

Inhalt

Geleitwort des Fachgutachters	21
Einleitung	23

1 Angular-Kickstart: Ihre erste Angular-Webapplikation 25

1.1 Installation der benötigten Software	25
1.1.1 Node.js und npm	25
1.1.2 Visual Studio Code – eine kostenlose Entwicklungsumgebung für TypeScript und Angular	26
1.1.3 Alternative: Webstorm – perfekte Angular-Unterstützung	27
1.2 Hallo Angular	27
1.2.1 Komponenten konfigurieren	30
1.2.2 Die Komponenten-Klasse	32
1.2.3 Das Applikationsmodul: das Hauptmodul der Anwendung konfigurieren	33
1.2.4 main.ts: Wahl der Ausführungsplattform und Start des Applikationsmoduls	35
1.2.5 index.html und SystemJS: die Anwendung ausführen	35
1.3 Die Blogging-Anwendung	40
1.3.1 Start der Applikation	43
1.3.2 Einige Tipps zur Fehlersuche	44
1.3.3 Die Formularekomponente: Daten aus der View in den Controller übertragen	45
1.3.4 Das Applikationsmodell	47
1.3.5 Darstellung der Liste in der View	50
1.3.6 Modularisierung der Anwendung	53
1.4 Zusammenfassung und Ausblick	55

2 Das Angular-CLI: professionelle Projektorganisation für Angular-Projekte 57

2.1 Das Angular-CLI installieren	58
2.2 ng new: Ein Grundgerüst für die Applikation erstellen	58
2.2.1 Konfigurationsoptionen für die Projekt-Generierung	60

2.2.2	Das generierte Projekt im Detail – Unterschiede zum Kickstart-Beispiel	61
2.3	ng init: Ihr Projekt auf die neueste Angular-CLI-Version updaten	65
2.4	ng serve: die Anwendung starten	66
2.4.1	Die Proxy-Konfiguration	67
2.5	npm start: Start über die lokale CLI-Version	68
2.6	ng generate: Komponenten generieren	69
2.6.1	Konfigurationsoptionen bei der Komponentengenerierung	70
2.6.2	Weitere Generatoren	71
2.7	ng lint: Linting und der Angular-Style-Guide	72
2.8	Komponenten- und Ende-zu-Ende-Tests ausführen	74
2.8.1	ng test – Unit- und Komponententests ausführen	74
2.8.2	ng e2e – Ende-zu-Ende-Tests ausführen	75
2.9	CSS-Präprozessoren verwenden	77
2.10	Drittanbieter-Bibliotheken einbinden	77
2.10.1	Bibliotheken über die index.html einbinden	78
2.11	ng build: deploybare Builds erstellen	79
2.12	Build-Targets und Environments: Konfiguration unterschiedlicher Build- und Ausführungsumgebungen	79
2.12.1	Eigene Environments hinzufügen	81
2.13	Der AOT-Modus	81
2.14	Zusammenfassung und Ausblick	82

3 Komponenten und Templating: der Angular-Sprachkern 85

3.1	Etwas Theorie: der Angular-Komponentenbaum	85
3.2	Selektoren: vom DOM-Element zur Angular-Komponente	89
3.2.1	Tag-Selektoren	89
3.2.2	Attribut-Selektoren	89
3.2.3	Klassen-Selektoren	91
3.2.4	not()-Selektoren	91
3.2.5	Verknüpfung von Selektoren	91
3.3	Die Templating-Syntax: Verbindung zwischen Applikationslogik und Darstellung	92
3.3.1	Fallbeispiel: Timepicker-Komponente	92

3.3.2	Property-Bindings	93
3.3.3	Sonderfälle: Attribute, Klassen und Styles setzen	96
3.3.4	Interpolation – Darstellung von Werten im View	98
3.3.5	Event-Bindings	99
3.3.6	Two-Way-Data-Bindings mit NgModel	104
3.3.7	Lokale Template-Variablen	106
3.3.8	Die *-Templating-Microsyntax – neue DOM-Elemente dynamisch einfügen	107
3.3.9	Templating-Syntax-Spickzettel	111
3.4	Komponentenschnittstellen definieren: von der einzelnen Komponente zur vollständigen Applikation	112
3.4.1	Input-Bindings – Werte in Ihre Komponenten hineinreichen	113
3.4.2	Output-Bindings – andere Komponenten über Datenänderungen informieren	117
3.4.3	Two-Way-Data-Bindings – syntaktischer Zucker für Ihre Komponentenschnittstelle	120
3.4.4	Auf Änderungen von Bindings reagieren	121
3.4.5	Lokale Komponentenvariablen – Zugriff auf die API Ihrer Kind-Elemente im HTML-Code	123
3.5	ViewChildren: Zugriff auf Kind-Elemente aus der Komponenteklasse ...	123
3.6	Content-Insertion: dynamische Komponentenhierarchien erstellen	126
3.6.1	Einfachen HTML-Code injizieren	126
3.6.2	ContentChildren: Erzeugung von dynamischen Komponenten- bäumen am Beispiel einer Tabs-Komponente	132
3.7	Der Lebenszyklus einer Komponente	136
3.7.1	Der Konstruktor: Instanziierung der Komponente	139
3.7.2	ngOnInit – Initialisierung der eigenen Komponente	140
3.7.3	ngOnChanges – auf Änderungen reagieren	141
3.7.4	ngAfterContentInit – auf die Initialisierung von Content-Children reagieren	142
3.7.5	ngAfterViewInit – auf die Initialisierung von ViewChildren reagieren	142
3.7.6	ngOnDestroy – Aufräumarbeiten vornehmen	143
3.7.7	ngAfterContentChecked, ngAfterViewChecked – den ChangeDetection-Mechanismus verfolgen	144
3.7.8	ngDoCheck – den ChangeDetection-Mechanismus verändern	145
3.8	Zusammenfassung und Ausblick	147

4 Direktiven: Komponenten ohne eigenes Template 149

4.1	ElementRef und Renderer: Manipulation von DOM-Eigenschaften eines Elements	150
4.1.1	Die Renderer-Klasse: das native Element plattformunabhängig manipulieren	153
4.2	HostBinding und HostListener: Auslesen und Verändern von Host-Eigenschaften und -Events	154
4.2.1	Kanonisches Host-Binding	156
4.3	Anwendungsfall: Einbinden von Drittanbieter-Bibliotheken	157
4.3.1	Two-Way-Data-Binding für die Slider-Komponente	159
4.4	Anwendungsfall: Accordion-Direktive – mehrere Kind-Komponenten steuern	161
4.5	exportAs: Zugriff auf die Schnittstelle einer Direktive	164
4.6	Zusammenfassung und Ausblick	166

5 Fortgeschrittene Komponentenkonzepte 169

5.1	Styling von Angular-Komponenten	169
5.1.1	Styles an der Komponente definieren	170
5.1.2	ViewEncapsulation – Strategien zum Kapseln Ihrer Styles	171
5.2	TemplateRef und NgTemplateOutlet: dynamisches Austauschen von Komponenten-Templates	178
5.2.1	NgFor mit angepassten Templates verwenden	179
5.2.2	NgTemplateOutlet: zusätzliche Templates an die Komponente übergeben	182
5.3	ViewContainerRef und ComponentFactory: Komponenten zur Laufzeit hinzufügen	186
5.3.1	ViewContainerRef und entryComponents: Komponenten zur Laufzeit hinzufügen	187
5.3.2	ComponentRef: Interaktion mit der dynamisch erzeugten Komponente	190
5.3.3	Komponenten an einer bestimmten Stelle einfügen	191
5.3.4	Komponenten innerhalb des ViewContainers verschieben und löschen	191
5.3.5	createEmbeddedView: Templates dynamisch einbinden	193

5.4	ChangeDetection-Strategien: Performance-Boost für Ihre Applikation	196
5.4.1	Die Beispielapplikation	197
5.4.2	Veränderungen des Applikationsstatus	200
5.4.3	ChangeDetection-Strategien: Optimierung des Standard- verhaltens	203
5.4.4	ChangeDetectorRef: die vollständige Kontrolle über den ChangeDetector	206
5.5	Zusammenfassung und Ausblick	210

6 Standarddirektiven und Pipes: wissen, was das Framework an Bord hat 213

6.1	Standarddirektiven	214
6.1.1	NgIf: Elemente abhängig von Bedingungen darstellen	214
6.1.2	NgSwitch: Switch-Case-Verhalten implementieren	215
6.1.3	NgClass: CSS-Klassen dynamisch hinzufügen und entfernen	216
6.1.4	NgStyle: das style-Attribut manipulieren	220
6.1.5	NgFor: Komfortabel über Listen iterieren	221
6.1.6	NgNonBindable-Auswertung durch die Templating-Syntax verhindern	225
6.2	Pipes: Werte vor dem Rendern transformieren	226
6.2.1	UpperCasePipe und LowerCasePipe: Strings transformieren	227
6.2.2	Die SlicePipe: nur bestimmte Bereiche von Arrays und Strings darstellen	227
6.2.3	Die JSON-Pipe: JavaScript-Objekte als String ausgeben	230
6.2.4	DecimalPipe: Zahlenwerte formatieren	231
6.2.5	Kurzexkurs: Lokalisierbare Pipes – Werte der aktuellen Sprache entsprechend formatieren	231
6.2.6	DatePipe: Datums- und Zeitwerte darstellen	233
6.2.7	Percent- und CurrencyPipe: Prozent- und Währungswerte formatieren	235
6.2.8	Die AsyncPipe: auf asynchrone Werte warten	237
6.2.9	Pipes im Komponentencode verwenden	239
6.2.10	Eigene Pipes implementieren	241
6.2.11	Pure vs. Impure Pipes: Pipe, ändere dich!	244
6.3	Zusammenfassung und Ausblick	247

7 Services und Dependency-Injection: lose Kopplung für Ihre Business-Logik 249

7.1	Grundlagen der Dependency-Injection	250
7.2	Services in Angular-Applikationen	252
7.3	Das Angular-Dependency-Injection-Framework	253
7.3.1	Injector- und Provider-Konfiguration: das Herz der DI	254
7.3.2	Weitere Provider-Formen	257
7.3.3	Opaque-Tokens: kollisionsfreie Definition von DI-Schlüsseln	259
7.4	Verwendung des DI-Frameworks in Angular-Applikationen	261
7.4.1	Der Injector-Baum	261
7.4.2	Registrierung von globalen Services: der UserService	262
7.4.3	Registrieren von komponentenbezogenen Services: MusicSearchService und VideoSearchService	265
7.5	Injection by Type: Vereinfachungen für TypeScript-Nutzer	269
7.5.1	Den @Inject-Decorator vermeiden	269
7.5.2	Der @Injectable-Decorator: TypeScript-optimierte Injections für Services	270
7.5.3	Member-Injection – automatische Erzeugung von Membervariablen	271
7.6	Sichtbarkeit und Lookup von Dependencys	272
7.6.1	Sichtbarkeit von Providern beschränken	272
7.6.2	Den Lookup von Abhängigkeiten beeinflussen	275
7.7	Zusammenfassung und Ausblick	279

8 Template-Driven Forms: einfache Formulare auf Basis von HTML 281

8.1	Grundlagen zu Formularen: template-driven oder reaktiv?	283
8.2	Das erste Formular: Übersicht über die Forms-API	283
8.2.1	Einbinden des Formular-Moduls	284
8.2.2	Implementierung des ersten Formular-Prototyps	284
8.2.3	NgModel, NgForm, FormControl und FormGroup: die wichtigsten Bestandteile der Forms-API	288
8.3	NgModel im Detail: Two-Way-Data-Binding oder nicht?	289
8.3.1	One-Way-Binding mit NgModel	290

8.4	Kurzexkurs: Verwendung von Interfaces für die Definition des Applikationsmodells	294
8.5	Weitere Eingabeelemente	296
8.5.1	Auswahllisten	296
8.5.2	Checkboxen	301
8.5.3	Radio-Buttons	301
8.6	Verschachtelte Eigenschaften definieren	303
8.6.1	Verschachtelte Eigenschaften mit NgModelGroup	303
8.7	Validierungen	304
8.7.1	Vom Framework mitgelieferte Validierungsregeln	305
8.7.2	Validierungen im Formular darstellen	305
8.7.3	Implementierung einer generischen ShowError-Komponente	308
8.7.4	Eigene Validierungsregeln definieren	312
8.7.5	Asynchrone Validierungen	315
8.7.6	Feldübergreifende Validierungen	319
8.8	Implementierung der Tags-Liste: wiederholbare Strukturen mit Template-Driven Forms	321
8.9	Zusammenfassung und Ausblick	324
9	Model-Driven Forms: Formulare dynamisch in der Applikationslogik definieren	327

9.1	Aktivierung von Model-Driven Forms für Ihre Applikation	328
9.2	Das Task-Formular im modellgetriebenen Ansatz	328
9.2.1	Definition des Formulars im TypeScript-Code	329
9.2.2	Verknüpfung des Formulars mit dem HTML-Code	330
9.2.3	FormArray im Detail: Wiederholbare Strukturen definieren	333
9.2.4	Verbindung des Formulars mit dem Applikationsmodell	337
9.2.5	Der FormBuilder – komfortable Definition von Formularen	341
9.2.6	Validierungen von Model-Driven Forms	342
9.3	Formulare und Kontrollelemente auf Änderungen überwachen	349
9.4	Fallbeispiel: Umfragebogen – Formulare komplett dynamisch definieren	350
9.5	Die Forms-API im Überblick	357
9.5.1	AbstractControl: die Basis für alle Forms-API-Basisklassen	357
9.5.2	FormControl: Eigenschaften und Methoden für einzelne Kontrollelemente	359

9.5.3	FormGroup: API zur Verwaltung von Gruppen und Formularen	359
9.5.4	FormArray: wiederholbare Strukturen managen	360
9.6	Zusammenfassung und Ausblick	360

10 Routing: Navigation innerhalb der Anwendung 363

10.1	Project-Manager: die Beispielanwendung	364
10.2	Die erste Routenkonfiguration: das Routing-Framework einrichten	365
10.3	Location-Strategien: »schöne URLs« vs. »Routing ohne Server-Konfiguration«	370
10.3.1	PathLocation-Strategie – schöne URLs	370
10.3.2	HashLocation-Strategie – Routing ohne aufwendige Konfiguration	372
10.4	ChildRoutes: verschachtelte Routenkonfigurationen erstellen	373
10.4.1	Componentless-Routes: Routendefinitionen ohne eigene Komponente	376
10.4.2	Relative Links	377
10.5	RouterLinkActive: Styling des aktiven Links	379
10.5.1	RouterLinkActiveOptions: Exakt oder nicht?	379
10.6	Routing-Parameter: dynamische Adresszeilenparameter auswerten	381
10.6.1	Pfad-Parameter: Pflicht-Parameter in Routen definieren	382
10.6.2	Snapshots – statisch auf Parameterwerte zugreifen	384
10.6.3	Matrix-Parameter: optionale Parameter	385
10.6.4	Query-Parameter: optionale Parameter unabhängig vom Segment definieren	389
10.6.5	Fragmentbezeichner	390
10.7	Aus der Anwendungslogik heraus navigieren	392
10.7.1	Die navigate-Methode: Navigation auf Basis der Routing-DSL	392
10.7.2	navigateByUrl: Navigation auf Basis von URLs	393
10.8	Routing-Guards: Routen absichern und die Navigation generisch beeinflussen	394
10.8.1	CanActivate – Routen absichern	395
10.8.2	CanDeactivate – das Verlassen einer Route verhindern	398
10.9	Redirects und Wildcard-URLs	400
10.9.1	Absolute Redirects	400
10.9.2	Relative Redirects	401
10.9.3	Wildcard-URLs – Platzhalter-Routen definieren	402

10.10 Data: statische Metadaten an Routen hinterlegen	402
10.11 Resolve: dynamische Daten über den Router injizieren	403
10.11.1 Verwendung einer resolve-Funktion anstelle einer Resolver-Klasse	405
10.12 Der Title-Service: den Seitentitel verändern	406
10.13 Router-Tree und Router-Events: generisch auf Seitenwechsel reagieren	408
10.13.1 Der events-Stream: bei Seitenwechseln informiert werden	408
10.13.2 Der Router-Tree: den aktuellen Router-Baum durchlaufen	409
10.14 Location: direkte Interaktion mit der Adresszeile des Browsers	411
10.15 Mehrere RouterOutlets: maximale Flexibilität beim Routing	413
10.15.1 Zusätzliche Outlets – ein Chat-Fenster einblenden	413
10.15.2 Komplexere Outlet-Konfigurationen: eine Task-Schnellansicht	416
10.16 Zusammenfassung und Ausblick	419

11 HTTP: Anbindung von Angular-Applikationen an einen Webserver 421

11.1 Die Server-Applikation	422
11.1.1 Die json-server-Bibliothek	423
11.2 Das Angular-HTTP-Modul verwenden	426
11.3 Der erste GET-Request: Grundlagen zur HTTP-API	427
11.3.1 Auf Fehler reagieren	430
11.4 Asynchrone Service-Schnittstellen modellieren: Anpassung des TaskService	431
11.4.1 Observables statt Callbacks – Daten reaktiv verwalten	432
11.5 Die AsyncPipe: noch eleganter mit asynchronen Daten arbeiten	434
11.6 URLSearchParams: elegant dynamische Suchen definieren	435
11.7 POST, PUT, DELETE, PATCH und HEAD: Verwendung der weiteren HTTP-Methoden	438
11.7.1 HTTP-POST: neue Tasks anlegen	438
11.7.2 HTTP-PUT: bestehende Tasks editieren	439
11.7.3 HTTP-DELETE: Tasks löschen	441
11.7.4 Generische Anfragen: die »request«-Methode	442
11.7.5 HTTP-PATCH: Tasks partiell verändern	445
11.7.6 HTTP-HEAD: der kleine Bruder von GET	446

11.8	JSONP	447
11.8.1	Der Angular-Jsonp-Service	448
11.9	Die Http-API im Detail: Überblick über die wichtigsten Klassen des Frameworks	451
11.9.1	Der Http-Service	451
11.9.2	Das RequestOptionsArgs-Interface	452
11.9.3	Die Headers-Klasse	453
11.9.4	Die Response-Klasse	454
11.10	Zusammenfassung und Ausblick	454
12	Reaktive Architekturen mit RxJS	457
<hr/>		
12.1	Kurzeinführung in RxJS	458
12.1.1	Observable.create und Observer-Functions – die Kernelemente der reaktiven Programmierung	458
12.1.2	Subscriptions und Disposing-Functions – Observables sauber beenden	460
12.1.3	Subjects: Multicast-Funktionalität auf Basis von RxJS	463
12.2	Implementierung einer Typeahead-Suche	465
12.2.1	mergeMap: verschachtelte Observables verbinden	469
12.2.2	switchMap – nur die aktuellsten Ergebnisse verarbeiten	470
12.2.3	merge – mehrere Streams vereinen	471
12.3	Reaktive Datenarchitekturen in Angular-Applikationen	474
12.3.1	Shared Services – der erste Schritt in die richtige Richtung	476
12.3.2	Die neue Datenarchitektur: »Push« statt »Pull«	479
12.3.3	Umsetzung des neuen Konzepts in Angular	481
12.3.4	Anbindung der TaskListComponent an den Store	488
12.3.5	Der »In Bearbeitung«-Zähler	489
12.4	Anbindung von WebSockets zur Implementierung einer Echtzeitanwendung	491
12.4.1	Der WebSocket-Server	491
12.4.2	Integration von Socket.IO in die Anwendung	493
12.4.3	Verwendung von Socket.IO im TaskService	494
12.5	ChangeDetectionStrategy.OnPush: Performance-Schub durch die reaktive Architektur	496
12.6	Zusammenfassung und Ausblick	497

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

501

13.1 Karma und Jasmine: Grundlagen zu Unit- und Komponententests in Angular-Anwendungen	502
13.1.1 Karma einrichten	502
13.2 Der erste Unit-Test: einfache Klassen und Funktionen testen	506
13.2.1 Die Testausführung starten	508
13.2.2 Nur bestimmte Tests ausführen	510
13.3 Isolierte Komponenten testen: Grundlagen zu Komponententests mit dem Angular-Testing-Framework	511
13.3.1 Die zu testende Komponente	512
13.3.2 TestBed, ComponentFixture & Co – Konfiguration des Testmoduls und Erzeugung von Testkomponenten	513
13.4 Mocks und Spies: Komponenten mit Abhängigkeiten testen	516
13.4.1 Eigene Mocks für die Simulation von Services bereitstellen	518
13.4.2 inject – Zugriff auf die im Testkontext vorhandenen Services	520
13.4.3 TestBed.get: alternativer Zugriff auf die Services aus dem Ausführungskontext	521
13.4.4 Spies: ausgehende Aufrufe überwachen und auswerten	522
13.5 Services und HTTP-Backends testen	524
13.6 Formulare testen	529
13.6.1 Model-Driven Forms: Formulare direkt über die API testen	530
13.6.2 Template-Driven Forms: generierte Formulare über die Forms-API testen	531
13.6.3 Formulare über die Oberfläche testen	534
13.7 Direktiven und ngContent-Komponenten testen	536
13.7.1 overrideComponent und compileComponents: Komponenten-Templates für den Test überschreiben	537
13.8 async und fakeAsync: mehr Kontrolle über asynchrone Tests	539
13.8.1 async: automatisch auf asynchrone Aufrufe warten	539
13.8.2 fakeAsync: komplexere asynchrone Szenarien steuern	540
13.9 Routing-Funktionalität testen	541
13.9.1 Manipulation von Router-Diensten im Komponententest	542
13.9.2 Ausführung echter Navigationsvorgänge	543
13.10 Zusammenfassung und Ausblick	545

14 Integrationstests mit Protractor 547

14.1 Start der Tests und Konfiguration von Protractor	548
14.1.1 Installation und Konfiguration von Protractor	549
14.2 Anpassung der Applikationskonfiguration über Environments	550
14.3 Das browser-Objekt und Locators:	
Übersicht über die Kernbestandteile von Protractor	552
14.3.1 browser – die Schnittstelle zur Interaktion mit dem Webbrowser	553
14.3.2 element und by – Protractor-Locators in Aktion	554
14.3.3 Promises bei der Arbeit mit der Protractor-API	557
14.4 Page-Objects: Trennung von Testlogik und technischen Details	558
14.5 Formulare und Alert-Boxen testen: der Edit-Task-Test	561
14.6 Seitenübergreifende Workflows testen	564
14.6.1 ExpectedConditions: komfortabel auf das Eintreten von Bedingungen warten	566
14.6.2 Zahlenwerte vergleichen – manuelle Auswertung der Promise.then-Rückgabewerte	567
14.7 Debugging von Protractor-Tests	568
14.8 Screenshots anfertigen	570
14.8.1 Nach jedem Test einen Screenshot aufnehmen	571
14.9 Zusammenfassung	573

15 NgModule und Lazy-Loading: Modularisierung Ihrer Anwendungen 575

15.1 Feature-Modules: Teilbereiche der Applikation kapseln	576
15.1.1 Feature-Modules – den Aufgabenbereich modularisieren	577
15.1.2 Das Common-Module: Import von Angular-Standard- funktionalität	579
15.1.3 Routing in Feature-Modules – die Routing-Konfiguration modularisieren	579
15.1.4 Anpassungen am Hauptmodul und Integration des Feature-Modules	580
15.2 Shared-Modules: gemeinsam genutzte Funktionalität kapseln	584
15.2.1 Boilerplate-Code durch Shared-Modules vermeiden	587
15.2.2 Gemeinsame Services über Shared-Modules verwalten	589

15.3	Lazy-Loading von Applikationsbestandteilen	591
15.3.1	Preloading von Feature-Modulen: das Beste aus beiden Welten	593
15.4	entryComponents: dynamisch geladene Komponenten registrieren	594
15.5	Zusammenfassung und Ausblick	595

16 Der Angular-Template-Compiler, Ahead-of-time Compilation und Tree-Shaking 597

16.1	Grundlagen zum Angular-Template-Compiler	598
16.2	Der Ahead-of-time Compilation-Modus: Leistungsschub für Ihre Anwendung	600
16.2.1	Den Template-Compiler ausführen	601
16.2.2	Start der Anwendung über die statische Browser-Plattform	602
16.3	Tree-Shaking der Anwendung mit Rollup	602
16.4	Implementierungsregeln beim Einsatz von AOT	605
16.4.1	Konsistenz zwischen HTML- und Komponentencode	605
16.4.2	Kein Einsatz von privaten Membervariablen im Zusammenspiel mit Templates	606
16.4.3	Verzicht auf Inline-Funktionen	607
16.5	Zusammenfassung und Ausblick	608

17 ECMAScript 5: Angular-Anwendungen auf klassische Weise entwickeln 611

17.1	Hello ES5	612
17.2	Kalender und Timepicker in ES5	616
17.2.1	Immediately-Invoked Function Expressions zur Simulation von Modulen	616
17.2.2	Template- und Style-URLs	618
17.2.3	Methoden, Bindings & Co.	618
17.2.4	Querys: Kind-Komponenten injizieren	620
17.2.5	Start der Anwendung	621
17.2.6	Services- und HTTP-Anbindung	622
17.3	Zusammenfassung	626

18 Internationalisierung: mehrsprachige Angular-Anwendungen implementieren 627

18.1 Die Grundlagen des i18n-Frameworks	628
18.1.1 Bestimmen Sie die Sprache der Anwendung	629
18.2 ng-xi18n: automatische Generierung der Message-Datei	632
18.2.1 Exkurs: die Übersetzungen mit git verwalten	635
18.3 Description und Meaning: Metadaten für Übersetzer übergeben	636
18.4 Weitere Übersetzungstechniken	637
18.4.1 Attribute (und Input-Bindings) übersetzen	637
18.4.2 Mehrere parallele Knoten übersetzen	638
18.5 Pluralisierung und geschlechterspezifische Texte	639
18.5.1 Pluralisierung: Texte abhängig vom Zahlenwert einer Variablen	639
18.5.2 Pluralisierungen übersetzen	642
18.5.3 i18nSelectPipe: geschlechterspezifische Texte festlegen	643
18.6 Statisch übersetzte Applikationen im AOT-Modus generieren	646
18.7 Zusammenfassung und Ausblick	649

19 Das Animations-Framework: Angular-Anwendungen animieren 651

19.1 Die erste Animation: Grundlagen zum Animation-Framework	652
19.1.1 Bidirektionale Transitionen	655
19.2 void und *: spezielle States zum Hinzufügen und Entfernen von DOM-Elementen	656
19.2.1 :enter und :leave – Shortcuts für das Eintreten und Verlassen des DOM	658
19.3 Animationen in Verbindung mit automatisch berechneten Eigenschaften	659
19.4 Animation-Lifecycles: auf den Start und das Ende von Animationen reagieren	661
19.5 Animation von Routing-Vorgängen	662
19.5.1 Gemeinsam genutzte Animationen auslagern	664
19.5.2 Lifecycle-Hooks für Routing-Animationen	664
19.6 Keyframes: Definition von komplexen, mehrstufigen Animationen	665

19.7 Styling von Komponenten, die in Animationen verwendet werden	667
19.8 Groups und Sequences: mehrere Animationen kombinieren	668
19.8.1 group: Animationsschritte parallel ausführen	669
19.8.2 sequence: Animationsschritte nacheinander ausführen	670
19.8.3 Kombination von sequence und group	670
19.9 Zusammenfassung	671
19.10 Schlusswort	672

Anhang

A ECMAScript 2015	675
B Typsicheres JavaScript mit TypeScript	727
Index	775