne aufwendige Konfiguration	406
10.4 ChildRoutes: verschachtelte Routenkonfigurationen erstellen	407
10.4.1 Componentless-Routes: Routendefinitionen ohne eigene Komponente	410
10.4.2 Relative Links	412
10.5 RouterLinkActive: Styling des aktiven Links	413
10.5.1 RouterLinkActiveOptions: Exakt oder nicht?	414
10.6 Routing-Parameter: dynamische Adresszeilenparameter auswerten	415
10.6.1 Pfad-Parameter: Pflicht-Parameter in Routen definieren	416
10.6.2 Snapshots – statisch auf Parameterwerte zugreifen	419
10.6.3 Matrix-Parameter: optionale Parameter	419
10.6.4 Query-Parameter: optionale Parameter unabhängig vom Segment definieren	423
10.6.5 Fragmentbezeichner	424
10.7 Aus der Anwendungslogik heraus navigieren	426
10.7.1 Die navigate-Methode: Navigation auf Basis der Routing-DSL	427
10.7.2 navigateByUrl: Navigation auf Basis von URLs	428
10.8 Routing-Guards: Routen absichern und die Navigation generisch beeinflussen	428
10.8.1 CanActivate – Routen absichern	429
10.8.2 CanDeactivate – das Verlassen einer Route verhindern	432
10.9 Redirects und Wildcard-URLs	434
10.9.1 Absolute Redirects	434
10.9.2 Relative Redirects	435
10.9.3 Wildcard-URLs – Platzhalter-Routen definieren	436
10.10 Data: statische Metadaten an Routen hinterlegen	436
10.11 Resolve: dynamische Daten über den Router injizieren	437
10.11.1 Verwendung einer resolve-Funktion anstelle einer Resolver-Klasse	439

10.12 Der Title-Service: den Seitentitel verändern	441
10.13 Router-Tree und Router-Events:	
generisch auf Seitenwechsel reagieren	442
10.13.1 Der events-Stream: bei Seitenwechseln informiert werden	442
10.13.2 Der Router-Tree: den aktuellen Router-Baum durchlaufen	443
10.14 Location: direkte Interaktion mit der Adresszeile des Browsers	445
10.15 Mehrere RouterOutlets: maximale Flexibilität beim Routing	448
10.15.1 Zusätzliche Outlets – ein Chat-Fenster einblenden	448
10.15.2 Komplexere Outlet-Konfigurationen: eine Task-Schnellansicht	451
10.16 Zusammenfassung und Ausblick	454

11 HTTP: Anbindung von Angular-Applikationen an einen Webserver 455

11.1 Die Server-Applikation	456
11.1.1 Die json-server-Bibliothek	457
11.2 Das Angular-HTTP-Modul verwenden	460
11.3 Der erste GET-Request: Grundlagen zur HTTP-API	460
11.3.1 Auf Fehler reagieren	462
11.4 Asynchrone Service-Schnittstellen modellieren:	
Anpassung des TaskService	464
11.4.1 Observables statt Callbacks – Daten reaktiv verwalten	464
11.5 Die AsyncPipe: noch eleganter mit asynchronen Daten arbeiten	466
11.6 HttpParams: elegant dynamische Suchen definieren	467
11.7 Die observe-Eigenschaft: die komplette HttpResponse auswerten	470
11.8 POST, PUT, DELETE, PATCH und HEAD:	
Verwendung der weiteren HTTP-Methoden	471
11.8.1 HTTP-POST: neue Tasks anlegen	471
11.8.2 HTTP-PUT: bestehende Tasks editieren	473
11.8.3 HTTP-DELETE: Tasks löschen	474
11.8.4 Generische Anfragen: die »request«-Methode	476
11.8.5 HTTP-PATCH: Tasks partiell verändern	477
11.8.6 HTTP-HEAD: der kleine Bruder von GET	478

11.9 JSONP	479
11.9.1 Die jsonp-Methode	481
11.10 Zusammenfassung und Ausblick	483

12 Reaktive Architekturen mit RxJS 485

12.1 Kurzeinführung in RxJS	486
12.1.1 Observable.create und Observer-Functions – die Kernelemente der reaktiven Programmierung	486
12.1.2 Subscriptions und Disposing-Functions – Observables sauber beenden	488
12.1.3 Subjects: Multicast-Funktionalität auf Basis von RxJS	491
12.2 Implementierung einer Typeahead-Suche	493
12.2.1 mergeMap: verschachtelte Observables verbinden	497
12.2.2 switchMap – nur die aktuellsten Ergebnisse verarbeiten	498
12.2.3 merge – mehrere Streams vereinen	499
12.3 Reaktive Datenarchitekturen in Angular-Applikationen	502
12.3.1 Shared Services – der erste Schritt in die richtige Richtung	504
12.3.2 Die neue Datenarchitektur: »Push« statt »Pull«	507
12.3.3 Umsetzung des neuen Konzepts in Angular	509
12.3.4 Anbindung der TaskListComponent an den Store	516
12.3.5 Der »In Bearbeitung«-Zähler	517
12.4 Anbindung von Websockets zur Implementierung einer Echtzeitanwendung	519
12.4.1 Der WebSocket-Server	519
12.4.2 Integration von Socket.IO in die Anwendung	521
12.4.3 Verwendung von Socket.IO im TaskService	522
12.5 ChangeDetectionStrategy.OnPush: Performance-Schub durch die reaktive Architektur	525
12.6 Zusammenfassung und Ausblick	526

13 Komponenten- und Unit-Tests: das Angular-Testing-Framework 529

13.1 Karma und Jasmine: Grundlagen zu Unit- und Komponententests in Angular-Anwendungen	530
--	-----

13.1.1	Karma einrichten	530
13.2	Der erste Unit-Test: einfache Klassen und Funktionen testen	534
13.2.1	Die Testausführung starten	536
13.2.2	Nur bestimmte Tests ausführen	538
13.3	Isolierte Komponenten testen: Grundlagen zu Komponententests mit dem Angular-Testing-Framework	539
13.3.1	Die zu testende Komponente	540
13.3.2	TestBed, ComponentFixture & Co – Konfiguration des Testmoduls und Erzeugung von Testkomponenten	541
13.4	Mocks und Spies: Komponenten mit Abhängigkeiten testen	544
13.4.1	Eigene Mocks für die Simulation von Services bereitstellen	546
13.4.2	inject – Zugriff auf die im Testkontext vorhandenen Services	548
13.4.3	TestBed.get: alternativer Zugriff auf die Services aus dem Ausführungskontext	549
13.4.4	Spies: ausgehende Aufrufe überwachen und auswerten	550
13.5	Services und HTTP-Backends testen	552
13.6	Formulare testen	557
13.6.1	Reactive Forms: Formulare direkt über die API testen	557
13.6.2	Template-driven Forms: generierte Formulare über die Forms-API testen	559
13.6.3	Formulare über die Oberfläche testen	561
13.7	Direktiven und ngContent-Komponenten testen	563
13.7.1	overrideComponent und compileComponents: Komponenten-Templates für den Test überschreiben	565
13.8	async und fakeAsync: mehr Kontrolle über asynchrone Tests	566
13.8.1	async: automatisch auf asynchrone Aufrufe warten	566
13.8.2	fakeAsync: komplexere asynchrone Szenarien steuern	567
13.9	Routing-Funktionalität testen	568
13.9.1	Manipulation von Router-Diensten im Komponententest	569
13.9.2	Ausführung echter Navigationsvorgänge	570
13.10	Zusammenfassung und Ausblick	572
14	Integrationstests mit Protractor	575
<hr/>		
14.1	Start der Tests und Konfiguration von Protractor	576
14.1.1	Installation und Konfiguration von Protractor	577

14.2	Anpassung der Applikationskonfiguration über die angular.json	578
14.3	Das browser-Objekt und Locators:	
	Übersicht über die Kernbestandteile von Protractor	581
14.3.1	browser – die Schnittstelle zur Interaktion mit dem Webbrowser	582
14.3.2	element und by – Protractor-Locators in Aktion	583
14.3.3	Promises bei der Arbeit mit der Protractor-API	586
14.4	Page-Objects: Trennung von Testlogik und technischen Details	587
14.5	Formulare und Alert-Boxen testen: der Edit-Task-Test	590
14.6	Seitenübergreifende Workflows testen	593
14.6.1	ExpectedConditions: komfortabel auf das Eintreten von Bedingungen warten	595
14.6.2	Zahlenwerte vergleichen – manuelle Auswertung der Promise.then-Rückgabewerte	596
14.7	Debugging von Protractor-Tests	597
14.8	Screenshots anfertigen	599
14.8.1	Nach jedem Test einen Screenshot aufnehmen	600
14.9	Zusammenfassung	602

15 NgModule und Lazy-Loading: Modularisierung Ihrer Anwendungen 605

15.1	Feature-Modules: Teilbereiche der Applikation kapseln	606
15.1.1	Feature-Module – den Aufgabenbereich modularisieren	607
15.1.2	Das Common-Module: Import von Angular-Standardfunktionalität	609
15.1.3	Routing in Feature-Modules – die Routing-Konfiguration modularisieren	609
15.1.4	Anpassungen am Hauptmodul und Integration des Feature-Modules	610
15.2	Shared-Modules: gemeinsam genutzte Funktionalität kapseln	614
15.2.1	Boilerplate-Code durch Shared-Modules vermeiden	617
15.3	Services und Modularisierung	618
15.3.1	Stolperfallen beim Einsatz von TreeShakable-Providern	618
15.3.2	Modularisierung von Services über die »provide«-Syntax	619

15.4	Lazy-Loading von Applikationsbestandteilen	622
15.4.1	Preloading von Feature-Modulen: das Beste aus beiden Welten	625
15.5	entryComponents: dynamisch geladene Komponenten registrieren	626
15.6	Zusammenfassung und Ausblick	627

16 Der Angular-Template-Compiler, Ahead-of-time Compilation und Tree-Shaking 629

16.1	Grundlagen zum Angular-Template-Compiler	630
16.2	Der Ahead-of-time-Compilation-Modus: Leistungsschub für Ihre Anwendung	632
16.2.1	Den Template-Compiler ausführen	633
16.2.2	Start der Anwendung über die statische Browser-Plattform	634
16.3	Tree-Shaking der Anwendung mit Rollup	634
16.4	Implementierungsregeln beim Einsatz von AOT	637
16.4.1	Konsistenz zwischen HTML- und Komponentencode	637
16.4.2	Kein Einsatz von privaten Membervariablen im Zusammenspiel mit Templates	638
16.4.3	Verzicht auf Inline-Funktionen	639
16.5	Zusammenfassung und Ausblick	640

17 Internationalisierung: mehrsprachige Angular-Anwendungen implementieren 643

17.1	Die Grundlagen des i18n-Frameworks	644
17.1.1	Bestimmen Sie die Sprache der Anwendung	645
17.2	ng-xi18n: automatische Generierung der Message-Datei	648
17.2.1	ng xi18n: Message-Dateien mithilfe des Angular-CLI generieren	651
17.2.2	Exkurs: die Übersetzungen mit git verwalten	651
17.3	Description und Meaning: Metadaten für Übersetzer übergeben	652
17.4	Weitere Übersetzungstechniken	653
17.4.1	Attribute (und Input-Bindings) übersetzen	653
17.4.2	Mehrere parallele Knoten übersetzen	654

17.5	Pluralisierung und geschlechterspezifische Texte	655
17.5.1	Pluralisierung: Texte abhängig vom Zahlenwert einer Variablen	655
17.5.2	Pluralisierungen übersetzen	658
17.5.3	l18nSelectPipe: geschlechterspezifische Texte festlegen	659
17.6	Statisch übersetzte Applikationen im AOT-Modus generieren	662
17.7	Zusammenfassung und Ausblick	666

18 Das Animation-Framework: Angular-Anwendungen animieren 669

18.1	Die erste Animation: Grundlagen zum Animation-Framework	670
18.1.1	Bidirektionale Transitionen	674
18.2	void und *: spezielle States zum Hinzufügen und Entfernen von DOM-Elementen	674
18.2.1	:enter und :leave – Shortcuts für das Eintreten und Verlassen des DOM	676
18.3	Animationen in Verbindung mit automatisch berechneten Eigenschaften	677
18.4	Animation-Lifecycles: auf den Start und das Ende von Animationen reagieren	679
18.5	Keyframes: Definition von komplexen, mehrstufigen Animationen	680
18.6	Styling von Komponenten, die in Animationen verwendet werden	682
18.7	Groups und Sequences: mehrere Animationen kombinieren	683
18.7.1	group: Animationsschritte parallel ausführen	683
18.7.2	sequence: Animationsschritte nacheinander ausführen	684
18.7.3	Kombination von sequence und group	685
18.8	Querying: komplexe Komponenten animieren	686
18.8.1	Selektoren für die animierten Elemente	689
18.8.2	Optionale Animationselemente definieren	690
18.9	Staggering: ausgefeilte Listenanimationen definieren	690
18.10	Animation von Routing-Vorgängen	692
18.10.1	Lifecycle-Hooks für Routing-Animationen	695
18.11	Zusammenfassung und Ausblick	697

19 Vollendet in Form und Funktion: Material Design und Angular Material 699

19.1 Material Design	700
19.1.1 Grundideen in der Welt von Material Design	701
19.1.2 Gestaltungsraster von Kopf bis Fuß	706
19.1.3 Farben: weniger ist manchmal mehr	712
19.1.4 Aufmerksamkeit erzeugen, ohne zu stören: Animationen	714
19.2 Angular Material	716
19.2.1 Beispielprojekt: Babywatch	716
19.2.2 Erstellen des Projekts und Installation von Angular Material	717
19.2.3 Einen Header für Babywatch erstellen	721
19.2.4 Anlegen von Menü- und Hauptanzeigebereich mithilfe von mat-sidenav	724
19.2.5 mat-nav-list und mat-list-item: Erstellen von Menüeinträgen zur Navigation	730
19.2.6 BabywatchService: Daten für die Applikation	734
19.2.7 mat-card: Darstellung der Timeline im Karten-Format	735
19.2.8 Eigene Icons mit mat-icon verwenden	738
19.2.9 Floating Action Button: die primäre Aktion der Applikation auslösen	739
19.2.10 MatInputModule und Kollegen: Erstellung der Eingabemaske für neue Timeline-Events	742
19.2.11 Einstellungen: Ändern des Babynamens und Löschen der Timeline	749
19.2.12 Der MatDialog-Service: eine Abfrage vor dem Löschen der Timeline implementieren	754
19.2.13 Eigenes Theme erstellen	757
19.3 Zusammenfassung	762

20 NPM-Libraries und Mono-Repos: Funktionalität in Bibliotheken auslagern und per NPM veröffentlichen 765

20.1 Das Angular-CLI Projekt einrichten	766
20.1.1 Fine-Tuning der Projektstruktur	767

20.2 Die generierte Bibliothek im Detail	769
20.2.1 public_api.ts und ng-package.json: Steuerungsdateien für die Bibliothekserzeugung	769
20.2.2 package.json und tsconfig.json: steuern, wie die Bibliothek veröffentlicht wird	771
20.3 Die Bibliothek kompilieren und im Demo-Projekt einbinden	773
20.4 Der Mono-Repo-Ansatz für die Entwicklung von mehreren Webapplikationen	776
20.4.1 Potenzielle Nachteile von echten NPM-Libraries	777
20.4.2 Der Mono-Repo-Ansatz: ständige Integration der aktuellen Library in allen Projekten	778
20.4.3 Nachteile von Mono-Repos (und Lösungsansätze dafür)	778
20.5 Die Bibliothek über npm veröffentlichen	780
20.5.1 Eine eigene NPM-Registry aufsetzen	780
20.5.2 Die eigene Registry verwenden	782
20.5.3 npm publish: die Library veröffentlichen	783
20.5.4 Die Library in einem anderen Projekt verwenden	784
20.5.5 Versionierung von NPM-Bibliotheken: Einführung in Semantic-Versioning	786
20.6 Best Practices für die Implementierung von stylebaren Komponenten	787
20.6.1 BEM und ViewEncapsulation.None: eigene Kapselung über definierte CSS-Strukturen	788
20.7 Zusammenfassung und Ausblick	792

21 Angular-Elements: Angular-Komponenten als WebComponent bereitstellen 795

21.1 Einführung in Custom-Elements und Angular-Elements	796
21.1.1 Hello-Custom-Elements: das erste Custom-Element	796
21.2 Angular-Komponenten als WebComponents bereitstellen	797
21.2.1 Die WebComponent über das AppModule registrieren	798
21.2.2 Den ButtonChooser als WebComponent bereitstellen	802
21.3 Zoneless-Applications:	
Angular-Anwendungen unabhängig von Zone.js machen	806
21.3.1 Die ChangeDetection selbst managen	807
21.4 Den Build für die WebComponent-Auslieferung optimieren	808

21.5	Die WebComponent in einem Angular-Projekt verwenden	810
21.6	Die WebComponent in einem Vue-Projekt verwenden	812
21.7	Zusammenfassung und Ausblick	816

22 Server-Side Rendering: Angular-Anwendungen auf dem Server rendern 819

22.1	Einführung in Server-Side Rendering (SSR): Grundlagen und Vorteile	819
22.1.1	Grundidee und Einsatzszenarien für das Server-Side Rendering	821
22.2	Das Angular-Projekt für das Server-Side Rendering vorbereiten	822
22.2.1	Notwendige Abhängigkeiten installieren	822
22.2.2	AppModule und AppServerModule implementieren: Ihre Anwendung auf SSR vorbereiten	823
22.2.3	Das Angular-Anwendungs-Bundle für den Server kompilieren	825
22.2.4	server.ts: die eigentliche Server-Anwendung auf Basis von Node.js implementieren	827
22.2.5	Webpack-Build für die Server-Applikation	830
22.2.6	Den Server starten: Server-Side Rendering in Aktion	831
22.3	isPlatformServer und isPlatformBrowser: Wo bin ich gerade?	833
22.4	Die State-Transfer-API: geladene Daten vom Server auf den Client transferieren	834
22.5	Title-Service und Meta-Service: Suchmaschinen-Optimierung und Einbindung in Social-Media-Seiten leicht gemacht	838
22.6	Notwendige Anpassungen am Project-Manager-Code: Stolperfallen und alternative Lösungsansätze beim Server-Side Rendering	842
22.6.1	Kein Zugriff auf den Local Storage (und andere Browserdienste)	842
22.6.2	Probleme bei der Verbindung mit Websockets	845
22.7	Die Anwendung in der Cloud deployen	846
22.7.1	Das Projekt in der Google Cloud Platform anlegen	847
22.7.2	Das Cloud-SDK installieren	849
22.7.3	Konfiguration des Projekts	849
22.7.4	Die Anwendung in der Cloud deployen	850
22.7.5	Den Projects-Server deployen	853
22.8	Zusammenfassung	856
22.9	Schlusswort	857

A	ECMAScript 2015 (and beyond)	859
A.1	ECMAScript : Was ist das?	859
A.2	Die neuen ECMAScript-Features kompilieren	860
A.3	Block-Scope-Variablen	861
A.4	Arrow Functions	864
A.5	Rest-Parameter	867
A.6	Spread-Operatoren	868
A.7	Default-Parameter	870
A.8	Destructuring	871
A.9	Klassen	876
A.10	Die Module-Syntax: JavaScript-Anwendungen modularisieren	882
A.11	Template-Strings	886
A.12	Promises	887
A.13	Die Async-Await-Syntax	893
A.14	Die for-of-Schleife	896
A.15	Symbole	896
A.16	Iteratoren und Iterables	898
A.17	Generatoren	900
A.18	Set und Map: neue Collection-Klassen	903
A.19	Erweiterungen von vorhandenen Standardklassen	907
B	Typsicheres JavaScript mit TypeScript	917
B.1	Einfache Typen	918
B.2	Klassen	928
B.3	Interfaces	935
B.4	Module	940
B.5	Type Inference: Typsicherheit ohne explizite Typangabe	943
B.6	Erweiterte Typ-Techniken	945
B.7	Generics	953
B.8	Typdeklarationen für nicht typisierte Bibliotheken	955
B.9	tsconfig.json: Konfiguration des TypeScript-Projekts	961
	Index	967