

Auf einen Blick

1	Angular-Kickstart: Ihre erste Angular-Webapplikation	31
2	Das Angular-CLI: professionelle Projektorganisation für Angular-Projekte	61
3	Komponenten und Templating: der Angular-Sprachkern	103
4	Direktiven: Komponenten ohne eigenes Template	167
5	Fortgeschrittene Komponenten-Konzepte	185
6	Standarddirektiven und Pipes: wissen, was das Framework an Bord hat	233
7	Services und Dependency-Injection: lose Kopplung für Ihre Business-Logik	273
8	Template-driven Forms: einfache Formulare auf Basis von HTML	307
9	Reactive Forms: Formulare dynamisch in der Applikationslogik definieren	355
10	Routing: Navigation innerhalb der Anwendung	397
11	HTTP: Anbindung von Angular-Applikationen an einen Webserver	457
12	Reaktive Architekturen mit RxJS	489
13	Komponenten- und Unit-Tests: das Angular-Testing-Framework	533
14	Cypress: komfortable Integrationstests für Ihre Anwendung	581
15	NgModule und Lazy-Loading: Modularisierung Ihrer Anwendungen	625
16	Internationalisierung: mehrsprachige Angular-Anwendungen implementieren	653
17	Das Animation-Framework: Angular-Anwendungen animieren	679
18	Vollendet in Form und Funktion: Material Design und Angular Material	709
19	NPM-Librarys und Mono-Repos: Funktionalität in Bibliotheken auslagern und per NPM veröffentlichen	773
20	Angular Elements: Angular-Komponenten als WebComponents bereitstellen	805
21	Docker: Die Anwendung im Container deployen	829
22	Server-Side Rendering: Angular-Anwendungen auf dem Server rendern	851

Inhalt

Materialien zum Buch	25
Vorwort	27

1

**Angular-Kickstart:
Ihre erste Angular-Webapplikation**

31

1.1	Installation der benötigten Software	31
1.1.1	Node.js und npm	31
1.1.2	Visual Studio Code: eine kostenlose Entwicklungsumgebung für TypeScript und Angular	32
1.1.3	Alternative: Webstorm: perfekte Angular-Unterstützung	32
1.2	Hallo Angular	33
1.2.1	Komponenten konfigurieren	36
1.2.2	Die Komponenten-Klasse	38
1.2.3	Das Applikationsmodul: das Hauptmodul der Anwendung konfigurieren	39
1.2.4	main.ts: Wahl der Ausführungsplattform und Start des Applikationsmoduls	41
1.2.5	index.html: Einbinden der Bootstrap-Komponente und Start der Anwendung	41
1.3	Die Blogging-Anwendung	43
1.3.1	Start der Applikation	46
1.3.2	Einige Tipps zur Fehlersuche	47
1.3.3	Die Formular-Komponente: Daten aus der View in die Komponenten-Klasse übertragen	48
1.3.4	Das Applikationsmodell	50
1.3.5	Darstellung der Liste in der View	53
1.3.6	Modularisierung der Anwendung	55
1.4	Zusammenfassung und Ausblick	59

2	Das Angular-CLI: professionelle Projektorganisation für Angular-Projekte	61
2.1	Das Angular-CLI installieren	62
2.2	ng new: ein Grundgerüst für die Applikation erstellen	62
2.2.1	Konfigurationsoptionen für die Projektgenerierung	65
2.2.2	Das generierte Projekt im Detail	66
2.3	ng serve: die Anwendung starten	69
2.3.1	Die Proxy-Konfiguration	71
2.3.2	ng serve-Default-Optionen über die angular.json einstellen	71
2.4	npm start: Start über die lokale CLI-Version	73
2.5	ng generate: Komponenten generieren	74
2.5.1	Konfigurationsoptionen bei der Komponentengenerierung	76
2.5.2	Weitere Generatoren	77
2.6	ng update: Angular und weitere Abhängigkeiten auf die neueste Version updaten	78
2.7	ng lint: Linting und der Angular-Style-Guide	80
2.8	Komponenten- und Ende-zu-Ende-Tests ausführen	82
2.8.1	ng test: Unit- und Komponenten-Tests ausführen	82
2.8.2	ng e2e: Ende-zu-Ende-Tests ausführen	84
2.9	CSS-Präprozessoren verwenden	85
2.10	Drittanbieter-Bibliotheken einbinden	86
2.10.1	Bibliotheken über die index.html einbinden	87
2.11	ng add: Angular-spezifische Abhängigkeiten zu Ihrer Anwendung hinzufügen	87
2.12	ng build: deploybare Builds erstellen	90
2.12.1	Konfigurationsoptionen für die Ausführung des Builds	90
2.13	Configurations: Konfiguration unterschiedlicher Build- und Ausführungsumgebungen	92
2.13.1	File-Replacements: Dateien abhängig von der Konfiguration austauschen	93
2.13.2	Eigene Build-Konfigurationen anlegen und aktivieren	94
2.13.3	Konfigurationen für den ng serve-Befehl	95
2.13.4	Mehrere Konfigurationen für Ihren Build aktivieren	97
2.14	ng deploy: die Anwendung im Web deployen	97
2.14.1	Praxisbeispiel: Deployment auf GitHub Pages	98
2.15	Zusammenfassung und Ausblick	101

3	Komponenten und Templating: der Angular-Sprachkern	103
3.1	Etwas Theorie: der Angular-Komponenten-Baum	103
3.2	Selektoren: vom DOM-Element zur Angular-Komponente	107
3.2.1	Tag-Selektoren	107
3.2.2	Attribut-Selektoren	108
3.2.3	Klassen-Selektoren	109
3.2.4	not()-Selektoren	109
3.2.5	Verknüpfung von Selektoren	109
3.3	Die Templating-Syntax: Verbindung zwischen Applikationslogik und Darstellung	110
3.3.1	Fallbeispiel: Timepicker-Komponente	110
3.3.2	Property-Bindings	111
3.3.3	Sonderfälle: Attribute, Klassen und Styles setzen	114
3.3.4	Interpolation: Darstellung von Werten im View	116
3.3.5	Event-Bindings	117
3.3.6	Two-Way-Data-Bindings mit NgModel	121
3.3.7	Lokale Template-Variablen	123
3.3.8	Die *-Templating-Microsyntax – neue DOM-Elemente dynamisch einfügen	124
3.3.9	Templating-Syntax-Spickzettel	128
3.4	Komponenten-Schnittstellen definieren: von der einzelnen Komponente zur vollständigen Applikation	129
3.4.1	Input-Bindings: Werte in Ihre Komponenten hineinreichen	129
3.4.2	Output-Bindings: andere Komponenten über Datenänderungen informieren	134
3.4.3	Two-Way-Data-Bindings: syntaktischer Zucker für Ihre Komponenten-Schnittstelle	136
3.4.4	ngOnChanges: auf Änderungen von Bindings reagieren	137
3.4.5	Lokale Komponenten-Variablen: Zugriff auf die API Ihrer Kind-Elemente im HTML-Code	139
3.5	ViewChildren: Zugriff auf Kind-Elemente aus der Komponenten-Klasse	140
3.6	Content-Insertion: dynamische Komponenten-Hierarchien erstellen	143
3.6.1	Einfachen HTML-Code injizieren	143
3.6.2	ContentChildren: Erzeugung von dynamischen Komponenten-Bäumen am Beispiel einer Tabs-Komponente	149
3.7	Der Lebenszyklus einer Komponente	153
3.7.1	Der Konstruktor: Instanziierung der Komponente	156

3.7.2	ngOnInit: Initialisierung der eigenen Komponente	157
3.7.3	ngOnChanges: auf Änderungen reagieren	158
3.7.4	ngAfterContentInit: auf die Initialisierung von Content-Children reagieren	159
3.7.5	ngAfterViewInit: auf die Initialisierung von ViewChildren reagieren	159
3.7.6	ngOnDestroy: Aufräumarbeiten vornehmen	160
3.7.7	ngAfterContentChecked, ngAfterViewChecked: den ChangeDetection-Mechanismus verfolgen	161
3.7.8	ngDoCheck: den ChangeDetection-Mechanismus verändern	162
3.8	Zusammenfassung und Ausblick	164

4 Direktiven: Komponenten ohne eigenes Template 167

4.1	ElementRef und Renderer2: Manipulation von DOM-Eigenschaften eines Elements	168
4.1.1	Die Renderer2-Klasse: das native Element plattformunabhängig manipulieren	171
4.2	HostBinding und HostListener: Auslesen und Verändern von Host-Eigenschaften und -Events	172
4.3	Anwendungsfall: Einbinden von Drittanbieter-Bibliotheken	174
4.3.1	Two-Way-Data-Binding für die Slider-Komponente	177
4.4	Anwendungsfall: Accordion-Direktive – mehrere Kind-Komponenten steuern	178
4.5	exportAs: Zugriff auf die Schnittstelle einer Direktive	181
4.6	Zusammenfassung und Ausblick	183

5 Fortgeschrittene Komponenten-Konzepte 185

5.1	Styling von Angular-Komponenten	185
5.1.1	Styles an der Komponente definieren	186
5.1.2	ViewEncapsulation – Strategien zum Kapseln Ihrer Styles	187
5.2	TemplateRef und NgTemplateOutlet: dynamisches Austauschen von Komponenten-Templates	196
5.2.1	NgFor mit angepassten Templates verwenden	197

5.2.2	NgTemplateOutlet: zusätzliche Templates an die Komponente übergeben	200
5.3	ViewContainerRef: Komponenten zur Laufzeit hinzufügen	204
5.3.1	ViewContainerRef: Komponenten zur Laufzeit hinzufügen	204
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	207
5.3.5	createEmbeddedView: eigene strukturelle Direktiven implementieren	209
5.4	NgComponentOutlet: dynamisch erzeugte Komponenten noch einfacher verwalten	213
5.4.1	Übergabe von dynamischen Eigenschaften an NgComponentOutlet	214
5.5	ChangeDetection-Strategien: Performance-Boost für Ihre Applikation	217
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	241
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	247
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	250
6.2.4	KeyValuePipe: über Objekte und Maps iterieren	251
6.2.5	DecimalPipe: Zahlenwerte formatieren	253
6.2.6	Kurzexkurs: lokalisierbare Pipes – Werte der aktuellen Sprache entsprechend formatieren	254
6.2.7	DatePipe: Datums- und Zeitwerte darstellen	255
6.2.8	Percent- und CurrencyPipe: Prozent- und Währungswerte formatieren	258
6.2.9	Die AsyncPipe: auf asynchrone Werte warten	259
6.2.10	Pipes im Komponenten-Code verwenden	262
6.2.11	Eigene Pipes implementieren	263
6.2.12	Pure vs. Impure Pipes: Pipe, ändere dich!	267
6.3	Zusammenfassung und Ausblick	270

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
7.3.3	Den @Inject-Decorator vermeiden	281
7.3.4	Der @Injectable-Decorator: TypeScript-optimierte Injektion in Services	282
7.3.5	Member-Injection: automatische Erzeugung von Membervariablen	283
7.4	Weitere Provider-Formen	284
7.4.1	Injection-Tokens: kollisionsfreie Definition von DI-Schlüsseln	286
7.5	Der hierarchische Injector-Baum: volle Flexibilität bei der Definition Ihrer Abhängigkeiten	288
7.5.1	Der Injector-Baum	288
7.5.2	Registrierung von globalen Services: der UserService	290
7.5.3	Registrieren von komponenten-bezogenen Services: MusicSearchService und VideoSearchService	292
7.6	Treeshakable-Providers: der DI-Mechanismus auf den Kopf gestellt	296

7.7	Sichtbarkeit und Lookup von Dependencies	297
7.7.1	Sichtbarkeit von Providern beschränken	298
7.7.2	Den Lookup von Abhängigkeiten beeinflussen	300
7.8	Zusammenfassung und Ausblick	304

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 Eingabeelemente	322
8.5.1	Auswahllisten	322
8.5.2	Checkboxes	327
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	334
8.7.4	Eigene Validierungsregeln definieren	338
8.7.5	Asynchrone Validierungen	341
8.7.6	Feldübergreifende Validierungen	345
8.8	Implementierung der Tags-Liste: wiederholbare Strukturen mit Template-driven Forms	347
8.9	updateOn: steuern, wann Änderungen übernommen werden	351
8.10	Zusammenfassung und Ausblick	352

9	Reactive Forms: Formulare dynamisch in der Applikationslogik definieren	355
9.1	Aktivierung von Reactive Forms für Ihre Applikation	356
9.2	Das Task-Formular im reaktiven Ansatz	356
9.2.1	Definition des Formulars im TypeScript-Code	357
9.2.2	Verknüpfung des Formulars mit dem HTML-Code	358
9.2.3	FormArray im Detail: wiederholbare Strukturen definieren	360
9.2.4	Verbindung des Formulars mit dem Applikationsmodell	364
9.2.5	Der FormBuilder: komfortable Definition von Formularen	368
9.2.6	Validierungen von Reactive Forms	369
9.2.7	updateOn in Reactive Forms	376
9.3	Formulare und Kontrollelemente auf Änderungen überwachen	377
9.4	Fallbeispiel: Umfragebogen – Formulare komplett dynamisch definieren	378
9.5	ControlValueAccessor: eigene Eingabeelemente für die Forms-API implementieren	385
9.5.1	Das neue Eingabeelement bei der Forms-API registrieren	387
9.5.2	writeValue und registerOnChange: Werte von der Forms-API empfangen und an die Form-API propagieren	387
9.5.3	Verwendung des neuen Eingabeelements in Formularen	388
9.5.4	registerOnTouched: den touched-Status nach außen propagieren	391
9.5.5	setDisabledState: den disabled-Status Ihres Eingabeelements verwalten	393
9.6	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
10.2.1	RouterOutlet: festlegen, wo der Inhalt von Routen dargestellt werden soll	402
10.3	Location-Strategien: »schöne URLs« vs. »Routing ohne Server-Konfiguration«	404
10.3.1	PathLocation-Strategie: schöne URLs	404
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
10.5.2 isActiveChange: auf das Aktivieren eines Links reagieren	415
10.6 Routing-Parameter: dynamische Adresszeilenparameter auswerten	416
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	420
10.6.4 Query-Parameter: optionale Parameter unabhängig vom Segment definieren	424
10.6.5 Fragmentbezeichner	425
10.7 Aus der Anwendungslogik heraus navigieren	427
10.7.1 Die navigate-Methode: Navigation auf Basis der Routing-DSL	427
10.7.2 navigateByUrl: Navigation auf Basis von URLs	429
10.8 Routing-Guards: Routen absichern und die Navigation generisch beeinflussen	429
10.8.1 CanActivate: Routen absichern	430
10.8.2 CanActivateChild	433
10.8.3 CanDeactivate: das Verlassen einer Route verhindern	434
10.9 Redirects und Wildcard-URLs	436
10.9.1 Absolute Redirects	436
10.9.2 Relative Redirects	437
10.9.3 Wildcard-URLs: Platzhalter-Routen definieren	438
10.10 Data: statische Metadaten an Routen hinterlegen	439
10.11 Resolve: dynamische Daten über den Router injizieren	439
10.11.1 Verwendung einer resolve-Funktion anstelle einer Resolver-Klasse	441
10.12 Der Title-Service: den Seitentitel verändern	442
10.13 Router-Tree und Router-Events: generisch auf Seitenwechsel reagieren	444
10.13.1 Der events-Stream: bei Seitenwechseln informiert werden	444
10.13.2 Der Router-Tree: den aktuellen Router-Baum durchlaufen	445
10.14 Location: direkte Interaktion mit der Adresszeile des Browsers	447
10.15 Mehrere RouterOutlets: maximale Flexibilität beim Routing	449
10.15.1 Zusätzliche Outlets: ein Chat-Fenster einblenden	449
10.15.2 Komplexere Outlet-Konfigurationen: eine Task-Schnellansicht	452
10.16 Zusammenfassung und Ausblick	455

11	HTTP: Anbindung von Angular-Applikationen an einen Webserver	457
11.1	Die Server-Applikation	458
11.1.1	Die json-server-Bibliothek	459
11.2	Das Angular-HTTP-Modul verwenden	462
11.3	Der erste GET-Request: Grundlagen zur HTTP-API	462
11.3.1	Auf Fehler reagieren	464
11.4	Asynchrone Service-Schnittstellen modellieren: Anpassung des TaskService	466
11.4.1	Observables statt Callbacks: Daten reaktiv verwalten	466
11.5	Die AsyncPipe: noch eleganter mit asynchronen Daten arbeiten	468
11.6	HttpParams: elegant dynamische Suchen definieren	469
11.7	Die observe-Eigenschaft: die komplette HttpResponse auswerten	472
11.8	POST, PUT, DELETE, PATCH und HEAD: Verwendung der weiteren HTTP-Methoden	474
11.8.1	HTTP-POST: neue Tasks anlegen	474
11.8.2	HTTP-PUT: bestehende Tasks editieren	476
11.8.3	HTTP-DELETE: Tasks löschen	477
11.8.4	Generische Anfragen: die »request«-Methode	478
11.8.5	HTTP-PATCH: Tasks partiell verändern	480
11.8.6	HTTP-HEAD: der kleine Bruder von GET	481
11.9	JSONP	482
11.9.1	Die jsonp-Methode	484
11.10	Zusammenfassung und Ausblick	486
12	Reaktive Architekturen mit RxJS	489
12.1	Kurzeinführung in RxJS	490
12.1.1	Observables und Observer-Functions: die Kernelemente der reaktiven Programmierung	490
12.1.2	Subscriptions, der takeUntil-Operator und Disposing-Functions: Observables sauber beenden	492
12.1.3	Subjects: Multicast-Funktionalität auf Basis von RxJS	496
12.2	Implementierung einer Typeahead-Suche	498
12.2.1	mergeMap: verschachtelte Observables verbinden	502

12.2.2	switchMap: nur die aktuellsten Ergebnisse verarbeiten	503
12.2.3	merge: mehrere Streams vereinen	504
12.3	Reaktive Datenarchitekturen in Angular-Applikationen	507
12.3.1	Shared Services: der erste Schritt in die richtige Richtung	509
12.3.2	Die neue Datenarchitektur: »Push« statt »Pull«	511
12.3.3	Umsetzung des neuen Konzepts in Angular	514
12.3.4	Anbindung der TaskListComponent an den Store	521
12.3.5	Der »In Bearbeitung«-Zähler	523
12.4	Anbindung von Websockets zur Implementierung einer Echtzeitanwendung	524
12.4.1	Der WebSocket-Server	525
12.4.2	Integration von Socket.IO in die Anwendung	527
12.4.3	Verwendung von Socket.IO im TaskService	528
12.5	ChangeDetectionStrategy.OnPush: Performance-Schub durch die reaktive Architektur	530
12.6	Zusammenfassung und Ausblick	531

13 Komponenten- und Unit-Tests: das Angular-Testing-Framework 533

13.1	Karma und Jasmine: Grundlagen zu Unit- und Komponenten-Tests in Angular-Anwendungen	534
13.1.1	Karma einrichten	534
13.2	Der erste Unit-Test: einfache Klassen und Funktionen testen	537
13.2.1	Die Testausführung starten	540
13.2.2	Nur bestimmte Tests ausführen	541
13.3	Isolierte Komponenten testen: Grundlagen zu Komponenten-Tests mit dem Angular-Testing-Framework	543
13.3.1	Die zu testende Komponente	543
13.3.2	TestBed, ComponentFixture & Co: Konfiguration des Testmoduls und Erzeugung von Test-Komponenten	545
13.3.3	nativeElement und detectChange: mit dem DOM-Baum der Komponente interagieren.	547
13.4	Mocks und Spies: Komponenten mit Abhängigkeiten testen	549
13.4.1	Eigene Mocks für die Simulation von Services bereitstellen	550
13.4.2	TestBed.inject: Zugriff auf die im Testkontext vorhandenen Services	552

13.4.3	spyOn: das Verhalten von Funktionen manipulieren	553
13.4.4	Spies: ausgehende Aufrufe überwachen, manipulieren und auswerten	553
13.5	Services und HTTP-Backends testen	555
13.6	Formulare testen	560
13.6.1	Reactive Forms: Formulare direkt über die API testen	560
13.6.2	Template-driven Forms: generierte Formulare über die Forms-API testen	562
13.6.3	Formulare über die Oberfläche testen	564
13.7	Direktiven und ngContent-Komponenten testen	566
13.7.1	overrideComponent und compileComponents: Komponenten-Templates für den Test überschreiben	568
13.8	waitForAsync und fakeAsync: mehr Kontrolle über asynchrone Tests	569
13.8.1	waitForAsync: automatisch auf asynchrone Aufrufe warten	569
13.8.2	fakeAsync: komplexere asynchrone Szenarien steuern	571
13.9	Routing-Funktionalität testen	572
13.9.1	Manipulation von Router-Diensten im Komponenten-Test	573
13.9.2	Ausführung echter Navigationsvorgänge	574
13.10	Die Tests auf Ihrem Build-Server ausführen	576
13.11	Zusammenfassung und Ausblick	577

14 Cypress: komfortable Integrationstests für Ihre Anwendung 581

14.1	Cypress zum Projekt hinzufügen und ausführen	582
14.2	Cypress lokal und auf Ihrem Build-Server ausführen	583
14.2.1	Der interaktive Modus: sehen, was passiert	583
14.2.2	Der Headless-Modus: perfekte Integration in Ihr Build-System	586
14.3	Cypress konfigurieren	587
14.3.1	Mehrere Konfigurationsdateien verwenden	590
14.3.2	Überschreiben von Konfigurationsoptionen über Commandline-Parameter	590
14.4	Cypress-Grundlagen: Ihre ersten eigenen Cypress-Tests	590
14.4.1	Test-Struktur	591
14.4.2	Erwartungen (Assertions)	592
14.4.3	Cypress-Kommandos, das Chainable-Interface und das globale cy-Objekt	592

14.5 Selektoren, Interaktion mit Elementen und weitere Assertion-Typen: den Tasks-Bereich testen	594
14.5.1 cy.get und cy.its: auf DOM-Elemente und deren Eigenschaften zugreifen	594
14.5.2 type und click: Interaktion mit DOM-Elementen	596
14.5.3 Tasks erzeugen: Test der Formularfunktionalität und des Seitenwechsels	598
14.6 cy.on: auf Browser-Events reagieren	601
14.7 Intercept: REST-Requests untersuchen und manipulieren	602
14.7.1 Manipulation von Antworten mithilfe von Fixture-Dateien	603
14.7.2 Auf REST-Anfragen warten: eine bessere Lösung für wait()	604
14.7.3 REST-Anfragen und -Antworten untersuchen	605
14.8 Custom Commands: Den Funktionsumfang von Cypress mit eigenen Kommandos dynamisch erweitern	609
14.8.1 Definieren von TypeScript-Typings für Ihre eigenen Kommandos	610
14.9 Screenshots und Video-Recordings: sehen, was schief läuft	612
14.9.1 Automatische und manuelle Screenshots Ihrer Tests	612
14.9.2 Video-Recordings Ihrer Testausführung	614
14.10 Debugging von Cypress-Tests	615
14.10.1 Verwendung des Schlüsselworts debugger	615
14.10.2 Verwendung des debug-Kommandos	616
14.10.3 Die Testausführung schrittweise durchlaufen: das pause-Kommando	618
14.11 Die Cypress-Beispiele als Dokumentation nutzen	619
14.12 Zusammenfassung und Ausblick	621

15 NgModule und Lazy-Loading: Modularisierung Ihrer Anwendungen 625

15.1 Feature-Module: Teilbereiche der Applikation kapseln	626
15.1.1 Feature-Module: den Aufgabenbereich modularisieren	628
15.1.2 Das Common-Modul: Import von Angular-Standardfunktionalität	629
15.1.3 Routing in Feature-Modulen: die Routing-Konfiguration modularisieren	629
15.1.4 Anpassungen am AppRoutingModuleModule und Integration des Feature-Moduls	630

15.2	Shared-Modules: gemeinsam genutzte Funktionalität kapseln	634
15.2.1	Boilerplate-Code durch Shared-Module vermeiden	637
15.3	Module per Component: das höchste Level an Modularisierung	638
15.4	Services und Modularisierung	640
15.4.1	Services in Feature-Modulen registrieren	641
15.4.2	Stolperfallen beim Bereitstellen von Services über Shared-Module	642
15.4.3	ModuleWithProviders: individualisierte Module je nach Einsatzort – eigene forRoot- oder forChild-Module definieren	643
15.5	Lazy-Loading von Applikationsbestandteilen	647
15.5.1	Preloading von Feature-Modulen: das Beste aus beiden Welten	650
15.6	Zusammenfassung und Ausblick	651

16 Internationalisierung: mehrsprachige Angular-Anwendungen implementieren 653

16.1	Einrichtung des i18n-Frameworks	654
16.1.1	One Build per Locale: strikte Trennung der unterschiedlichen Sprachversionen	655
16.1.2	Den Build im Angular-CLI vorbereiten	656
16.1.3	Die darzustellende Sprache dynamisch ermitteln	658
16.1.4	Sprachspezifische Build-Konfigurationen verwalten	660
16.2	ng extract-i18n: automatische Generierung der Message-Datei	662
16.3	Eigene Übersetzungsschlüssel definieren	664
16.4	Description und Meaning: Metadaten für Übersetzer übergeben	665
16.5	Weitere Übersetzungstechniken	666
16.5.1	Attribute (und Input-Bindings) übersetzen	666
16.5.2	Mehrere parallele Knoten übersetzen	667
16.6	\$localize: Texte aus dem TypeScript-Code heraus übersetzen	668
16.6.1	Den \$localize-Service mit Platzhaltern verwenden	669
16.7	Pluralisierung und geschlechtsspezifische Texte	670
16.7.1	Pluralisierung: Texte abhängig vom Zahlenwert einer Variablen	670
16.7.2	Pluralisierungen übersetzen	673
16.7.3	I18nSelectPipe: geschlechtsspezifische Texte festlegen	675
16.8	Zusammenfassung und Ausblick	677

17 Das Animation-Framework: Angular-Anwendungen animieren 679

17.1 Die erste Animation: Grundlagen zum Animation-Framework	680
17.1.1 Bidirektionale Transitionen	684
17.2 void und *: spezielle States zum Hinzufügen und Entfernen von DOM-Elementen	684
17.2.1 :enter und :leave: Shortcuts für das Eintreten und Verlassen des DOM-Baums	686
17.3 Animationen in Verbindung mit automatisch berechneten Eigenschaften	687
17.4 Animation-Lifecycles: auf den Start und das Ende von Animationen reagieren	689
17.5 Keyframes: Definition von komplexen, mehrstufigen Animationen	690
17.6 Styling von Komponenten, die in Animationen verwendet werden	691
17.7 Groups und Sequences: mehrere Animationen kombinieren	693
17.7.1 group: Animationsschritte parallel ausführen	693
17.7.2 sequence: Animationsschritte nacheinander ausführen	694
17.7.3 Kombination von sequence und group	695
17.8 Querying: komplexe Komponenten animieren	696
17.8.1 Selektoren für die animierten Elemente	699
17.8.2 Optionale Animationselemente definieren	700
17.9 Staggering: ausgefeilte Listenanimationen definieren	700
17.10 Animation von Routing-Vorgängen	702
17.10.1 Lifecycle-Hooks für Routing-Animationen	705
17.11 Zusammenfassung und Ausblick	707

18 Vollendet in Form und Funktion: Material Design und Angular Material 709

18.1 Material Design	710
18.1.1 Die Grundideen in der Welt von Material Design	711
18.1.2 Gestaltungsraster von Kopf bis Fuß	715
18.1.3 Farben: weniger ist manchmal mehr	722
18.1.4 Aufmerksamkeit erzeugen, ohne zu stören: Animationen	724

18.2	Angular Material	725
18.2.1	Beispielprojekt: Babywatch	726
18.2.2	Erstellen des Projekts und Installation von Angular Material	727
18.2.3	Einen Header für Babywatch erstellen	731
18.2.4	Anlegen von Menü- und Hauptanzeigebereich mithilfe von mat-sidenav	734
18.2.5	mat-nav-list und mat-list-item: Erstellen von Menüeinträgen zur Navigation	740
18.2.6	BabywatchService: Daten für die Applikation	744
18.2.7	mat-card: Darstellung der Timeline im Karten-Format	744
18.2.8	Eigene Icons mit mat-icon verwenden	747
18.2.9	Floating Action Button: die primäre Aktion der Applikation auslösen	749
18.2.10	MatInputModule und Kollegen: die Eingabemaske für neue Timeline-Events erstellen	752
18.2.11	Einstellungen: Ändern des Babynamens und Löschen der Timeline	758
18.2.12	Der MatDialog-Service: eine Abfrage vor dem Löschen der Timeline implementieren	763
18.2.13	Eigenes Theme erstellen	765
18.3	Zusammenfassung	771

19 NPM-Librarys und Mono-Repos: Funktionalität in Bibliotheken auslagern und per NPM veröffentlichen 773

19.1	Das Angular-CLI-Projekt einrichten	774
19.1.1	Fine-Tuning der Projektstruktur	775
19.2	Die generierte Bibliothek im Detail	777
19.2.1	public-api.ts und ng-package.json: Steuerungsdateien für die Bibliothekserzeugung	777
19.2.2	package.json und tsconfig.json: steuern, wie die Bibliothek veröffentlicht wird	778
19.3	Die Bibliothek kompilieren und im Demo-Projekt einbinden	781
19.4	Der Mono-Repo-Ansatz für die Entwicklung von mehreren Webapplikationen	784
19.4.1	Potenzielle Nachteile von echten NPM-Librarys	785
19.4.2	Der Mono-Repo-Ansatz: ständige Integration der aktuellen Library in allen Projekten	785
19.4.3	Nachteile von Mono-Repos (und Lösungsansätze dafür)	786

19.5	Die Bibliothek über npm veröffentlichen	788
19.5.1	Eine eigene NPM-Registry aufsetzen	788
19.5.2	Die eigene Registry verwenden	790
19.5.3	npm publish: die Library veröffentlichen	791
19.5.4	Die Library in einem anderen Projekt verwenden	792
19.5.5	Versionierung von NPM-Bibliotheken: Einführung in das Semantic-Versioning	793
19.6	Best Practices für die Implementierung von stylebaren Komponenten	794
19.6.1	Lösung 1: CSS-Variablen	796
19.6.2	Lösung 2: BEM und ViewEncapsulation.None: eigene Kapselung über definierte CSS-Strukturen	798
19.7	Zusammenfassung und Ausblick	802

20 Angular Elements: Angular-Komponenten als WebComponents bereitstellen 805

20.1	Einführung in Custom Elements und Angular Elements	806
20.1.1	Hello-Custom Elements: das erste Custom-Element	806
20.2	Angular-Komponenten als WebComponents bereitstellen	807
20.2.1	Die WebComponent über das AppModule registrieren	808
20.2.2	Den ButtonChooser als WebComponent bereitstellen	812
20.3	Zoneless-Applications: Angular-Anwendungen unabhängig von Zone.js machen	816
20.3.1	Die ChangeDetection selbst managen	817
20.4	Den Build für die WebComponent-Auslieferung optimieren	818
20.5	Die WebComponent in einem Angular-Projekt verwenden	819
20.6	Die WebComponent in einem Vue-Projekt verwenden	822
20.7	Zusammenfassung und Ausblick	826

21 Docker: Die Anwendung im Container deployen 829

21.1	Deployment über nginx: das Docker-Image erstellen und als Container starten	830
21.1.1	nginx.conf: den Webserver konfigurieren	830
21.1.2	Ihr erstes Dockerfile: die Anwendung in nginx bereitstellen	831
21.1.3	docker build: das Docker-Image erzeugen	832

21.1.4	docker run: den Docker-Container starten	833
21.1.5	Die Docker-Befehle in npm-Skripte auslagern	834
21.2	Multi-Stage Builds	835
21.2.1	Das Build-Ergebnis im nginx-Image verwenden	838
21.2.2	Die Tests im Docker-Container ausführen	838
21.3	Die Anwendung über Umgebungsvariablen konfigurieren	840
21.3.1	Dynamische Konfigurationen per JSON-verwalten	842
21.3.2	Der APP_INITIALIZER-Mechanismus: Aufgaben vor dem Anwendungsstart verarbeiten	843
21.3.3	Entrypoint und envsubst: Umgebungsvariablen in die Anwendung hineingeben	845
21.4	Zusammenfassung und Ausblick	848

22 Server-Side Rendering: Angular-Anwendungen auf dem Server rendern 851

22.1	Einführung in Server-Side Rendering (SSR): Grundlagen und Vorteile	851
22.1.1	Grundidee und Einsatzszenarien für das Server-Side Rendering	853
22.2	Das Angular-Projekt für das Server-Side Rendering vorbereiten	854
22.2.1	Notwendige Abhängigkeiten installieren	854
22.2.2	AppModule und AppServerModule implementieren: Ihre Anwendung auf SSR vorbereiten	855
22.2.3	Anpassungen und Erweiterungen am Angular-Anwendungscode	856
22.2.4	server.ts: die eigentliche Server-Anwendung auf Basis von Node.js implementieren	857
22.2.5	Der Angular-CLI-Build für die Server-Applikation	859
22.2.6	Den Server starten: Server-Side Rendering in Aktion	860
22.3	isPlatformServer und isPlatformBrowser: Wo bin ich gerade?	862
22.4	Die State-Transfer-API: geladene Daten vom Server auf den Client transferieren	862
22.5	Title-Service und Meta-Service: Suchmaschinen-Optimierung und Einbindung in Social-Media-Seiten leicht gemacht	867
22.6	Notwendige Anpassungen am Project-Manager-Code: Stolperfallen und alternative Lösungsansätze beim Server-Side Rendering	870
22.6.1	Einschränkungen bei der Verwendung von absoluten Links	870
22.6.2	Kein Zugriff auf den Local Storage (und andere Browserdienste)	872
22.6.3	Probleme bei der Verbindung mit Websockets	875

22.7 Die Anwendung in der Cloud deployen	876
22.7.1 Das Projekt in der Google Cloud Platform anlegen	876
22.7.2 (Optional) Budget-Alarme festlegen	878
22.7.3 Das Cloud-SDK installieren	880
22.7.4 Konfiguration des Projekts	881
22.7.5 Die Anwendung in der Cloud deployen	882
22.7.6 Den Projects-Server deployen	884
22.7.7 envsub-Umgebungsvariablen per npm-Skript ersetzen	886
22.8 Zusammenfassung	889

Anhang	891
---------------	-----

A ECMAScript 2015 (and beyond)	891
A.1 ECMAScript: Was ist das?	891
A.2 Die neuen ECMAScript-Features kompilieren	892
A.3 Block-Scope-Variablen	893
A.4 Arrow Functions	896
A.5 Rest-Parameter	899
A.6 Spread-Operatoren	900
A.7 Default-Parameter	902
A.8 Destructuring	903
A.9 Klassen	908
A.10 Die Module-Syntax: JavaScript-Anwendungen modularisieren	914
A.11 Template-Strings	918
A.12 Promises	918
A.13 Die async/await-Syntax	924
A.14 Die for-of-Schleife	927
A.15 Symbole	928
A.16 Iteratoren und Iterables	929
A.17 Generatoren	931
A.18 Set und Map: neue Collection-Klassen	935
A.19 Erweiterungen von vorhandenen Standardklassen	938
B Typsicheres JavaScript mit TypeScript	947
B.1 Einfache Typen	947
B.2 Klassen	958
B.3 Interfaces	964
B.4 Module	969
B.5 Type Inference: Typsicherheit ohne explizite Typangabe	972
B.6 Erweiterte Typ-Techniken	974

B.7	Generics	981
B.8	null- und undefined-Handling: Arbeit mit optionalen Werten und Eigenschaften	983
B.9	Utility-Typen: dynamische Typen auf Basis existierender Klassen und Interfaces	990
B.10	Typdeklarationen für nicht typisierte Bibliotheken	995
B.11	tsconfig.json: Konfiguration des TypeScript-Projekts	1001
Index		1007