



Kostenloses eBook

LERNEN

Angular 2

Free unaffiliated eBook created from
Stack Overflow contributors.

#angular2

Inhaltsverzeichnis

Über.....	1
Kapitel 1: Erste Schritte mit Angular 2	2
Bemerkungen.....	2
Versionen.....	2
Examples.....	3
Installieren Sie angle2 mit dem Winkelcli.....	3
Voraussetzungen:	3
Ein neues Projekt einrichten	3
Hinzufügen zu einem vorhandenen Projekt	4
Projekt lokal ausführen	4
Komponenten, Richtlinien, Pipes und Services generieren	4
Erste Schritte mit Angular 2 ohne eckige Kli.....	6
Schritt 1.....	6
Schritt 2.....	6
Schritt 3.....	8
Schritt 5.....	9
Schritt 6.....	9
Schritt 7.....	10
Schritt 8.....	11
Was jetzt?.....	11
Synchronisierung von Visual Studios mit NPM- und NODE-Updates.....	11
Durchkommen dieses lästigen Firmenvertreter.....	12
Erste Schritte mit Angular 2 mit node.js / expressjs Backend (http-Beispiel enthalten).....	13
Voraussetzungen	13
Roadmap	13
Schritt 1.....	13
Schritt 2.....	13
Schritt 3.....	14
Lassen Sie uns in Angular 4 eintauchen!.....	19

Kapitel 2: Aktualisierungen	24
Examples	24
Aktualisierungen der Typisierung in folgenden Fällen: Typisierung WARN ist veraltet	24
Kapitel 3: Angular - ForLoop	25
Syntax	25
Bemerkungen	25
Examples	25
Winkel für 2-Schleife	25
NgFor - Markup für Schleife	26
* ngFür in den Tabellenzeilen	26
* ngFür mit Komponente	26
* ngFür X Anzahl der Elemente pro Zeile	27
Kapitel 4: Angular 2 Änderungserkennung und manuelle Auslösung	28
Examples	28
Grundlegendes Beispiel	28
Kapitel 5: Angular 2 datengesteuerte Formulare	30
Bemerkungen	30
Examples	31
Datengesteuertes Formular	31
Kapitel 6: Angular 2 Forms Update	33
Bemerkungen	33
Examples	33
Einfaches Passwortänderungsformular mit Multi Control Validation	33
pw-change.template.html	33
pw-change.component.ts	34
pw-validators.ts	34
Winkel 2: Vorlagengesteuerte Formulare	35
Winkel 2-Formular - Benutzerdefinierte E-Mail- / Passwort-Überprüfung	36
Winkel 2: Reaktive Formen (auch als Model-Driven Forms bezeichnet)	37
Registrierungsformular.Komponente.ts	37
Anmeldeformular.html	38
Angular 2 Forms (Reactive Forms) mit Registrierungsformular und Bestätigung der Passworte	38

app.module.ts.....	38
app.component.ts.....	38
app.component.html.....	39
validators.ts.....	39
Angular2 - Formularersteller.....	40
Kapitel 7: Angular RXJS Subjects und Observables mit API-Anforderungen.....	42
Bemerkungen.....	42
Examples.....	42
Grundanforderung.....	42
API-Anforderungen kapseln.....	42
Warten Sie auf mehrere Anfragen.....	43
Kapitel 8: Angular2 Animationen.....	45
Einführung.....	45
Examples.....	45
Grundanimation - Übergibt ein Element zwischen zwei Zuständen, die von einem Modellattribu.....	45
Kapitel 9: Angular2 Benutzerdefinierte Validierungen.....	47
Parameter.....	47
Examples.....	47
Beispiele für benutzerdefinierte Validatoren:.....	47
Validatoren im FormBuilder verwenden.....	47
get / set FormBuilder steuert die Parameter.....	48
Kapitel 10: Angular2 CanActivate.....	49
Examples.....	49
Angular2 CanActivate.....	49
Kapitel 11: Angular2 Datenbindung.....	50
Examples.....	50
@Eingang().....	50
Übergeordnete Komponente: Benutzerlisten initialisieren.....	50
Kapitel 12: Angular2 Eingang () Ausgang ().....	52
Examples.....	52
Eingang().....	52

Übergeordnete Komponente: Benutzerlisten initialisieren.....	52
Einfaches Beispiel für Eingabeeigenschaften.....	53
Kapitel 13: Angular2 In Memory-Web-API.....	54
Bemerkungen.....	54
Examples.....	54
Grundeinstellung.....	54
Einrichten mehrerer Test-API-Routen.....	55
Kapitel 14: Angular2 mit Webpack.....	57
Examples.....	57
Angular 2 Webpack-Setup.....	57
Kapitel 15: Angular2 stellt App vor dem Bootstrap externe Daten bereit.....	61
Einführung.....	61
Examples.....	61
Über Abhängigkeitsinjektion.....	61
Kapitel 16: Angular-cli.....	62
Einführung.....	62
Examples.....	62
Erstellen Sie eine leere Angular2-Anwendung mit angle-cli.....	62
Komponenten, Richtlinien, Pipes und Services generieren.....	62
Hinzufügen von Drittanbieter-Bibliotheken.....	62
bauen mit eckig-cli.....	63
Neues Projekt mit scss / sass als Stylesheet.....	63
Legen Sie das Garn als Standardpaketmanager für @ angle / cli fest.....	63
Bedarf.....	64
Kapitel 17: Angulares Materialdesign.....	65
Examples.....	65
Md2Select.....	65
Md2Tooltip.....	65
Md2Toast.....	65
Md2Datepicker.....	66
Md2Accordion und Md2Collapse.....	66
Kapitel 18: Animation.....	68

Examples.....	68
Übergang zwischen Nullzuständen.....	68
Animieren zwischen mehreren Zuständen.....	68
Kapitel 19: AOT-Compilierung mit Angular 2.....	70
Examples.....	70
1. Installieren Sie die Abhängigkeiten von Angular 2 mit dem Compiler.....	70
2. Fügen Sie der Datei "tsconfig.json" die Option "angleCompilerOptions" hinzu.....	70
3. Führen Sie den Winkelcompiler ngc aus.....	70
4. Ändern Sie die Datei "main.ts", um den Browser NgFactory und die statische Plattform zu.....	70
Warum brauchen wir Zusammenstellung, Zusammenstellung von Ereignissen und Beispiel?.....	71
AoT-Kompilierung mit Angular CLI verwenden.....	72
Kapitel 20: Attributanweisungen, um den Wert von Eigenschaften auf dem Hostknoten mithilfe ..	73
Examples.....	73
@HostBinding.....	73
Kapitel 21: Beispiel für Routen wie / route / subroute für statische URLs.....	74
Examples.....	74
Grundlegendes Routenbeispiel mit Unterroutenbaum.....	74
Kapitel 22: Beispiele für erweiterte Komponenten.....	75
Bemerkungen.....	75
Examples.....	75
Bildauswahl mit Vorschau.....	75
Tabellenwerte über die Eingabe herausfiltern.....	76
Kapitel 23: benutzerdefinierte ngx-bootstrap datepicker + input.....	78
Examples.....	78
benutzerdefinierte ngx-bootstrap datepicker.....	78
Kapitel 24: Bootstrap Leeres Modul in Winkel 2.....	81
Examples.....	81
Ein leeres Modul.....	81
Ein Modul mit Vernetzung im Webbrowser.....	81
Bootstrapping deines Moduls.....	81
Anwendungsstammmodul.....	82
Statisches Bootstrapping mit Factory-Klassen.....	82

Kapitel 25: Brute Force Upgrade	83
Einführung.....	83
Bemerkungen.....	83
Examples.....	83
Gerüstbau eines neuen CLI-Projekts.....	83
Kapitel 26: CRUD in Angular2 mit Restful-API	84
Syntax.....	84
Examples.....	84
Aus einer Restful-API in Angular2 lesen.....	84
Kapitel 27: Debuggen der Angular2-Typoskriptanwendung mit Visual Studio Code	86
Examples.....	86
Launch.json-Setup für Ihren Arbeitsbereich.....	86
Kapitel 28: Dienste und Abhängigkeitsinjektion	88
Examples.....	88
Beispieldienst.....	88
Beispiel mit Promise.resolve.....	89
Testen eines Dienstes.....	90
Kapitel 29: Dropzone in Angular2	93
Examples.....	93
Abwurfgebiet.....	93
Kapitel 30: Dynamisches Hinzufügen von Komponenten mithilfe von ViewContainerRef.createCom	95
Examples.....	95
Eine Wrapper-Komponente, die dynamische Komponenten deklarativ hinzufügt.....	95
Dynamisches Hinzufügen einer Komponente zu einem bestimmten Ereignis (Klicken).....	96
Dynamisch erstelltes Komponenten-Array für Template-HTML in Angular2.....	97
Kapitel 31: eckiger Redux	101
Examples.....	101
Basic.....	101
Holen Sie sich den aktuellen Stand.....	102
Zustand ändern.....	102
Fügen Sie das Redux-Chrome-Tool hinzu.....	103

Kapitel 32: Ermitteln von Größenänderungsereignissen	104
Examples	104
Eine Komponente, die das Fenster zur Größenänderung des Fensters überwacht	104
Kapitel 33: Erstellen einer Angular-npm-Bibliothek	105
Einführung	105
Examples	105
Minimalmodul mit Serviceklasse	105
Dateistruktur	105
Service und Modul	105
Zusammenstellung	106
NPM-Einstellungen	107
Kontinuierliche Integration	108
Kapitel 34: Erstellen Sie ein Angular 2+ NPM-Paket	110
Einführung	110
Examples	110
Einfachstes Paket	110
Konfigurationsdateien	110
.gitignore	110
.npmignore	110
gulpfile.js	111
index.d.ts	111
index.js	111
package.json	111
dist / tsconfig.json	112
src / angle-x-minimal-npm-package.component.ts	113
src / angle-x-minimal-npm-package.component.html	113
src / angle-x-data-table.component.css	113
src / angle-x-minimal-npm-package.module.ts	113
Bauen und kompilieren	113
Veröffentlichen	114

Kapitel 35: EventEmitter-Dienst	115
Examples	115
Klassenübersicht	115
Klassenkomponente	115
Ereignisse ausstatten	115
Die Veranstaltung abfangen	115
Live-Beispiel	116
Kapitel 36: Fass	117
Einführung	117
Examples	117
Fass verwenden	117
Kapitel 37: Faules Laden eines Moduls	118
Examples	118
Lazy Loading Beispiel	118
Kapitel 38: Funktionsmodule	120
Examples	120
Ein Funktionsmodul	120
Kapitel 39: Geräteprüfung	121
Examples	121
Grundeinheitstest	121
Komponentendatei	121
Kapitel 40: Häufig eingebaute Richtlinien und Dienste	123
Einführung	123
Examples	123
Standortklasse	123
AsyncPipe	123
Anzeige der aktuellen Version von angle2, die in Ihrem Projekt verwendet wird	124
Währungsrohr	124
Kapitel 41: HTTP Interceptor	126
Bemerkungen	126
Examples	126

Einfache Klasse Erweiterung der Http-Klasse von Winkeln.....	126
Verwendung unserer Klasse anstelle von Angulars Http.....	127
Simple HttpClient AuthToken Interceptor (Angular 4.3+).....	128
Kapitel 42: Installieren von Drittanbieter-Plugins mit angle-cli@1.0.0-beta.10.....	129
Bemerkungen.....	129
Examples.....	129
Hinzufügen einer JQuery-Bibliothek zum angle-cli-Projekt.....	129
Fügen Sie eine Drittanbieter-Bibliothek hinzu, die keine Typisierung hat.....	131
Kapitel 43: Komponenten.....	133
Einführung.....	133
Examples.....	133
Eine einfache Komponente.....	133
Vorlagen & Stile.....	133
Vorlage als Dateipfad übergeben.....	133
Vorlage als Inline-Code übergeben.....	134
Ein Array von Dateipfaden übergeben.....	134
Übergeben eines Arrays von Inline-Codes.....	134
Eine Komponente testen.....	134
Komponenten verschachteln.....	136
Kapitel 44: Komponenteninteraktionen.....	137
Syntax.....	137
Parameter.....	137
Examples.....	137
Parent - Child-Interaktion mit den Eigenschaften @Input & @Output.....	137
Parent - Child-Interaktion mit ViewChild.....	138
Bidirektionale Eltern-Kind-Interaktion über einen Dienst.....	139
Kapitel 45: Komponenteninteraktionen.....	142
Einführung.....	142
Examples.....	142
Übergabe der Daten von einem Elternteil an ein Kind mit Eingabe.....	142
Kapitel 46: Konfigurieren der ASP.net Core-Anwendung für die Arbeit mit Angular 2 und Type.....	150
Einführung.....	150

Examples.....	150
Asp.Net Core + Angular2 + Gulp.....	150
[Seed] Asp.Net Core + Angular2 + Gulp auf Visual Studio 2017.....	154
MVC <-> Winkel 2.....	154
Kapitel 47: Lebenszyklus-Haken.....	156
Bemerkungen.....	156
Verfügbarkeit von Veranstaltungen.....	156
Events bestellen.....	156
Lesen Sie weiter.....	156
Examples.....	156
OnInit.....	156
OnDestroy.....	157
OnChange.....	157
AfterContentInit.....	157
AfterContentChecked.....	158
AfterViewInit.....	158
AfterViewChecked.....	159
DoCheck.....	159
Kapitel 48: Module.....	160
Einführung.....	160
Examples.....	160
Ein einfaches Modul.....	160
Verschachteln von Modulen.....	160
Kapitel 49: NgModel testen.....	162
Einführung.....	162
Examples.....	162
Grundtest.....	162
Kapitel 50: ngrx.....	164
Einführung.....	164
Examples.....	164
Vollständiges Beispiel: Anmelden / Abmelden eines Benutzers.....	164

1) Definieren Sie die IUser Schnittstelle	164
2) Deklarieren Sie die Aktionen zur Manipulation des User	165
3) Definieren Sie den Anfangszustand des UserReducer	166
4) Erstellen Sie den Reduzierer UserReducer	166
Zur Erinnerung: Ein Reduzierer muss irgendwann initialisiert werden	167
5) Importieren Sie unseren UserReducer in unser Hauptmodul, um den Store zu erstellen	168
6) Verwenden Sie Daten aus dem Store , um Informationen in unserer Ansicht anzuzeigen	168
Kapitel 51: Optimieren des Renderns mit ChangeDetectionStrategy	171
Examples	171
Standard vs OnPush	171
Kapitel 52: OrderBy Pipe	173
Einführung	173
Examples	173
Das Rohr	173
Kapitel 53: Pfeifen	176
Einführung	176
Parameter	176
Bemerkungen	176
Examples	176
Verkettung von Rohren	176
Kundenspezifische Rohre	177
Eingebaute Pfeifen	177
Angular2 wird mit ein paar eingebauten Rohren geliefert:	177
Beispiel	178
hotel-reservation.component.ts	178
hotel-reservation.template.html	178
Ausgabe	178
Debuggen mit JsonPipe	178
Code	179
Ausgabe	179
Weltweit verfügbares Custom Pipe	179

Benutzerdefiniertes Rohr erstellen.....	179
Async-Werte mit Async-Pipe auspacken.....	180
Bestehendes Rohr verlängern.....	180
Stateful Pipes.....	181
Dynamisches Rohr.....	182
Eine Pfeife testen.....	184
Kapitel 54: Richtlinien.....	185
Syntax.....	185
Bemerkungen.....	185
Examples.....	185
Attribut-Direktive.....	185
Komponente ist eine Direktive mit Vorlage.....	185
Strukturelle Richtlinien.....	185
Zollrichtlinie.....	185
* ngFor.....	186
Direkt in die Zwischenablage kopieren.....	187
Testen einer benutzerdefinierten Direktive.....	188
Kapitel 55: Richtlinien und Komponenten: @Input @Output.....	191
Syntax.....	191
Examples.....	191
Eingabebeispiel.....	191
Angular2 @Input und @Output in einer verschachtelten Komponente.....	192
Angular2 @Input mit asynchronen Daten.....	193
Übergeordnete Komponente mit asynchronem Aufruf an einen Endpunkt.....	193
Untergeordnete Komponente mit asynchronen Daten als Eingabe.....	194
Kapitel 56: Routing.....	196
Examples.....	196
Grundlegendes Routing.....	196
Untergeordnete Routen.....	198
ResolveData.....	199
Routing mit Kindern.....	202
Kapitel 57: Routing (3.0.0+).....	204

Bemerkungen.....	204
Examples.....	204
Bootstrapping.....	204
Router-Steckdose konfigurieren.....	204
Routen ändern (mithilfe von Vorlagen und Anweisungen).....	205
Routen einstellen.....	206
Zugriff auf oder von einer Route steuern.....	207
Funktionsweise von Route Guards.....	207
Route Guard-Schnittstellen.....	207
Synchronous vs. Asynchronous Route Guards.....	207
Synchroner Routenschutz.....	208
Asynchroner Routenschutz.....	208
Wächter zur Routenkonfiguration hinzufügen.....	209
Verwenden Sie Guard im App-Bootstrap.....	209
Resolver und Guards verwenden.....	210
Kapitel 58: Seitentitel.....	212
Einführung.....	212
Syntax.....	212
Examples.....	212
den Seitentitel ändern.....	212
Kapitel 59: Servicemitarbeiter.....	213
Einführung.....	213
Examples.....	213
Füge Service Worker unserer App hinzu.....	213
Kapitel 60: Spott @ ngrx / Store.....	216
Einführung.....	216
Parameter.....	216
Bemerkungen.....	216
Examples.....	217
Beobachter Mock.....	217
Komponententest für Komponente mit Scheinspeicher.....	217

Gerätetest für das Ausspähen von Komponenten im Laden.....	218
Winkel 2 - Scheinbeobachtung (Service + Komponente).....	219
Einfacher Laden.....	222
Kapitel 61: Testen einer Angular 2 App.....	225
Examples.....	225
Installation des Jasmine-Testframeworks.....	225
Installieren.....	225
Überprüfen.....	225
Einrichten von Tests mit Gulp, Webpack, Karma und Jasmine.....	225
HTTP-Dienst testen.....	230
Winkelkomponenten testen - Basic.....	232
Kapitel 62: Umgehen Desinfektion für vertrauenswürdige Werte.....	234
Parameter.....	234
Bemerkungen.....	234
SUPER WICHTIG!.....	234
DIE UNTERSUCHUNG DER SANITIERUNG VERLETZT SIE AUF RISIKO VON XSS (Cross-Site Scripting) un.....	234
Examples.....	234
Bypass-Desinfektion mit Rohren (zur Wiederverwendung von Code).....	234
Kapitel 63: Verwenden Sie in Angular 2 native Webkomponenten.....	238
Bemerkungen.....	238
Examples.....	238
Fügen Sie dem Modul benutzerdefinierte Elemente hinzu.....	238
Verwenden Sie Ihre Webkomponente in einer Vorlage.....	238
Kapitel 64: Verwenden von Drittanbieter-Bibliotheken wie jQuery in Angular 2.....	239
Einführung.....	239
Examples.....	239
Konfiguration mit Winkelkli.....	239
NPM.....	239
Assets-Ordner.....	239
Hinweis.....	239

jQuery in Angular 2.x-Komponenten verwenden.....	239
Kapitel 65: Verwendung von ngfor.....	241
Einführung.....	241
Examples.....	241
Beispiel für ungeordnete Listen.....	241
Komplexeres Template-Beispiel.....	241
Beispiel für aktuelles Zusammenspiel verfolgen.....	241
Angular2-Alias exportierte Werte.....	241
* ngFür mit Rohr.....	242
Kapitel 66: Vorlagen.....	243
Einführung.....	243
Examples.....	243
Angular 2 Vorlagen.....	243
Kapitel 67: Wie benutze ich ngif?.....	245
Einführung.....	245
Syntax.....	245
Examples.....	245
Zeigt eine Lademeldung an.....	245
Warnmeldung zu einer Bedingung anzeigen.....	246
Eine Funktion am Anfang oder Ende der * ngFor-Schleife ausführen Mit * ngIf.....	246
Verwenden Sie * ngIf mit * ngFor.....	246
Kapitel 68: Winkel 2 - Winkelmesser.....	248
Examples.....	248
Testen der Navbar-Routen mit dem Winkelmesser.....	248
Angular2-Winkelmesser - Installation.....	249
Kapitel 69: Winkel-Cli-Testabdeckung.....	251
Einführung.....	251
Examples.....	251
Eine einfache Winkelüberprüfung für den Befehlstest.....	251
Detaillierte grafische Berichterstattung zur Testabdeckung für einzelne Komponenten.....	251
Kapitel 70: Zone.js.....	253
Examples.....	253

Verweis auf NgZone bekommen.....	253
Verwenden Sie NgZone, um mehrere HTTP-Anforderungen auszuführen, bevor Sie die Daten anzei.....	253
Credits.....	255



You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [angular-2](#)

It is an unofficial and free Angular 2 ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Angular 2.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit Angular 2

Bemerkungen

Dieser Abschnitt bietet einen Überblick darüber, wie Angular2+ für den Einsatz in verschiedenen Umgebungen und IDE mit Tools wie die Community entwickelt, installiert und konfiguriert [Winkelcli](#).

Die Vorgängerversion von Angular ist [AngularJS](#) oder auch Angular 1 genannt. Sehen Sie hier die [Dokumentation](#).

Versionen

Ausführung	Veröffentlichungsdatum
4.3.3	2017-08-02
4.3.2	2017-07-26
4.3.1	2017-07-19
4.3.0	2017-07-14
4.2.0	2017-06-08
4.1.0	2017-04-26
4.0.0	2017-03-23
2.3.0	2016-12-08
2.2.0	2016-11-14
2.1.0	2016-10-13
2.0.2	2016-10-05
2.0.1	2016-09-23
2.0.0	2016-09-14
2.0.0-rc.7	2016-09-13
2.0.0-rc.6	2016-08-31
2.0.0-rc.5	2016-08-09
2.0.0-rc.4	2016-06-30

Ausführung	Veröffentlichungsdatum
2.0.0-rc.3	2016-06-21
2.0.0-rc.2	2016-06-15
2.0.0-rc.1	2016-05-03
2.0.0-rc.0	2016-05-02

Examples

Installieren Sie `angle2` mit dem Winkelcli

Dieses Beispiel dient zum schnellen Einrichten von Angular 2 und zum Erstellen eines schnellen Beispielprojekts.

Voraussetzungen:

- [Node.js v4](#) oder höher.
- [npm v3](#) oder höher oder [garn](#) .

Öffnen Sie ein Terminal und führen Sie die Befehle nacheinander aus:

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

oder

```
yarn global add @angular/cli
```

abhängig von Ihrer Wahl des Paketmanagers.

Mit dem vorherigen Befehl wird **@ angle / cli** global installiert, wobei die ausführbare Datei `ng` PATH hinzugefügt wird.

Ein neues Projekt einrichten

Navigieren Sie mit dem Terminal zu einem Ordner, in dem Sie das neue Projekt einrichten möchten.

Führen Sie die Befehle aus:

```
ng new PROJECT_NAME  
cd PROJECT_NAME  
ng serve
```

Nun haben Sie ein einfaches Beispielprojekt, das mit Angular 2 erstellt wurde. Sie können jetzt zu dem in Terminal angezeigten Link navigieren und sehen, was läuft.

Hinzufügen zu einem vorhandenen Projekt

Navigieren Sie zum Stammverzeichnis Ihres aktuellen Projekts.

Führen Sie den Befehl aus:

```
ng init
```

Damit fügen Sie Ihrem Projekt das erforderliche Gerüst hinzu. Die Dateien werden im aktuellen Verzeichnis erstellt. Stellen Sie daher sicher, dass Sie diese in einem leeren Verzeichnis ausführen.

Projekt lokal ausführen

Um Ihre Anwendung während der Ausführung im Browser anzuzeigen und mit ihr zu interagieren, müssen Sie einen lokalen Entwicklungsserver starten, der die Dateien für Ihr Projekt hostet.

```
ng serve
```

Wenn der Server erfolgreich gestartet wurde, sollte eine Adresse angezeigt werden, unter der der Server ausgeführt wird. Normalerweise ist dies:

```
http://localhost:4200
```

Standardmäßig ist dieser lokale Entwicklungsserver mit Hot Module Reloading verbunden. Änderungen an HTML, Typoscript oder css führen dazu, dass der Browser automatisch neu geladen wird (kann aber bei Bedarf deaktiviert werden).

Komponenten, Richtlinien, Pipes und Services generieren

Mit dem Befehl `ng generate <scaffold-type> <name>` (oder einfach `ng g <scaffold-type> <name>`) können Sie automatisch Angular-Komponenten generieren:

```
# The command below will generate a component in the folder you are currently at
ng generate component my-generated-component
# Using the alias (same outcome as above)
ng g component my-generated-component
```

Es gibt verschiedene Arten von Gerüsten, die eckige Kli erzeugen können:

Gerüsttyp	Verwendungszweck
Modul	<code>ng g module my-new-module</code>
Komponente	<code>ng g component my-new-component</code>
Richtlinie	<code>ng g directive my-new-directive</code>
Rohr	<code>ng g pipe my-new-pipe</code>
Bedienung	<code>ng g service my-new-service</code>
Klasse	<code>ng g class my-new-class</code>
Schnittstelle	<code>ng g interface my-new-interface</code>
Enum	<code>ng g enum my-new-enum</code>

Sie können den Typnamen auch durch den ersten Buchstaben ersetzen. Zum Beispiel:

`ng gm my-new-module` zum Erzeugen eines neuen Moduls oder `ng gc my-new-component` zum Erstellen einer Komponente.

Gebäude / Bündelung

Wenn Sie mit dem Erstellen der Angular 2-Web-App fertig sind und die Installation auf einem Webserver wie Apache Tomcat durchführen möchten, müssen Sie nur den Build-Befehl mit oder ohne Produktionsflag ausführen. Die Produktion minimiert den Code und optimiert für eine Produktionseinstellung.

```
ng build
```

oder

```
ng build --prod
```

Suchen Sie dann im Stammverzeichnis des Projekts nach einem Ordner `/dist`, der den Build enthält.

Wenn Sie die Vorteile eines kleineren Produktionspakets wünschen, können Sie auch die Ahead-of-Time-Vorlagenkompilierung verwenden, die den Vorlagen-Compiler aus dem endgültigen Build entfernt:

```
ng build --prod --aot
```

Unit Testing

Angular 2 bietet integrierte Unit-Tests, und jedes von `angle-cli` erstellte Element generiert einen grundlegenden Unit-Test, der erweitert werden kann. Die Unit-Tests werden mit Jasmin

geschrieben und mit Karma ausgeführt. Um den Test zu starten, führen Sie den folgenden Befehl aus:

```
ng test
```

Dieser Befehl führt alle Tests im Projekt aus und führt sie jedes Mal erneut aus, wenn sich eine Quelldatei ändert, unabhängig davon, ob es sich um einen Test oder Code aus der Anwendung handelt.

Weitere Informationen finden Sie auch auf der [Website von angle-cli github](#)

Erste Schritte mit Angular 2 ohne eckige Kli.

Winkel 2.0.0-rc.4

In diesem Beispiel erstellen wir eine "Hallo Welt!" App mit nur einer Wurzelkomponente (`AppComponent`) der Einfachheit halber.

Voraussetzungen:

- [Node.js](#) v5 oder höher
- `npm` v3 oder höher

Anmerkung: Sie können die Versionen überprüfen, indem Sie in der Konsole / im Terminal `node -v` und `npm -v` .

Schritt 1

Erstellen Sie einen neuen Ordner für Ihr Projekt und geben Sie ihn ein. Nennen wir es als `angular2-example` .

```
mkdir angular2-example
cd angular2-example
```

Schritt 2

Bevor wir mit dem Schreiben des App-Codes beginnen, fügen wir die vier folgenden Dateien hinzu: `package.json` , `tsconfig.json` , `typings.json` und `systemjs.config.js` .

Haftungsausschluss: Die gleichen Dateien finden Sie im [offiziellen 5-Minuten-Schnellstart](#) .

`package.json` - Ermöglicht das Herunterladen aller Abhängigkeiten mit `npm` und bietet eine einfache Skriptausführung, um das Leben einfacher Projekte zu erleichtern. (Sie sollten in Betracht ziehen, in Zukunft [Gulp](#) zu verwenden, um Aufgaben zu automatisieren.)

```
{
  "name": "angular2-example",
```

```

"version": "1.0.0",
"scripts": {
  "start": "tsc && concurrently \"npm run tsc:w\" \"npm run lite\" ",
  "lite": "lite-server",
  "postinstall": "typings install",
  "tsc": "tsc",
  "tsc:w": "tsc -w",
  "typings": "typings"
},
"license": "ISC",
"dependencies": {
  "@angular/common": "2.0.0-rc.4",
  "@angular/compiler": "2.0.0-rc.4",
  "@angular/core": "2.0.0-rc.4",
  "@angular/forms": "0.2.0",
  "@angular/http": "2.0.0-rc.4",
  "@angular/platform-browser": "2.0.0-rc.4",
  "@angular/platform-browser-dynamic": "2.0.0-rc.4",
  "@angular/router": "3.0.0-beta.1",
  "@angular/router-deprecated": "2.0.0-rc.2",
  "@angular/upgrade": "2.0.0-rc.4",
  "systemjs": "0.19.27",
  "core-js": "^2.4.0",
  "reflect-metadata": "^0.1.3",
  "rxjs": "5.0.0-beta.6",
  "zone.js": "^0.6.12",
  "angular2-in-memory-web-api": "0.0.14",
  "bootstrap": "^3.3.6"
},
"devDependencies": {
  "concurrently": "^2.0.0",
  "lite-server": "^2.2.0",
  "typescript": "^1.8.10",
  "typings": "^1.0.4"
}
}

```

tsconfig.json - Konfiguriert den TypeScript-Transpiler.

```

{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "moduleResolution": "node",
    "sourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "removeComments": false,
    "noImplicitAny": false
  }
}

```

typings.json - typings.json TypeScript die von uns verwendeten Bibliotheken erkennen.

```

{
  "globalDependencies": {
    "core-js": "registry:dt/core-js#0.0.0+20160602141332",
    "jasmine": "registry:dt/jasmine#2.2.0+20160621224255",
    "node": "registry:dt/node#6.0.0+20160621231320"
  }
}

```



```
}  
}
```

systemjs.config.js - Konfiguriert [SystemJS](#) (Sie können auch [Webpack verwenden](#)).

```
/**  
 * System configuration for Angular 2 samples  
 * Adjust as necessary for your application's needs.  
 */  
(function(global) {  
  // map tells the System loader where to look for things  
  var map = {  
    'app': 'app', // 'dist',  
    '@angular': 'node_modules/@angular',  
    'angular2-in-memory-web-api': 'node_modules/angular2-in-memory-web-api',  
    'rxjs': 'node_modules/rxjs'  
  };  
  // packages tells the System loader how to load when no filename and/or no extension  
  var packages = {  
    'app': { main: 'main.js', defaultExtension: 'js' },  
    'rxjs': { defaultExtension: 'js' },  
    'angular2-in-memory-web-api': { main: 'index.js', defaultExtension: 'js' },  
  };  
  var ngPackageNames = [  
    'common',  
    'compiler',  
    'core',  
    'forms',  
    'http',  
    'platform-browser',  
    'platform-browser-dynamic',  
    'router',  
    'router-deprecated',  
    'upgrade',  
  ];  
  // Individual files (~300 requests):  
  function packIndex(pkgName) {  
    packages['@angular/'+pkgName] = { main: 'index.js', defaultExtension: 'js' };  
  }  
  // Bundled (~40 requests):  
  function packUmd(pkgName) {  
    packages['@angular/'+pkgName] = { main: '/bundles/' + pkgName + '.umd.js',  
    defaultExtension: 'js' };  
  }  
  // Most environments should use UMD; some (Karma) need the individual index files  
  var setPackageConfig = System.packageWithIndex ? packIndex : packUmd;  
  // Add package entries for angular packages  
  ngPackageNames.forEach(setPackageConfig);  
  var config = {  
    map: map,  
    packages: packages  
  };  
  System.config(config);  
})(this);
```

Schritt 3

Installieren wir die Abhängigkeiten durch Eingabe

```
npm install
```

in der Konsole / im Terminal.

Schritt 4

Erstellen Sie die `angular2-example index.html` im Ordner `angular2-example`.

```
<html>
  <head>
    <title>Angular2 example</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- 1. Load libraries -->
    <!-- Polyfill(s) for older browsers -->
    <script src="node_modules/core-js/client/shim.min.js"></script>
    <script src="node_modules/zone.js/dist/zone.js"></script>
    <script src="node_modules/reflect-metadata/Reflect.js"></script>
    <script src="node_modules/systemjs/dist/system.src.js"></script>
    <!-- 2. Configure SystemJS -->
    <script src="systemjs.config.js"></script>
    <script>
      System.import('app').catch(function(err){ console.error(err); });
    </script>
  </head>
  <!-- 3. Display the application -->
  <body>
    <my-app></my-app>
  </body>
</html>
```

Ihre Anwendung wird zwischen den `my-app` Tags gerendert.

Allerdings weiß Angular immer noch nicht, was gerendert werden soll. Um das zu sagen, definieren wir `AppComponent`.

Schritt 5

Erstellen Sie einen Unterordner mit dem Namen `app` in dem wir die Komponenten und [Dienste](#) definieren können, aus denen sich unsere App zusammensetzt. (In diesem Fall enthält es nur den `AppComponent` Code und `main.ts`)

```
mkdir app
```

Schritt 6

Erstellen Sie die Datei `app/app.component.ts`

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
```

```

template: `
  <h1>{{title}}</h1>
  <ul>
    <li *ngFor="let message of messages">
      {{message}}
    </li>
  </ul>
`
})
export class AppComponent {
  title = "Angular2 example";
  messages = [
    "Hello World!",
    "Another string",
    "Another one"
  ];
}

```

Was ist los? Zunächst importieren wir den `@Component` Dekorator, mit dem wir Angular das HTML-Tag und die Vorlage für diese Komponente `@Component` . Dann erstellen wir die Klasse `AppComponent` mit den Variablen `title` und `messages` , die wir in der Vorlage verwenden können.

Nun sehen wir uns diese Vorlage an:

```

<h1>{{title}}</h1>
<ul>
  <li *ngFor="let message of messages">
    {{message}}
  </li>
</ul>

```

Wir zeigen die `title` Variable in einem `h1` Tag an und `*ngFor` dann eine Liste, die jedes Element des `messages` Arrays mithilfe der Direktive `*ngFor` . Für jedes Element im Array erstellt `*ngFor` eine `message` , die wir im `li` Element verwenden. Das Ergebnis wird sein:

```

<h1>Angular 2 example</h1>
<ul>
  <li>Hello World!</li>
  <li>Another string</li>
  <li>Another one</li>
</ul>

```

Schritt 7

Jetzt erstellen wir eine `main.ts` Datei, die die erste Datei ist, die Angular betrachtet.

Erstellen Sie die Datei `app/main.ts`

```

import { bootstrap } from '@angular/platform-browser-dynamic';
import { AppComponent } from './app.component';

bootstrap(AppComponent);

```

Wir importieren die `bootstrap` Funktion und die `AppComponent` Klasse und verwenden dann `bootstrap`, um Angular mitzuteilen, welche Komponente als Root verwendet werden soll.

Schritt 8

Es ist Zeit, deine erste App zu starten. Art

```
npm start
```

in Ihrer Konsole / Terminal. Dies wird ein bereits erstelltes Skript ausführen `package.json` die `lite-Server` startet, öffnet die App in einem Browser - Fenster, und betreibt das Typoskript Transpiler im Watch - Modus (so `.ts` Dateien transpiled werden und der Browser aktualisiert werden, wenn Sie die Änderungen speichern) .

Was jetzt?

Lesen Sie [das offizielle Angular 2-Handbuch](#) und die anderen Themen in [der Dokumentation](#) zu [StackOverflow](#) .

Sie können `AppComponent` auch bearbeiten, um externe Vorlagen, Stile zu verwenden oder Komponentenvariablen hinzuzufügen / zu bearbeiten. Sie sollten Ihre Änderungen sofort nach dem Speichern der Dateien sehen.

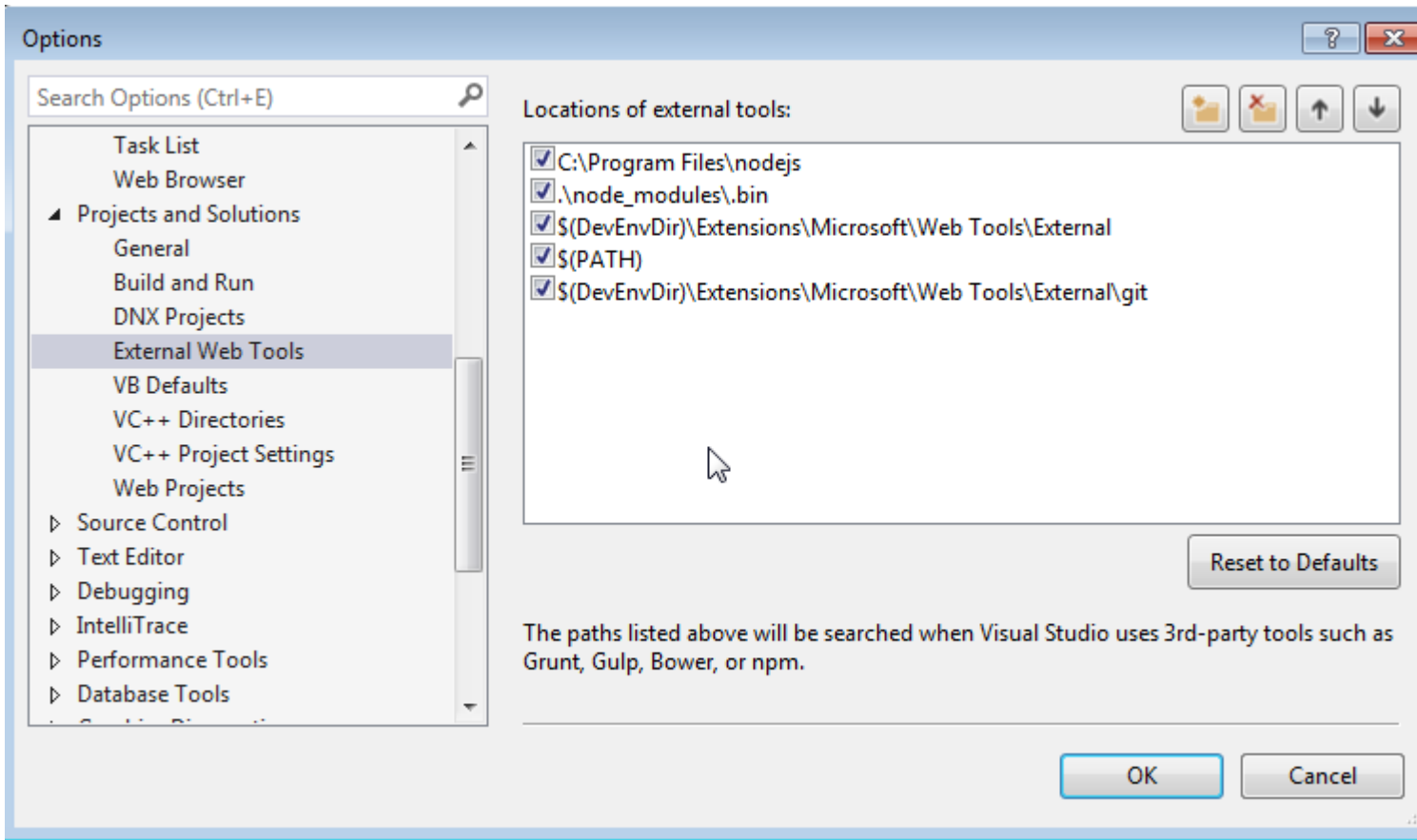
Synchronisierung von Visual Studios mit NPM- und NODE-Updates

Schritt 1: Suchen Sie nach dem Download von Node.js, der normalerweise unter `C: / program files / nodejs` installiert ist

Schritt 2: Öffnen Sie Visual Studios und navigieren Sie zu "Extras> Optionen".

Schritt 3: Navigieren Sie im Optionsfenster zu "Projekte und Lösungen> Externe Webtools".

Schritt 4: Fügen Sie einen neuen Eintrag für Ihre Node.js-Dateiposition hinzu (`C: / program files / nodejs`). WICHTIG Verwenden Sie die Pfeiltasten im Menü, um Ihre Referenz an den Anfang der Liste zu verschieben.



Schritt 5: Starten Sie Visual Studios neu und führen Sie eine npm-Installation für Ihr Projekt über das npm-Befehlsfenster aus

Durchkommen dieses lästigen Firmenvertreters

Wenn Sie versuchen, eine Angular2-Site auf Ihrem Windows-Arbeitscomputer bei XYZ MegaCorp zum Laufen zu bringen, besteht die Möglichkeit, dass Sie Probleme mit dem Proxy des Unternehmens haben.

Es gibt (mindestens) zwei Paketmanager, die den Proxy durchlaufen müssen:

1. NPM
2. Typisierungen

Für NPM müssen Sie der `.npmrc` Datei die folgenden Zeilen `.npmrc` :

```
proxy=http://[DOMAIN]%5C[USER]:[PASS]@[PROXY]:[PROXYPORT] /
https-proxy=http://[DOMAIN]%5C[USER]:[PASS]@[PROXY]:[PROXYPORT] /
```

Für Typings müssen Sie der `.typingsrc` Datei die folgenden Zeilen `.typingsrc` :

```
proxy=http://[DOMAIN]%5C[USER]:[PASS]@[PROXY]:[PROXYPORT] /
https-proxy=http://[DOMAIN]%5C[USER]:[PASS]@[PROXY]:[PROXYPORT] /
rejectUnauthorized=false
```

Diese Dateien sind wahrscheinlich noch nicht vorhanden, sodass Sie sie als leere Textdateien

erstellen können. Sie können der Projektwurzel hinzugefügt werden (dieselbe Stelle wie `package.json` oder Sie können sie in `%HOMEPATH%` und stehen für alle Ihre Projekte zur Verfügung.

Das etwas, das nicht offensichtlich ist und der Hauptgrund ist, warum die Proxy-Einstellungen nicht funktionieren, ist der `%5C` der die URL-Kodierung des `\`, um die Domänen- und Benutzernamen zu trennen. Vielen Dank an Steve Roberts für diesen Einsatz: [Verwenden von npm hinter dem Firmenproxy .pac](#)

Erste Schritte mit Angular 2 mit node.js / expressjs Backend (http-Beispiel enthalten)

Wir erstellen ein einfaches "Hallo Welt!" App mit Angular2 2.4.1 (`@NgModule` Änderung) mit einem node.js (expressjs) -Backend.

Voraussetzungen

- [Node.js](#) v4.xx oder höher
- [npm](#) v3.xx oder höher oder [Garn](#)

Führen Sie dann `npm install -g typescript` oder `yarn global add typescript`, um TypeScript global zu installieren

Roadmap

Schritt 1

Erstellen Sie einen neuen Ordner (und das Stammverzeichnis unseres Back-End) für unsere App. Nennen wir es `Angular2-express`.

Befehlszeile :

```
mkdir Angular2-express
cd Angular2-express
```

Schritt 2

Erstellen Sie die `package.json` (für Abhängigkeiten) und `app.js` (für das Bootstrapping) für unsere `node.js` App.

package.json:

```
{
  "name": "Angular2-express",
  "version": "1.0.0",
  "description": "",
```

```
"scripts": {
  "start": "node app.js"
},
"author": "",
"license": "ISC",
"dependencies": {
  "body-parser": "^1.13.3",
  "express": "^4.13.3"
}
}
```

app.js:

```
var express = require('express');
var app = express();
var server = require('http').Server(app);
var bodyParser = require('body-parser');

server.listen(process.env.PORT || 9999, function(){
  console.log("Server connected. Listening on port: " + (process.env.PORT || 9999));
});

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: true}));

app.use(express.static(__dirname + '/front'));

app.get('/test', function(req,res){ //example http request receiver
  return res.send(myTestVar);
});

//send the index.html on every page refresh and let angular handle the routing
app.get('/*', function(req, res, next) {
  console.log("Reloading");
  res.sendFile('index.html', { root: __dirname });
});
```

Führen Sie dann eine `npm install` oder ein `yarn`, um die Abhängigkeiten zu installieren.

Nun ist unsere Backend-Struktur abgeschlossen. Lasst uns zum Front-End gehen.

Schritt 3

Unser Frontend sollte sich in einem Ordner namens `front` innerhalb unseres `Angular2-express` Ordners befinden.

Befehlszeile:

```
mkdir front
cd front
```

Genau wie wir es mit unserem Backend getan haben, benötigt auch unser Frontend die Abhängigkeitsdateien. Lassen Sie uns die folgenden Dateien erstellen: `package.json`, `systemjs.config.js`, `tsconfig.json`

package.json :

```
{
  "name": "Angular2-express",
  "version": "1.0.0",
  "scripts": {
    "tsc": "tsc",
    "tsc:w": "tsc -w"
  },
  "licenses": [
    {
      "type": "MIT",
      "url": "https://github.com/angular/angular.io/blob/master/LICENSE"
    }
  ],
  "dependencies": {
    "@angular/common": "~2.4.1",
    "@angular/compiler": "~2.4.1",
    "@angular/compiler-cli": "^2.4.1",
    "@angular/core": "~2.4.1",
    "@angular/forms": "~2.4.1",
    "@angular/http": "~2.4.1",
    "@angular/platform-browser": "~2.4.1",
    "@angular/platform-browser-dynamic": "~2.4.1",
    "@angular/platform-server": "^2.4.1",
    "@angular/router": "~3.4.0",
    "core-js": "^2.4.1",
    "reflect-metadata": "^0.1.8",
    "rxjs": "^5.0.2",
    "systemjs": "0.19.40",
    "zone.js": "^0.7.4"
  },
  "devDependencies": {
    "@types/core-js": "^0.9.34",
    "@types/node": "^6.0.45",
    "typescript": "2.0.2"
  }
}
```

systemjs.config.js:

```
/**
 * System configuration for Angular samples
 * Adjust as necessary for your application needs.
 */
(function (global) {
  System.config({
    defaultJSExtensions:true,
    paths: {
      // paths serve as alias
      'npm:': 'node_modules/'
    },
    // map tells the System loader where to look for things
    map: {
      // our app is within the app folder
      app: 'app',
      // angular bundles
      '@angular/core': 'npm:@angular/core/bundles/core.umd.js',
      '@angular/common': 'npm:@angular/common/bundles/common.umd.js',

```



```

    '@angular/compiler': 'npm:@angular/compiler/bundles/compiler.umd.js',
    '@angular/platform-browser': 'npm:@angular/platform-browser/bundles/platform-
browser.umd.js',
    '@angular/platform-browser-dynamic': 'npm:@angular/platform-browser-
dynamic/bundles/platform-browser-dynamic.umd.js',
    '@angular/http': 'npm:@angular/http/bundles/http.umd.js',
    '@angular/router': 'npm:@angular/router/bundles/router.umd.js',
    '@angular/forms': 'npm:@angular/forms/bundles/forms.umd.js',
    // other libraries
    'rxjs': 'npm:rxjs',
    'angular-in-memory-web-api': 'npm:angular-in-memory-web-api',
  },
  // packages tells the System loader how to load when no filename and/or no extension
  packages: {
    app: {
      main: './main.js',
      defaultExtension: 'js'
    },
    rxjs: {
      defaultExtension: 'js'
    }
  }
});
})(this);

```

tsconfig.json:

```

{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "moduleResolution": "node",
    "sourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "removeComments": false,
    "noImplicitAny": false
  },
  "compileOnSave": true,
  "exclude": [
    "node_modules/*"
  ]
}

```

Führen Sie dann eine `npm install` oder ein `yarn`, um die Abhängigkeiten zu installieren.

Nun, da unsere Abhängigkeitsdateien vollständig sind. Gehen wir weiter zu unserem `index.html`:

index.html:

```

<html>
  <head>
    <base href="/">
    <title>Angular2-express</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- 1. Load libraries -->
    <!-- Polyfill(s) for older browsers -->

```

```

<script src="node_modules/core-js/client/shim.min.js"></script>
<script src="node_modules/zone.js/dist/zone.js"></script>
<script src="node_modules/reflect-metadata/Reflect.js"></script>
<script src="node_modules/systemjs/dist/system.src.js"></script>
<!-- 2. Configure SystemJS -->
<script src="systemjs.config.js"></script>
<script>
  System.import('app').catch(function(err){ console.error(err); });
</script>

</head>
<!-- 3. Display the application -->
<body>
  <my-app>Loading...</my-app>
</body>
</html>

```

Jetzt können wir unsere erste Komponente erstellen. Erstellen Sie einen Ordner namens `app` in unserem `front` Ordner.

Befehlszeile:

```

mkdir app
cd app

```

Lassen Sie uns die folgenden Dateien mit dem Namen `main.ts`, `app.module.ts`, `app.component.ts`

main.ts:

```

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app.module';

const platform = platformBrowserDynamic();
platform.bootstrapModule(AppModule);

```

app.module.ts:

```

import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from "@angular/http";

import { AppComponent }  from './app.component';

@NgModule({
  imports:      [
    BrowserModule,
    HttpClientModule
  ],
  declarations: [
    AppComponent
  ],
  providers: [ ],
  bootstrap:   [ AppComponent ]
})
export class AppModule { }

```

app.component.ts:

```
import { Component } from '@angular/core';
import { Http } from '@angular/http';

@Component({
  selector: 'my-app',
  template: 'Hello World!',
  providers: []
})
export class AppComponent {
  constructor(private http: Http){
    //http get example
    this.http.get('/test')
      .subscribe((res)=>{
        console.log(res);
      });
  }
}
```

Danach kompilieren Sie die Typoscript-Dateien in Javascript-Dateien. Gehen Sie vom aktuellen Verzeichnis (innerhalb des Ordners Angular2-Express) aus um 2 Stufen und führen Sie den folgenden Befehl aus.

Befehlszeile:

```
cd ..
cd ..
tsc -p front
```

Unsere Ordnerstruktur sollte folgendermaßen aussehen:

```
Angular2-express
├─ app.js
├─ node_modules
├─ package.json
├─ front
│   └─ package.json
│   └─ index.html
│   └─ node_modules
│   └─ systemjs.config.js
│   └─ tsconfig.json
│   └─ app
│       └─ app.component.ts
│       └─ app.component.js.map
│       └─ app.component.js
│       └─ app.module.ts
│       └─ app.module.js.map
│       └─ app.module.js
│       └─ main.ts
│       └─ main.js.map
│       └─ main.js
```

Führen Sie schließlich im Ordner Angular2-Express den Befehl `node app.js` in der Befehlszeile aus. Öffnen Sie Ihren bevorzugten Browser und überprüfen Sie `localhost:9999`, um Ihre App anzuzeigen.

Lassen Sie uns in Angular 4 eintauchen!

Angular 4 ist jetzt verfügbar! Tatsächlich verwendet Angular seit Angular 2 Semver, was erfordert, dass die Hauptzahl erhöht wird, wenn brechende Änderungen eingeführt wurden. Das Angular-Team hat Funktionen verschoben, die zu brechenden Änderungen führen. Diese werden mit Angular 4 veröffentlicht. Angular 3 wurde übersprungen, um die Versionsnummern der Kernmodule anzugleichen, da der Router bereits Version 3 hatte.

Laut Angular-Team werden Angular 4-Anwendungen weniger Platz benötigen und schneller als zuvor sein. Sie haben das Animationspaket vom @angular/core / Kernpaket getrennt. Wenn jemand kein Animationspaket verwendet, wird zusätzlicher Code nicht in der Produktion enden. Die Vorlagenbindungssyntax unterstützt jetzt die if / else-Stilsyntax. Angular 4 ist jetzt mit der neuesten Version von Typescript 2.1 und 2.2 kompatibel. Angular 4 wird also aufregender.

Jetzt zeige ich Ihnen, wie Sie Angular 4 in Ihrem Projekt einrichten.

Beginnen wir mit dem Angular-Setup auf drei verschiedene Arten:

Sie können Angular-CLI (Command Line Interface) verwenden. Es installiert alle Abhängigkeiten für Sie.

- Sie können von Angular 2 zu Angular 4 wechseln.
- Sie können github verwenden und die Angular4-Boilerplate klonen. (Es ist das einfachste.)
- Winkeleinrichtung mit Winkelschnittstelle (Command Line Interface).

Vergewissern Sie sich vor der Verwendung von Angular-CLI, dass auf Ihrem Computer ein Knoten installiert ist. Hier verwende ich Knoten v7.8.0. Öffnen Sie nun Ihr Terminal und geben Sie den folgenden Befehl für Angular-CLI ein.

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

oder

```
yarn global add @angular/cli
```

abhängig vom verwendeten Paketmanager.

Installieren wir Angular 4 mithilfe von Angular-CLI.

```
ng new Angular4-boilerplate
```

cd Angular4-boilerplate Wir sind alle auf Angular 4 eingestellt. Seine recht einfache und unkomplizierte Methode.

Winkeleinrichtung durch Migration von Angular 2 zu Angular 4

Nun sehen wir den zweiten Ansatz. Ich zeige Ihnen, wie Sie Angular 2 nach Angular 4 migrieren.

Dazu müssen Sie ein Angular 2-Projekt klonen und die Angular 2-Abhängigkeiten mit der Angular 4-Abhängigkeit in Ihrem package.json wie folgt aktualisieren:

```
"dependencies": {
  "@angular/animations": "^4.1.0",
  "@angular/common": "4.0.2",
  "@angular/compiler": "4.0.2",
  "@angular/core": "^4.0.1",
  "@angular/forms": "4.0.2",
  "@angular/http": "4.0.2",
  "@angular/material": "^2.0.0-beta.3",
  "@angular/platform-browser": "4.0.2",
  "@angular/platform-browser-dynamic": "4.0.2",
  "@angular/router": "4.0.2",
  "typescript": "2.2.2"
}
```

Dies sind die Hauptabhängigkeiten für Angular 4. Jetzt können Sie npm installieren und dann starten, um die Anwendung auszuführen. Zur Referenz meine package.json.

Winkeleinrichtung aus dem Github-Projekt

Bevor Sie mit diesem Schritt beginnen, vergewissern Sie sich, dass git in Ihrem Computer installiert ist. Öffnen Sie Ihr Terminal und klonen Sie die angle4-boilerplate mit dem folgenden Befehl:

```
git@github.com: CypherTree/angular4-boilerplate.git
```

Installieren Sie dann alle Abhängigkeiten und führen Sie sie aus.

```
npm install
```

```
npm start
```

Und Sie sind mit dem Setup von Angular 4 fertig. Alle Schritte sind sehr unkompliziert, so dass Sie sich für jeden entscheiden können.

Verzeichnisstruktur der angle4-boilerplate

```
Angular4-boilerplate
-karma
-node_modules
-src
  -mocks
  -models
    -loginform.ts
    -index.ts
  -modules
    -app
      -app.component.ts
      -app.component.html
      -login
    -login.component.ts
    -login.component.html
```

```
-login.component.css
  -widget
-widget.component.ts
-widget.component.html
-widget.component.css
.....
-services
  -login.service.ts
  -rest.service.ts
-app.routing.module.ts
-app.module.ts
-bootstrap.ts
-index.html
-vendor.ts
-typings
-webpack
-package.json
-tsconfig.json
-tslint.json
-typings.json
```

Grundlegendes Verständnis für die Verzeichnisstruktur:

Der gesamte Code befindet sich im src-Ordner.

Der Ordner "Mocks" ist für Modelldaten, die zu Testzwecken verwendet werden.

Der Modellordner enthält die Klasse und die Schnittstelle, die in der Komponente verwendet werden.

Der Ordner "modules" enthält eine Liste von Komponenten wie App, Login, Widget usw. Alle Komponenten enthalten Typescript-, HTML- und CSS-Dateien. index.ts dient zum Exportieren der gesamten Klasse.

Der Ordner Dienste enthält eine Liste der in der Anwendung verwendeten Dienste. Ich habe Rest-Service und andere Komponentenservice getrennt. Im Ruhezustand enthält der Dienst verschiedene http-Methoden. Der AnmeldeDienst dient als Vermittler zwischen der Anmeldekomponente und dem Ruhezustandsdienst.

Die Datei app.routing.ts beschreibt alle möglichen Routen für die Anwendung.

app.module.ts beschreibt das App-Modul als Root-Komponente.

bootstrap.ts führt die gesamte Anwendung aus.

Der Webpack-Ordner enthält die Webpack-Konfigurationsdatei.

package.json-Datei ist für alle Abhängigkeiten.

Karma enthält die Karma-Konfiguration für den Komponententest.

node_modules enthält eine Liste von Paketpaketen.

Beginnen wir mit der Login-Komponente. In login.component.html

```

<form>Dreamfactory - Addressbook 2.0
  <label>Email</label> <input id="email" form="" name="email" type="email" />
  <label>Password</label> <input id="password" form="" name="password"
type="password" />
  <button form="">Login</button>
</form>

```

In login.component.ts

```

import { Component } from '@angular/core';
import { Router } from '@angular/router';
import { Form, FormGroup } from '@angular/forms';
import { LoginForm } from '../models';
import { LoginService } from '../services/login.service';

@Component({
  selector: 'login',
  template: require('./login.component.html'),
  styles: [require('./login.component.css')]
})
export class LoginComponent {

  constructor(private loginService: LoginService, private router: Router, form: LoginForm) {
  }

  getLogin(form: LoginForm): void {
    let username = form.email;
    let password = form.password;
    this.loginService.getAuthenticate(form).subscribe(() => {
      this.router.navigate(['/calender']);
    });
  }
}

```

Wir müssen diese Komponente in index.ts exportieren.

```
export * from './login/login.component';
```

Wir müssen Routen für die Anmeldung in app.routes.ts festlegen

```

const appRoutes: Routes = [
  {
    path: 'login',
    component: LoginComponent
  },
  .....
  {
    path: '',
    pathMatch: 'full',
    redirectTo: '/login'
  }
];

```

In der Wurzelkomponente app.module.ts müssen Sie nur diese Komponente importieren.

```
.....
```

```
import { LoginComponent } from './modules';
.....
@NgModule({
  bootstrap: [AppComponent],
  declarations: [
    LoginComponent
    .....
    .....
  ]
  .....
})
export class AppModule { }
```

und danach npm installieren und npm starten. Bitte schön! Sie können den Anmeldebildschirm in Ihrem localhost überprüfen. Bei Schwierigkeiten können Sie sich auf die [angle4-Boilerplate](#) beziehen.

Grundsätzlich kann ich mit der Anwendung Angular 4 weniger Gebäudepakete und schnelleres Ansprechen spüren und obwohl ich in der Codierung genau Angular 2 gefunden habe.

Erste Schritte mit Angular 2 online lesen: <https://riptutorial.com/de/angular2/topic/789/erste-schritte-mit-angular-2>

Kapitel 2: Aktualisierungen

Examples

Aktualisierungen der Typisierung in folgenden Fällen: Typisierung WARN ist veraltet

Warnmeldung:

```
typings WARN deprecated 10/25/2016: "registry:dt/jasmine#2.5.0+20161003201800" is deprecated  
(updated, replaced or removed)
```

Aktualisieren Sie die Referenz mit:

```
npm run typings -- install dt~jasmine --save --global
```

Ersetzen Sie [jasmine] für jede Bibliothek, die eine Warnung auslöst

Aktualisierungen online lesen: <https://riptutorial.com/de/angular2/topic/7814/aktualisierungen>

Kapitel 3: Angular - ForLoop

Syntax

1. `<div *ngFor = "Artikel lassen; lassen Sie i = index"> {{i}} {{item}} </div>`

Bemerkungen

Die `*ngFor` Strukturanweisung wird als Schleife in einer Auflistung ausgeführt und wiederholt ein Stück HTML für jedes Element einer Auflistung.

`@View` Dekorator ist jetzt veraltet. Entwickler sollten `template` oder `'templateUrl'`-Eigenschaften für den `@Component` Dekorator verwenden.

Examples

Winkel für 2-Schleife

Für Live- [Programme klicken Sie auf ...](#)

```
<!doctype html>
<html>
<head>
  <title>ng for loop in angular 2 with ES5.</title>
  <script type="text/javascript" src="https://code.angularjs.org/2.0.0-alpha.28/angular2.sfx.dev.js"></script>
  <script>
    var ngForLoop = function () {
      this.msg = "ng for loop in angular 2 with ES5.";
      this.users = ["Anil Singh", "Sunil Singh", "Sushil Singh", "Aradhya", 'Reena'];
    };

    ngForLoop.annotations = [
      new angular.Component({
        selector: 'ngforloop'
      }),
      new angular.View({
        template: '<H1>{{msg}}</H1>' +
          '<p> User List : </p>' +
          '<ul>' +
          '<li *ng-for="let user of users">' +
          '{{user}}' +
          '</li>' +
          '</ul>',
        directives: [angular.NgFor]
      })
    ];

    document.addEventListener("DOMContentLoaded", function () {
      angular.bootstrap(ngForLoop);
    });
  </script>
```

```

</head>
<body>
  <ngforloop></ngforloop>
  <h2>
    <a href="http://www.code-sample.com/" target="_blank">For more detail...</a>
  </h2>
</body>
</html>

```

NgFor - Markup für Schleife

Die **NgFor**- Direktive instanziiert eine Vorlage einmal pro Element aus einer Iteration. Der Kontext für jede instanziierte Vorlage erbt vom äußeren Kontext, wobei die angegebene Schleifenvariable vom aktuellen Element auf das aktuelle Element gesetzt wird.

Um den Standard-Tracking-Algorithmus anzupassen, unterstützt **NgFor** die Option **trackBy** . **trackBy** nimmt eine Funktion mit zwei Argumenten: index und item. Wenn **trackBy** angegeben ist, ändert sich der Winkel um den Rückgabewert der Funktion.

```

<li *ngFor="let item of items; let i = index; trackBy: trackByFn">
  {{i}} - {{item.name}}
</li>

```

Zusätzliche Optionen : NgFor stellt mehrere exportierte Werte zur Verfügung, die für lokale Variablen als Alias verwendet werden können:

- **Der Index** wird für jeden Schablonenkontext auf die aktuelle Schleifeniteration gesetzt.
- **first** wird auf einen booleschen Wert gesetzt, der angibt, ob das Element der erste in der Iteration ist.
- **last** wird auf einen booleschen Wert gesetzt, der angibt, ob das Element das letzte Element in der Iteration ist.
- **even** wird auf einen booleschen Wert gesetzt, der angibt, ob dieses Element einen geraden Index hat.
- **odd** wird auf einen booleschen Wert gesetzt, der angibt, ob dieses Element einen ungeraden Index hat.

* ngFür in den Tabellenzeilen

```

<table>
  <thead>
    <th>Name</th>
    <th>Index</th>
  </thead>
  <tbody>
    <tr *ngFor="let hero of heroes">
      <td>{{hero.name}}</td>
    </tr>
  </tbody>
</table>

```

* ngFür mit Komponente

```

@Component({
  selector: 'main-component',
  template: '<example-component
            *ngFor="let hero of heroes"
            [hero]="hero"></example-component>'
})

@Component({
  selector: 'example-component',
  template: '<div>{{hero?.name}}</div>'
})

export class ExampleComponent {
  @Input() hero : Hero = null;
}

```

* ngFür X Anzahl der Elemente pro Zeile

Beispiel zeigt 5 Elemente pro Zeile:

```

<div *ngFor="let item of items; let i = index">
  <div *ngIf="i % 5 == 0" class="row">
    {{ item }}
    <div *ngIf="i + 1 < items.length">{{ items[i + 1] }}</div>
    <div *ngIf="i + 2 < items.length">{{ items[i + 2] }}</div>
    <div *ngIf="i + 3 < items.length">{{ items[i + 3] }}</div>
    <div *ngIf="i + 4 < items.length">{{ items[i + 4] }}</div>
  </div>
</div>

```

Angular - ForLoop online lesen: <https://riptutorial.com/de/angular2/topic/6543/angular---forloop>

Kapitel 4: Angular 2 Änderungserkennung und manuelle Auslösung

Examples

Grundlegendes Beispiel

Übergeordnete Komponente:

```
import {Component} from '@angular/core';

@Component({
  selector: 'parent-component',
  templateUrl: './parent-component.html'
})
export class ParentComponent {
  users : Array<User> = [];
  changeUsersActivation(user : User){
    user.changeButtonState();
  }
  constructor(){
    this.users.push(new User('Narco', false));
    this.users.push(new User('Bombasto', false));
    this.users.push(new User('Celeritas', false));
    this.users.push(new User('Magneta', false));
  }
}

export class User {
  firstName : string;
  active : boolean;

  changeButtonState(){
    this.active = !this.active;
  }
  constructor(_firstName :string, _active : boolean){
    this.firstName = _firstName;
    this.active = _active;
  }
}
```

Übergeordnetes HTML:

```
<div>
  <child-component [usersDetails]="users"
    (changeUsersActivation)="changeUsersActivation($event)" >
  </child-component>
</div>
```

untergeordnete Komponente:

```

import {Component, Input, EventEmitter, Output} from '@angular/core';
import {User} from "../parent.component";

@Component({
  selector: 'child-component',
  templateUrl: './child-component.html',
  styles: [`
    .btn {
      height: 30px;
      width: 100px;
      border: 1px solid rgba(0, 0, 0, 0.33);
      border-radius: 3px;
      margin-bottom: 5px;
    }
  `]
})
export class ChildComponent {
  @Input() usersDetails : Array<User> = null;
  @Output() changeUsersActivation = new EventEmitter();

  triggerEvent(user : User) {
    this.changeUsersActivation.emit(user);
  }
}

```

untergeordnetes HTML:

```

<div>
  <div>
    <table>
      <thead>
        <tr>
          <th>Name</th>
          <th></th>
        </tr>
      </thead>
      <tbody *ngIf="user !== null">
        <tr *ngFor="let user of usersDetails">
          <td>{{user.firstName}}</td>
          <td><button class="btn" (click)="triggerEvent(user)">{{user.active}}</button></td>
        </tr>
      </tbody>
    </table>
  </div>
</div>

```

Angular 2 Änderungserkennung und manuelle Auslösung online lesen:

<https://riptutorial.com/de/angular2/topic/8971/angular-2-anderungserkennung-und-manuelle-auslosung>

Kapitel 5: Angular 2 datengesteuerte Formulare

Bemerkungen

```
this.myForm = this.formBuilder.group
```

erstellt ein Formularobjekt mit der Benutzerkonfiguration und ordnet es der `this.myForm`-Variablen zu.

```
'loginCredentials': this.formBuilder.group
```

Die Methode erstellt eine Gruppe von Steuerelementen, die aus einem **formControlName** bestehen, z. `login` und `value ['', Validators.required]`, wobei der erste Parameter der Anfangswert der Formulareingabe ist und die Sekunden ein Validator oder ein Array von Validatoren wie in `'email': ['', [Validators.required, customValidator]]`,

```
'hobbies': this.formBuilder.array
```

Erstellt ein Array von Gruppen, bei dem der Index der Gruppe **formGroupName** im Array lautet und auf die wie folgt zugegriffen wird:

```
<div *ngFor="let hobby of myForm.find('hobbies').controls; let i = index">
  <div formGroupName="{{i}}">...</div>
</div>
```

```
onAddHobby() {
  (<FormArray>this.myForm.find('hobbies')).push(new FormGroup({
    'hobby': new FormControl('', Validators.required)
  }))
}
```

Diese Beispielmethode fügt dem Array eine neue `formGroup` hinzu. Für den Zugriff muss derzeit der Typ des Steuerelements angegeben werden, auf den wir zugreifen möchten. In diesem Beispiel lautet dieser Typ: `<FormArray>`

```
removeHobby(index: number) {
  (<FormArray>this.myForm.find('hobbies')).removeAt(index);
}
```

Für das Entfernen eines bestimmten Formularsteuerelements aus dem Array gelten dieselben Regeln wie oben

Examples

Datengesteuertes Formular

Komponente

```
import {Component, OnInit} from '@angular/core';
import {
  FormGroup,
  FormControl,
  FORM_DIRECTIVES,
  REACTIVE_FORM_DIRECTIVES,
  Validators,
  FormBuilder,
  FormArray
} from "@angular/forms";
import {Control} from "@angular/common";

@Component({
  moduleId: module.id,
  selector: 'app-data-driven-form',
  templateUrl: 'data-driven-form.component.html',
  styleUrls: ['data-driven-form.component.css'],
  directives: [FORM_DIRECTIVES, REACTIVE_FORM_DIRECTIVES]
})
export class DataDrivenFormComponent implements OnInit {
  myForm: FormGroup;

  constructor(private formBuilder: FormBuilder) {}

  ngOnInit() {
    this.myForm = this.formBuilder.group({
      'loginCredentials': this.formBuilder.group({
        'login': ['', Validators.required],
        'email': ['', [Validators.required, customValidator]],
        'password': ['', Validators.required]
      }),
      'hobbies': this.formBuilder.array([
        this.formBuilder.group({
          'hobby': ['', Validators.required]
        })
      ])
    });
  }

  removeHobby(index: number){
    (<FormArray>this.myForm.find('hobbies')).removeAt(index);
  }

  onAddHobby() {
    (<FormArray>this.myForm.find('hobbies')).push(new FormGroup({
      'hobby': new FormControl('', Validators.required)
    }));
  }

  onSubmit() {
    console.log(this.myForm.value);
  }
}
```



```

}

function customValidator(control: Control): {[s: string]: boolean} {
  if(!control.value.match("[a-z0-9!#$%&'*/=?^_`{|}~-]+(?:\\.[a-z0-9!#$%&'*/=?^_`{|}~-]+)*@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?")) {
    return {error: true}
  }
}

```

HTML-Markup

```

<h3>Register page</h3>
<form [formGroup]="myForm" (ngSubmit)="onSubmit()">
  <div formGroupName="loginCredentials">
    <div class="form-group">
      <div>
        <label for="login">Login</label>
        <input id="login" type="text" class="form-control" formControlName="login">
      </div>
      <div>
        <label for="email">Email</label>
        <input id="email" type="text" class="form-control" formControlName="email">
      </div>
      <div>
        <label for="password">Password</label>
        <input id="password" type="text" class="form-control" formControlName="password">
      </div>
    </div>
  </div>
  <div class="row" >
    <div formGroupName="hobbies">
      <div class="form-group">
        <label>Hobbies array:</label>
        <div *ngFor="let hobby of myForm.find('hobbies').controls; let i = index">
          <div formGroupName="{{i}}">
            <input id="hobby_{{i}}" type="text" class="form-control" formControlName="hobby">
            <button *ngIf="myForm.find('hobbies').length > 1"
(click)="removeHobby(i)">x</button>
          </div>
        </div>
        <button (click)="onAddHobby()">Add hobby</button>
      </div>
    </div>
    <button type="submit" [disabled]="!myForm.valid">Submit</button>
  </form>

```

Angular 2 datengesteuerte Formulare online lesen:

<https://riptutorial.com/de/angular2/topic/6463/angular-2-datengesteuerte-formulare>

Kapitel 6: Angular 2 Forms Update

Bemerkungen

Angular 2 ermöglicht den Zugriff auf die ngForm-Instanz, indem eine lokale Vorlagenvariable erstellt wird. In Angular 2 werden Direktive wie ngForm durch Angabe der Eigenschaft exportAs der Metadaten der Direktive verfügbar gemacht. Der Vorteil hierbei ist, dass Sie ohne viel Code auf die ngForm-Instanz zugreifen und diese verwenden können, um auf übergebene Werte zuzugreifen oder anhand der Eigenschaften (gültig, übergeben, Wert usw.) zu überprüfen, ob alle Felder gültig sind.

```
#f = ngForm (creates local template instance "f")
```

ngForm gibt das Ereignis "ngSubmit" aus, wenn es übermittelt wird (Weitere Informationen zum Ereignisemitter finden Sie in der @Output-Dokumentation).

```
(ngSubmit)= "login(f.value, f.submitted) "
```

"ngModel" erstellt ein Formularsteuerelement in Kombination mit dem Eingabe-Namensattribut.

```
<input type="text" [(ngModel)]="username" placeholder="enter username" required>
```

Wenn das Formular gesendet wird, enthält f.value das JSON-Objekt, das die übermittelten Werte darstellt.

```
{Benutzername: 'Sachin', Passwort: 'Welcome1'}
```

Examples

Einfaches Passwortänderungsformular mit Multi Control Validation

Die folgenden Beispiele verwenden die in RC3 eingeführte neue Formular-API.

pw-change.template.html

```
<form class="container" [formGroup]="pwChangeForm">
  <label for="current">Current Password</label>
  <input id="current" formControlName="current" type="password" required><br />

  <label for="newPW">New Password</label>
  <input id="newPW" formControlName="newPW" type="password" required><br />
  <div *ngIf="newPW.touched && newPW.newIsNotOld">
    New password can't be the same as current password.
  </div>

  <label for="confirm">Confirm new password</label>
```

```

<input id="confirm" formControlName="confirm" type="password" required><br />
<div *ngIf="confirm.touched && confirm.errors.newMatchesConfirm">
  The confirmation does not match.
</div>

<button type="submit">Submit</button>
</form>

```

pw-change.component.ts

```

import {Component} from '@angular/core'
import {REACTIVE_FORM_DIRECTIVES, FormBuilder, AbstractControl, FormGroup,
  Validators} from '@angular/forms'
import {PWChangeValidators} from './pw-validators'

@Component({
  moduleId: module.id
  selector: 'pw-change-form',
  templateUrl: './pw-change.template.html`,
  directives: [REACTIVE_FORM_DIRECTIVES]
})

export class PWChangeFormComponent {
  pwChangeForm: FormGroup;

  // Properties that store paths to FormControls makes our template less verbose
  current: AbstractControl;
  newPW: AbstractControl;
  confirm: AbstractControl;

  constructor(private fb: FormBuilder) { }
  ngOnInit() {
    this.pwChangeForm = this.fb.group({
      current: ['', Validators.required],
      newPW: ['', Validators.required],
      confirm: ['', Validators.required]
    }, {
      // Here we create validators to be used for the group as a whole
      validator: Validators.compose([
        PWChangeValidators.newIsNotOld,
        PWChangeValidators.newMatchesConfirm
      ])
    });
    this.current = this.pwChangeForm.controls['current'];
    this.newPW = this.pwChangeForm.controls['newPW'];
    this.confirm = this.pwChangeForm.controls['confirm'];
  }
}

```

pw-validators.ts

```

import {FormControl, FormGroup} from '@angular/forms'
export class PWChangeValidators {

  static OldPasswordMustBeCorrect(control: FormControl) {
    var invalid = false;

```

```

    if (control.value != PWChangeValidators.oldPW)
        return { oldPasswordMustBeCorrect: true }
    return null;
}

// Our cross control validators are below
// NOTE: They take in type FormGroup rather than FormControl
static newIsNotOld(group: FormGroup){
    var newPW = group.controls['newPW'];
    if(group.controls['current'].value == newPW.value)
        newPW.setErrors({ newIsNotOld: true });
    return null;
}

static newMatchesConfirm(group: FormGroup){
    var confirm = group.controls['confirm'];
    if(group.controls['newPW'].value != confirm.value)
        confirm.setErrors({ newMatchesConfirm: true });
    return null;
}
}
}

```

Eine Übersicht mit einigen Bootstrap-Klassen finden Sie [hier](#) .

Winkel 2: Vorlagengesteuerte Formulare

```

import { Component } from '@angular/core';
import { Router , ROUTER_DIRECTIVES} from '@angular/router';
import { NgForm } from '@angular/forms';

@Component({
    selector: 'login',
    template: `
<h2>Login</h2>
<form #f="ngForm" (ngSubmit)="login(f.value,f.valid)" novalidate>
  <div>
    <label>Username</label>
    <input type="text" [(ngModel)]="username" placeholder="enter username" required>
  </div>
  <div>
    <label>Password</label>
    <input type="password" name="password" [(ngModel)]="password" placeholder="enter password" required>
  </div>
  <input class="btn-primary" type="submit" value="Login">
</form>`
    //For long form we can use **templateUrl** instead of template
})

export class LoginComponent{

    constructor(private router : Router){ }

    login (formValue: any, valid: boolean){
        console.log(formValue);

        if(valid){
            console.log(valid);
        }
    }
}

```

```
}  
}
```

Winkel 2-Formular - Benutzerdefinierte E-Mail- / Passwort-Überprüfung

Für Live-Demo [klicken Sie bitte ..](#)

App-Index ts

```
import {bootstrap} from '@angular/platform-browser-dynamic';  
import {MyForm} from './my-form.component.ts';  
  
bootstrap(MyForm);
```

Benutzerdefinierter Validator

```
import {Control} from '@angular/common';  
  
export class CustomValidators {  
  static emailFormat(control: Control): [[key: string]: boolean] {  
    let pattern:RegExp = /\S+@\S+\.\S+;/;  
    return pattern.test(control.value) ? null : {"emailFormat": true};  
  }  
}
```

Formularkomponenten ts

```
import {Component} from '@angular/core';  
import {FORM_DIRECTIVES, NgForm, FormBuilder, Control, ControlGroup, Validators} from  
'@angular/common';  
import {CustomValidators} from './custom-validators';  
  
@Component({  
  selector: 'my-form',  
  templateUrl: 'app/my-form.component.html',  
  directives: [FORM_DIRECTIVES],  
  styleUrls: ['styles.css']  
})  
export class MyForm {  
  email: Control;  
  password: Control;  
  group: ControlGroup;  
  
  constructor(builder: FormBuilder) {  
    this.email = new Control('',  
      Validators.compose([Validators.required, CustomValidators.emailFormat])  
    );  
  
    this.password = new Control('',  
      Validators.compose([Validators.required, Validators.minLength(4)])  
    );  
  
    this.group = builder.group({  
      email: this.email,  
      password: this.password  
    });  
}
```

```

    }

    onSubmit() {
      console.log(this.group.value);
    }
  }
}

```

Formularkomponenten HTML

```

<form [ngFormModel]="group" (ngSubmit)="onSubmit()" novalidate>

  <div>
    <label for="email">Email:</label>
    <input type="email" id="email" [ngFormControl]="email">

    <ul *ngIf="email.dirty && !email.valid">
      <li *ngIf="email.hasError('required')">An email is required</li>
    </ul>
  </div>

  <div>
    <label for="password">Password:</label>
    <input type="password" id="password" [ngFormControl]="password">

    <ul *ngIf="password.dirty && !password.valid">
      <li *ngIf="password.hasError('required')">A password is required</li>
      <li *ngIf="password.hasError('minlength')">A password needs to have at least 4
characters</li>
    </ul>
  </div>

  <button type="submit">Register</button>

</form>

```

Winkel 2: Reaktive Formen (auch als Model-Driven Forms bezeichnet)

In diesem Beispiel wird Angular 2.0.0 Final Release verwendet

Registrierungsformular.Komponente.ts

```

import { FormGroup,
  FormControl,
  FormBuilder,
  Validators } from '@angular/forms';

@Component({
  templateUrl: "./registration-form.html"
})
export class ExampleComponent {
  constructor(private _fb: FormBuilder) { }

  exampleForm = this._fb.group({
    name: ['DefaultValue', [<any>Validators.required, <any>Validators.minLength(2)]],
    email: ['default@defa.ult', [<any>Validators.required, <any>Validators.minLength(2)]]
  })
}

```

Anmeldeformular.html

```
<form [formGroup]="exampleForm" novalidate (ngSubmit)="submit(exampleForm)">
  <label>Name: </label>
  <input type="text" formControlName="name"/>
  <label>Email: </label>
  <input type="email" formControlName="email"/>
  <button type="submit">Submit</button>
</form>
```

Angular 2 Forms (Reactive Forms) mit Registrierungsformular und Bestätigung der Passwortbestätigung

app.module.ts

Fügen Sie diese in Ihre app.module.ts-Datei ein, um reaktive Formulare zu verwenden

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
@NgModule({
  imports: [
    BrowserModule,
    FormsModule,
    ReactiveFormsModule,
  ],
  declarations: [
    AppComponent
  ]
  providers: [],
  bootstrap: [
    AppComponent
  ]
})
export class AppModule {}
```

app.component.ts

```
import { Component, OnInit } from '@angular/core';
import template from './add.component.html';
import { FormGroup, FormBuilder, Validators } from '@angular/forms';
import { matchingPasswords } from './validators';
@Component({
  selector: 'app',
  template
})
export class AppComponent implements OnInit {
  addForm: FormGroup;
  constructor(private formBuilder: FormBuilder) {
  }
  ngOnInit() {
```

```

this.addForm = this.formBuilder.group({
  username: ['', Validators.required],
  email: ['', Validators.required],
  role: ['', Validators.required],
  password: ['', Validators.required],
  password2: ['', Validators.required] },
  { validator: matchingPasswords('password', 'password2')
  })
});

addUser() {
  if (this.addForm.valid) {
    var adduser = {
      username: this.addForm.controls['username'].value,
      email: this.addForm.controls['email'].value,
      password: this.addForm.controls['password'].value,
      profile: {
        role: this.addForm.controls['role'].value,
        name: this.addForm.controls['username'].value,
        email: this.addForm.controls['email'].value
      }
    };
    console.log(adduser); // adduser var contains all our form values. store it where you
    want
    this.addForm.reset(); // this will reset our form values to null
  }
}
}

```

app.component.html

```

<div>
  <form [formGroup]="addForm">
    <input type="text" placeholder="Enter username" formControlName="username" />
    <input type="text" placeholder="Enter Email Address" formControlName="email"/>
    <input type="password" placeholder="Enter Password" formControlName="password" />
    <input type="password" placeholder="Confirm Password" name="password2"
    formControlName="password2"/>
    <div class='error' *ngIf="addForm.controls.password2.touched">
      <div class="alert-danger errorMessageadduser"
      *ngIf="addForm.hasError('mismatchedPasswords') "> Passwords do
      not match
    </div>
  </div>
  <select name="Role" formControlName="role">
    <option value="admin" >Admin</option>
    <option value="Accounts">Accounts</option>
    <option value="guest">Guest</option>
  </select>
  <br/>
  <br/>
  <button type="submit" (click)="addUser()"><span><i class="fa fa-user-plus" aria-
  hidden="true"></i></span> Add User </button>
</form>
</div>

```

validators.ts


```

export function matchingPasswords(passwordKey: string, confirmPasswordKey: string) {
  return (group: ControlGroup): {
    [key: string]: any
  } => {
    let password = group.controls[passwordKey];
    let confirmPassword = group.controls[confirmPasswordKey];

    if (password.value !== confirmPassword.value) {
      return {
        mismatchedPasswords: true
      };
    }
  }
}

```

Angular2 - Formularersteller

FormComponent.ts

```

import {Component} from "@angular/core";
import {FormBuilder} from "@angular/forms";

@Component({
  selector: 'app-form',
  templateUrl: './form.component.html',
  styleUrls: ['./form.component.scss'],
  providers : [FormBuilder]
})

export class FormComponent{
  form : FormGroup;
  emailRegex = /^@w+([\.-]?\w+)*@w+([\.-]?\w+)*(\.\w{2,3})+$/;

  constructor(fb: FormBuilder) {

    this.form = fb.group({
      FirstName : new FormControl({value: null}, Validators.compose([Validators.required,
Validators.maxLength(15)])),
      LastName : new FormControl({value: null}, Validators.compose([Validators.required,
Validators.maxLength(15)])),
      Email : new FormControl({value: null}, Validators.compose([
Validators.required,
Validators.maxLength(15),
Validators.pattern(this.emailRegex)]))
    });
  }
}

```

form.component.html

```

<form class="form-details" role="form" [formGroup]="form">
  <div class="row input-label">
    <label class="form-label" for="FirstName">First name</label>
    <input
      [formControl]="form.controls['FirstName']"
      type="text"
      class="form-control"
      id="FirstName"
    >
  </div>

```

```

        name="FirstName">
</div>
<div class="row input-label">
  <label class="form-label" for="LastName">Last name</label>
  <input
    [formControl]="form.controls['LastName']"
    type="text"
    class="form-control"
    id="LastName"
    name="LastName">
</div>
<div class="row">
  <label class="form-label" for="Email">Email</label>
  <input
    [formControl]="form.controls['Email']"
    type="email"
    class="form-control"
    id="Email"
    name="Email">
</div>
<div class="row">
  <button
    (click)="submit()"
    role="button"
    class="btn btn-primary submit-btn"
    type="button"
    [disabled]="!form.valid">Submit</button>
</div>
</div>
</form>

```

Angular 2 Forms Update online lesen: <https://riptutorial.com/de/angular2/topic/4607/angular-2-forms-update>

Kapitel 7: Angular RXJS Subjects und Observables mit API-Anforderungen

Bemerkungen

API-Anforderungen mit dem Angular 2 HTTP-Dienst und RxJS zu machen, ist der Arbeit mit den Versprechen in Angular 1.x sehr ähnlich.

Verwenden Sie die [Http](#)- Klasse, um Anforderungen zu stellen. Die `Http` - Klasse stellt die Methoden zur Ausgabe von HTTP - Anforderungen `GET` , `POST` , `PUT` , `DELETE` , `PATCH` , `HEAD` - Anfragen über entsprechende Verfahren. Außerdem wird eine generische `request` für die Ausgabe von HTTP-Anforderungen jeglicher Art bereitgestellt.

Alle Methoden der `Http` Klasse geben eine `Observable<Response>` , auf die Sie [RxJS-Operationen](#) anwenden können. Sie rufen die `.subscribe()` -Methode auf und übergeben eine Funktion, die aufgerufen wird, wenn Daten im Observable-Stream zurückgegeben werden.

Der beobachtbare Datenstrom für eine Anforderung enthält nur einen Wert - die `Response` und wird abgeschlossen, wenn die HTTP-Anforderung erfolgreich abgeschlossen wurde, oder Fehler / Fehler, wenn ein Fehler ausgegeben wird.

Beachten Sie, dass die vom `Http` Modul zurückgegebenen Observables *kalt* sind. Wenn Sie also das Observable mehrmals abonnieren, wird die Ursprungsanforderung für jedes Abonnement einmal *ausgeführt* . Dies kann passieren, wenn Sie das Ergebnis in mehreren Komponenten Ihrer Anwendung verwenden möchten. Für GET-Anforderungen kann dies nur zu einigen zusätzlichen Anforderungen führen. Dies kann jedoch zu unerwarteten Ergebnissen führen, wenn PUT- oder POST-Anforderungen mehr als einmal abonniert werden.

Examples

Grundanforderung

Das folgende Beispiel veranschaulicht eine einfache HTTP-GET-Anforderung. `http.get()` gibt ein `Observable` das die Methode `subscribe` . Dieses fügt die zurückgegebenen Daten an das `posts` Array an.

```
var posts = []

getPosts(http: Http):void {
  this.http.get(`https://jsonplaceholder.typicode.com/posts`)
    .map(response => response.json())
    .subscribe(post => posts.push(post));
}
```

API-Anforderungen kapseln

Es kann sinnvoll sein, die HTTP-Verarbeitungslogik in einer eigenen Klasse zu kapseln. Die folgende Klasse macht eine Methode zum Abrufen von Posts verfügbar. Es ruft die `http.get()` - Methode auf und ruft `.map` für das zurückgegebene `Observable` auf, um das `Response` Objekt in ein `Post` Objekt zu konvertieren.

```
import {Injectable} from "@angular/core";
import {Http, Response} from "@angular/http";

@Injectable()
export class BlogApi {

  constructor(private http: Http) {
  }

  getPost(id: number): Observable<Post> {
    return this.http.get(`https://jsonplaceholder.typicode.com/posts/${id}`)
      .map((response: Response) => {
        const srcData = response.json();
        return new Post(srcData)
      });
  }
}
```

Im vorherigen Beispiel wird eine `Post` Klasse verwendet, um die zurückgegebenen Daten aufzunehmen. Dies könnte folgendermaßen aussehen:

```
export class Post {
  userId: number;
  id: number;
  title: string;
  body: string;

  constructor(src: any) {
    this.userId = src && src.userId;
    this.id = src && src.id;
    this.title = src && src.title;
    this.body = src && src.body;
  }
}
```

Eine Komponente kann nun die Verwendung `BlogApi` Klasse einfach abrufen `Post` Daten , ohne sich selbst in Bezug auf die Funktionsweise der `Http` - Klasse.

Warten Sie auf mehrere Anfragen

Ein häufiges Szenario besteht darin, auf eine Reihe von Anforderungen zu warten, bevor Sie fortfahren. Dies kann mit der [forkJoin Methode erreicht werden](#) .

Im folgenden Beispiel werden mit `forkJoin` zwei Methoden aufgerufen, die `Observables` . Der in der `.subscribe` Methode angegebene Rückruf wird aufgerufen, wenn beide `Observables` abgeschlossen sind. Die von `.subscribe` angegebenen Parameter `.subscribe` mit der Reihenfolge überein, die im Aufruf von `.forkJoin` . In diesem Fall erste `posts` dann `tags` .

```
loadData() : void {
  Observable.forkJoin(
    this.blogApi.getPosts(),
    this.blogApi.getTags()
  ).subscribe((([posts, tags]: [Post[], Tag[]]) => {
    this.posts = posts;
    this.tags = tags;
  }));
}
```

Angular RXJS Subjects und Observables mit API-Anforderungen online lesen:

<https://riptutorial.com/de/angular2/topic/3577/angular-rxjs-subjects-und-observables-mit-api-anforderungen>

Kapitel 8: Angular2 Animationen

Einführung

Mit dem Animationssystem von Angular können Sie Animationen erstellen, die die gleiche native Leistung wie reine CSS-Animationen aufweisen. Sie können Ihre Animationslogik auch eng in den restlichen Anwendungscode integrieren, um die Steuerung zu erleichtern.

Examples

Grundanimation - Übergibt ein Element zwischen zwei Zuständen, die von einem Modellattribut gesteuert werden.

app.component.html

```
<div>
  <div>
    <div *ngFor="let user of users">
      <button
        class="btn"
        [@buttonState]="user.active"
        (click)="user.changeButtonState()">{{user.firstName}}</button>
    </div>
  </div>
</div>
```

app.component.ts

```
import {Component, trigger, state, transition, animate, style} from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styles: [`
    .btn {
      height: 30px;
      width: 100px;
      border: 1px solid rgba(0, 0, 0, 0.33);
      border-radius: 3px;
      margin-bottom: 5px;
    }
  `],
  animations: [
    trigger('buttonState', [
      state('true', style({
        background: '#04b104',
        transform: 'scale(1)'
      })),
      state('false', style({
```

```

        background: '#e40202',
        transform: 'scale(1.1)'
    )),
    transition('true => false', animate('100ms ease-in')),
    transition('false => true', animate('100ms ease-out'))
  ])
]
})
export class AppComponent {
  users : Array<User> = [];
  constructor(){
    this.users.push(new User('Narco', false));
    this.users.push(new User('Bombasto', false));
    this.users.push(new User('Celeritas', false));
    this.users.push(new User('Magneta', false));
  }
}

export class User {
  firstName : string;
  active : boolean;

  changeButtonState(){
    this.active = !this.active;
  }
  constructor(_firstName :string, _active : boolean){
    this.firstName = _firstName;
    this.active = _active;
  }
}

```

Angular2 Animationen online lesen: <https://riptutorial.com/de/angular2/topic/8970/angular2-animationen>

Kapitel 9: Angular2 Benutzerdefinierte Validierungen

Parameter

Parameter	Beschreibung
Steuerung	Dies ist das Steuerelement, das geprüft wird. Normalerweise möchten Sie sehen, ob control.value einige Kriterien erfüllt.

Examples

Beispiele für benutzerdefinierte Validatoren:

In Angular 2 gibt es zwei Arten von benutzerdefinierten Validatoren. Synchroner Validatoren wie im ersten Beispiel, die direkt auf dem Client ausgeführt werden, und asynchrone Validatoren (zweites Beispiel), mit denen Sie einen Remote-Service aufrufen können, um die Validierung für Sie durchzuführen. In diesem Beispiel sollte der Prüfer den Server anrufen, um festzustellen, ob ein Wert eindeutig ist.

```
export class CustomValidators {  
  
  static cannotContainSpace(control: Control) {  
    if (control.value.indexOf(' ') >= 0)  
      return { cannotContainSpace: true };  
  
    return null;  
  }  
  
  static shouldBeUnique(control: Control) {  
    return new Promise((resolve, reject) => {  
      // Fake a remote validator.  
      setTimeout(function () {  
        if (control.value == "exisitingUser")  
          resolve({ shouldBeUnique: true });  
        else  
          resolve(null);  
      }, 1000);  
    });  
  }  
}
```

Wenn Ihr Steuerwert gültig ist, geben Sie einfach null an den Aufrufer zurück. Andernfalls können Sie ein Objekt zurückgeben, das den Fehler beschreibt.

Validatoren im FormBuilder verwenden

```
constructor(fb: FormBuilder) {
```



```
this.form = fb.group({
  firstInput: ['', Validators.compose([Validators.required,
  CustomValidators.cannotContainSpace]), CustomValidators.shouldBeUnique],
  secondInput: ['', Validators.required]
});
}
```

Hier verwenden wir den FormBuilder, um ein sehr einfaches Formular mit zwei Eingabefeldern zu erstellen. Der FormBuilder nimmt für jedes Eingabesteuerelement ein Array mit drei Argumenten an.

1. Der Standardwert des Steuerelements.
2. Die Validatoren, die auf dem Client ausgeführt werden. Sie können `Validators.compose ([arrayOfValidators])` verwenden, um mehrere Validatoren auf Ihr Steuerelement anzuwenden.
3. Ein oder mehrere asynchrone Validatoren auf ähnliche Weise wie das zweite Argument.

get / set FormBuilder steuert die Parameter

Es gibt zwei Möglichkeiten, FormBuilder-Steuerparameter einzustellen.

1. Beim initialisieren:

```
exampleForm : FormGroup;
constructor(fb: FormBuilder){
  this.exampleForm = fb.group({
    name : new FormControl({value: 'default name'}, Validators.compose([Validators.required,
  Validators.maxLength(15)]))
  });
}
```

2. Nach dem Initialisieren:

```
this.exampleForm.controls['name'].setValue('default name');
```

Rufen Sie den Wert des Steuerelements FormBuilder ab:

```
let name = this.exampleForm.controls['name'].value();
```

Angular2 Benutzerdefinierte Validierungen online lesen:

<https://riptutorial.com/de/angular2/topic/6284/angular2-benutzerdefinierte-validierungen>

Kapitel 10: Angular2 CanActivate

Examples

Angular2 CanActivate

In einem Router implementiert:

```
export const MainRoutes: Route[] = [{
  path: '',
  children: [ {
    path: 'main',
    component: MainComponent ,
    canActivate : [CanActivateRoute]
  } ]
}];
```

Die `canActivateRoute` Datei:

```
@Injectable()
export class CanActivateRoute implements CanActivate{
  constructor() {}
  canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
    return true;
  }
}
```

Angular2 CanActivate online lesen: <https://riptutorial.com/de/angular2/topic/8899/angular2-canactivate>

Kapitel 11: Angular2 Datenbindung

Examples

@Eingang()

Übergeordnete Komponente: Benutzerlisten initialisieren.

```
@Component({
  selector: 'parent-component',
  template: '<div>
    <child-component [users]="users"></child-component>
  </div>'
})
export class ParentComponent implements OnInit{
  let users : List<User> = null;

  ngOnInit() {
    users.push(new User('A', 'A', 'A@gmail.com');
    users.push(new User('B', 'B', 'B@gmail.com');
    users.push(new User('C', 'C', 'C@gmail.com');
  }
}
```

Untergeordnete Komponente mit Eingabe () Benutzer von der übergeordneten Komponente abrufen

```
@Component({
  selector: 'child-component',
  template: '<div>
    <table *ngIf="users !== null">
      <thead>
        <th>Name</th>
        <th>FName</th>
        <th>Email</th>
      </thead>
      <tbody>
        <tr *ngFor="let user of users">
          <td>{{user.name}}</td>
          <td>{{user.fname}}</td>
          <td>{{user.email}}</td>
        </tr>
      </tbody>
    </table>

  </div>',
})
export class ChildComponent {
  @Input() users : List<User> = null;
}

export class User {
```

```
name : string;
fname : string;
email : string;

constructor(_name : string, _fname : string, _email : string){
  this.name = _name;
  this.fname = _fname;
  this.email = _email;
}
}
```

Angular2 Datenbindung online lesen: <https://riptutorial.com/de/angular2/topic/9036/angular2-datenbindung>

Kapitel 12: Angular2 Eingang () Ausgang ()

Examples

Eingang()

Übergeordnete Komponente: Benutzerlisten initialisieren.

```
@Component({
  selector: 'parent-component',
  template: '<div>
    <child-component [users]="users"></child-component>
  </div>'
})
export class ParentComponent implements OnInit{
  let users : List<User> = null;

  ngOnInit() {
    users.push(new User('A', 'A', 'A@gmail.com'));
    users.push(new User('B', 'B', 'B@gmail.com'));
    users.push(new User('C', 'C', 'C@gmail.com'));
  }
}
```

Untergeordnete Komponente mit Eingabe () Benutzer von der übergeordneten Komponente abrufen

```
@Component({
  selector: 'child-component',
  template: '<div>
    <table *ngIf="users !== null">
      <thead>
        <th>Name</th>
        <th>FName</th>
        <th>Email</th>
      </thead>
      <tbody>
        <tr *ngFor="let user of users">
          <td>{{user.name}}</td>
          <td>{{user.fname}}</td>
          <td>{{user.email}}</td>
        </tr>
      </tbody>
    </table>

    </div>',
})
export class ChildComponent {
  @Input() users : List<User> = null;
}

export class User {
```

```
name : string;
fname : string;
email : string;

constructor(_name : string, _fname : string, _email : string){
  this.name = _name;
  this.fname = _fname;
  this.email = _email;
}
}
```

Einfaches Beispiel für Eingabeeigenschaften

Übergeordnetes Element HTML

```
<child-component [isSelected]="inputPropValue"></child-component>
```

Übergeordnetes Element ts

```
export class AppComponent {
  inputPropValue: true
}
```

Kindkomponente ts:

```
export class ChildComponent {
  @Input() inputPropValue = false;
}
```

Kindkomponente html:

```
<div [class.simpleCssClass]="inputPropValue"></div>
```

Mit diesem Code wird der `inputPropValue` von der übergeordneten Komponente an die untergeordnete Komponente gesendet, und der Wert, den wir in der übergeordneten Komponente festgelegt haben, wird angezeigt, wenn er dort ankommt. Dieser Wert kann dann in der untergeordneten Komponente verwendet werden, um beispielsweise einem Element eine Klasse hinzuzufügen.

Angular2 Eingang () Ausgang () online lesen:

<https://riptutorial.com/de/angular2/topic/8943/angular2-eingang----ausgang--->

Kapitel 13: Angular2 In Memory-Web-API

Bemerkungen

Ich habe dieses Thema hauptsächlich angefordert, weil ich keine Informationen zum Einrichten mehrerer API-Routen mit der Angular2-In-Memory-Web-API finden konnte. Am Ende fand ich es selbst heraus und dachte, das könnte für andere hilfreich sein.

Examples

Grundeinstellung

Scheindaten.ts

Erstellen Sie die Mock-API-Daten

```
export class MockData {
  createDb() {
    let mock = [
      { id: '1', name: 'Object A' },
      { id: '2', name: 'Object B' },
      { id: '3', name: 'Object C' },
      { id: '4', name: 'Object D' }
    ];

    return {mock};
  }
}
```

main.ts

Bitte Sie den Abhängigkeitsinjektor, den InMemoryBackendService für XHRBackend-Anforderungen bereitzustellen. Geben Sie auch eine Klasse an, die a enthält

```
createDb()
```

Funktion (in diesem Fall MockData), die die gespielten API-Routen für SEED_DATA-Anforderungen angibt.

```
import { XHRBackend, HTTP_PROVIDERS } from '@angular/http';
import { InMemoryBackendService, SEED_DATA } from 'angular2-in-memory-web-api';
import { MockData } from './mock-data';
import { bootstrap } from '@angular/platform-browser-dynamic';

import { AppComponent } from './app.component';

bootstrap(AppComponent, [
  HTTP_PROVIDERS,
  { provide: XHRBackend, useClass: InMemoryBackendService },
  { provide: SEED_DATA, useClass: MockData }
])
```

```
]);
```

Schein-Service.ts

Beispiel für das Aufrufen einer Get-Anforderung für die erstellte API-Route

```
import { Injectable } from '@angular/core';
import { Http, Response } from '@angular/http';
import { Mock } from './mock';

@Injectable()
export class MockService {
  // URL to web api
  private mockUrl = 'app/mock';

  constructor (private http: Http) {}

  getData(): Promise<Mock[]> {
    return this.http.get(this.mockUrl)
      .toPromise()
      .then(this.extractData)
      .catch(this.handleError);
  }

  private extractData(res: Response) {
    let body = res.json();
    return body.data || { };
  }

  private handleError (error: any) {
    let errMsg = (error.message) ? error.message :
      error.status ? `${error.status} - ${error.statusText}` : 'Server error';
    console.error(errMsg);
    return Promise.reject(errMsg);
  }
}
```

Einrichten mehrerer Test-API-Routen

Scheindaten.ts

```
export class MockData {
  createDb() {
    let mock = [
      { id: '1', name: 'Object A' },
      { id: '2', name: 'Object B' },
      { id: '3', name: 'Object C' }
    ];

    let data = [
      { id: '1', name: 'Data A' },
      { id: '2', name: 'Data B' },
      { id: '3', name: 'Data C' }
    ];

    return { mock, data };
  }
}
```



```
}
```

Jetzt können Sie mit beiden interagieren

```
app/mock
```

und

```
app/data
```

ihre entsprechenden Daten zu extrahieren.

Angular2 In Memory-Web-API online lesen:

<https://riptutorial.com/de/angular2/topic/6576/angular2-in-memory-web-api>

Kapitel 14: Angular2 mit Webpack

Examples

Angular 2 Webpack-Setup

webpack.config.js

```
const webpack = require("webpack")
const helpers = require('./helpers')
const path = require("path")
const WebpackNotifierPlugin = require('webpack-notifier');

module.exports = {

  // set entry point for your app module
  "entry": {
    "app": helpers.root("app/main.module.ts"),
  },

  // output files to dist folder
  "output": {
    "filename": "[name].js",
    "path": helpers.root("dist"),
    "publicPath": "/",
  },

  "resolve": {
    "extensions": ['.ts', '.js'],
  },

  "module": {
    "rules": [
      {
        "test": /\.ts$/,
        "loaders": [
          {
            "loader": 'awesome-typescript-loader',
            "options": {
              "configFileName": helpers.root("./tsconfig.json")
            }
          },
          "angular2-template-loader"
        ]
      },
    ],
  },

  "plugins": [

    // notify when build is complete
    new WebpackNotifierPlugin({title: "build complete"}),

    // get reference for shims
    new webpack.DllReferencePlugin({
      "context": helpers.root("src/app"),
    })
  ]
}
```

```

        "manifest": helpers.root("config/polyfills-manifest.json")
    }},
    // get reference of vendor DLL
    new webpack.DllReferencePlugin({
        "context": helpers.root("src/app"),
        "manifest": helpers.root("config/vendor-manifest.json")
    }),
    // minify compiled js
    new webpack.optimize.UglifyJsPlugin(),
],
}

```

vendor.config.js

```

const webpack = require("webpack")
const helpers = require('./helpers')
const path = require("path")

module.exports = {
    // specify vendor file where all vendors are imported
    "entry": {
        // optionally add your shims as well
        "polyfills": [helpers.root("src/app/shims.ts")],
        "vendor": [helpers.root("src/app/vendor.ts")],
    },

    // output vendor to dist
    "output": {
        "filename": "[name].js",
        "path": helpers.root("dist"),
        "publicPath": "/",
        "library": "[name]"
    },

    "resolve": {
        "extensions": ['.ts', '.js'],
    },

    "module": {
        "rules": [
            {
                "test": /\.ts$/,
                "loaders": [
                    {
                        "loader": 'awesome-typescript-loader',
                        "options": {
                            "configFileName": helpers.root("./tsconfig.json")
                        }
                    }
                ],
            }
        ],
    },

    "plugins": [

        // create DLL for entries
        new webpack.DllPlugin({

```

```

        "name": "[name]",
        "context": helpers.root("src/app"),
        "path": helpers.root("config/[name]-manifest.json")
    }),

    // minify generated js
    new webpack.optimize.UglifyJsPlugin(),
  ],
}

```

helpers.js

```

var path = require('path');

var _root = path.resolve(__dirname, '..');

function root(args) {
  args = Array.prototype.slice.call(arguments, 0);
  return path.join.apply(path, [_root].concat(args));
}

exports.root = root;

```

Verkäufer.ts

```

import "@angular/platform-browser"
import "@angular/platform-browser-dynamic"
import "@angular/core"
import "@angular/common"
import "@angular/http"
import "@angular/router"
import "@angular/forms"
import "rxjs"

```

index.html

```

<!DOCTYPE html>
<html>
<head>
  <title>Angular 2 webpack</title>

  <script src="/dist/vendor.js" type="text/javascript"></script>
  <script src="/dist/app.js" type="text/javascript"></script>
</head>
<body>
  <app>loading...</app>
</body>
</html>

```

package.json

```

{
  "name": "webpack example",
  "version": "0.0.0",
  "description": "webpack",
  "scripts": {

```

```
"build:webpack": "webpack --config config/webpack.config.js",
"build:vendor": "webpack --config config/vendor.config.js",
"watch": "webpack --config config/webpack.config.js --watch"
},
"devDependencies": {
  "@angular/common": "2.4.7",
  "@angular/compiler": "2.4.7",
  "@angular/core": "2.4.7",
  "@angular/forms": "2.4.7",
  "@angular/http": "2.4.7",
  "@angular/platform-browser": "2.4.7",
  "@angular/platform-browser-dynamic": "2.4.7",
  "@angular/router": "3.4.7",
  "webpack": "^2.2.1",
  "awesome-typescript-loader": "^3.1.2",
},
"dependencies": {
}
}
```

Angular2 mit Webpack online lesen: <https://riptutorial.com/de/angular2/topic/9554/angular2-mit-webpack>

Kapitel 15: Angular2 stellt App vor dem Bootstrap externe Daten bereit

Einführung

In diesem Beitrag werde ich zeigen, wie externe Daten an die Angular-App übergeben werden, bevor die App bootstraps. Diese externen Daten können Konfigurationsdaten, Altdaten, Server-Rendering usw. sein.

Examples

Über Abhängigkeitsinjektion

Packen Sie den Bootstrap-Code nicht direkt in den Angular-Bootstrap-Code, sondern in eine Funktion und exportieren Sie die Funktion. Diese Funktion kann auch Parameter akzeptieren.

```
import { platformBrowserDynamic } from "@angular/platform-browser-dynamic";
import { AppModule } from "../src/app";
export function runAngular2App(legacyModel: any) {
  platformBrowserDynamic([
    { provide: "legacyModel", useValue: model }
  ]).bootstrapModule(AppModule)
  .then(success => console.log("Ng2 Bootstrap success"))
  .catch(err => console.error(err));
}
```

Dann können wir in beliebige Dienste oder Komponenten das "Legacy-Modell" einfügen und darauf zugreifen.

```
import { Injectable } from "@angular/core";
@Injectable()
export class MyService {
  constructor(@Inject("legacyModel") private legacyModel) {
    console.log("Legacy data - ", legacyModel);
  }
}
```

Fordern Sie die App an und führen Sie sie aus.

```
require(["myAngular2App"], function(app) {
  app.runAngular2App(legacyModel); // Input to your APP
});
```

Angular2 stellt App vor dem Bootstrap externe Daten bereit online lesen:

<https://riptutorial.com/de/angular2/topic/9203/angular2-stellt-app-vor-dem-bootstrap-externe-daten-bereit>

Kapitel 16: Angular-cli

Einführung

Hier erfahren Sie, wie Sie mit angle-cli beginnen, neue Komponenten / Services / Pipe / Module mit angle-cli erstellen, 3 Parteien wie Bootstrap hinzufügen und ein Winkelprojekt erstellen.

Examples

Erstellen Sie eine leere Angular2-Anwendung mit angle-cli

Bedarf:

- NodeJS: [Download-Seite](#)
 - [npm](#) oder [garn](#)
-

Führen Sie die folgenden Befehle mit cmd aus dem neuen Ordner aus:

1. `npm install -g @angular/cli` **oder** `yarn global add @angular/cli`
 2. `ng new PROJECT_NAME`
 3. `cd PROJECT_NAME`
 4. `ng serve`
-

Öffnen Sie Ihren Browser unter localhost: 4200

Komponenten, Richtlinien, Pipes und Services generieren

Verwenden Sie einfach Ihren Befehl: Sie können den Befehl `ng generate` (oder einfach `ng g`) verwenden, um Angular-Komponenten zu generieren:

- **Komponente:** `ng g component my-new-component`
- **Direktive:** `ng g directive my-new-directive`
- **Pipe:** `ng g pipe my-new-pipe`
- **Service:** `ng g service my-new-service`
- **Klasse:** `ng g class my-new-classt`
- **Schnittstelle:** `ng g interface my-new-interface`
- `ng g enum my-new-enum` : `ng g enum my-new-enum`
- **Modul:** `ng g module my-module`

Hinzufügen von Drittanbieter-Bibliotheken

In `angle-cli.json` können Sie die App-Konfiguration ändern.

Wenn Sie beispielsweise `ng2-bootstrap` hinzufügen möchten:

1. `npm install ng2-bootstrap --save` **oder** `npm install ng2-bootstrap --save yarn add ng2-bootstrap`
2. Fügen Sie in `angle-cli.json` einfach den Pfad des Bootstraps bei `node-modules` hinzu.

```
"scripts": [  
  "../node_modules/jquery/dist/jquery.js",  
  "../node_modules/bootstrap/dist/js/bootstrap.js"  
]
```

bauen mit `eckig-cli`

In `angle-cli.json` unter `outDir` key können Sie Ihr Build-Verzeichnis definieren.

diese sind gleichwertig

```
ng build --target=production --environment=prod  
ng build --prod --env=prod  
ng build --prod
```

und so sind diese

```
ng build --target=development --environment=dev  
ng build --dev --e=dev  
ng build --dev  
ng build
```

Beim Erstellen können Sie das Basis-Tag (`<base>`) in Ihrer `index.html` mit der Option `--base-href your-url` ändern.

Setzt das Basis-Tag `href` in Ihrer `index.html` auf `/ myUrl /`

```
ng build --base-href /myUrl/  
ng build --bh /myUrl/
```

Neues Projekt mit `scss` / `sass` als Stylesheet

Die von `@angular/cli` generierten und kompilierten `@angular/cli` sind **css** .

Wenn Sie stattdessen **scss** verwenden **möchten** , generieren Sie Ihr Projekt mit:

```
ng new project_name --style=scss
```

Wenn Sie **sass** verwenden **möchten** , generieren Sie Ihr Projekt mit:

```
ng new project_name --style=sass
```

Legen Sie das Garn als Standardpaketmanager für `@ angle / cli` fest

Yarn ist eine Alternative für npm, den Standardpaketmanager für @ angle / cli. Wenn Sie Yarn als Paketmanager für @ angle / cli verwenden möchten, gehen Sie folgendermaßen vor:

Bedarf

- [Yarn](#) (`npm install --global yarn` oder siehe [Installationsseite](#))
- [@ eckig / cli](#) (`npm install -g @angular/cli` oder `yarn global add @angular/cli`)

Yarn als @ angle / cli Paketmanager einstellen:

```
ng set --global packageManager=yarn
```

Um npm als @ angle / cli-Paketmanager zurückzusetzen:

```
ng set --global packageManager=npm
```

[Angular-cli online lesen: https://riptutorial.com/de/angular2/topic/8956/angular-cli](https://riptutorial.com/de/angular2/topic/8956/angular-cli)

Kapitel 17: Angulares Materialdesign

Examples

Md2Select

Komponente :

```
<md2-select [(ngModel)]="item" (change)="change($event)" [disabled]="disabled">
<md2-option *ngFor="let i of items" [value]="i.value" [disabled]="i.disabled">
  {{i.name}}</md2-option>
</md2-select>
```

Wählen Sie die Option aus, dass der Benutzer eine Option aus den Optionen auswählen kann.

```
<md2-select></md2-select>
<md2-option></md2-option>
<md2-select-header></md2-select-header>
```

Md2Tooltip

Die QuickInfo ist eine Direktive, mit der der Benutzer Hinweistext anzeigen kann, während der Mauszeiger über ein Element bewegt wird.

Ein Tooltip hätte die folgende Markierung.

```
<span tooltip-direction="left" tooltip="On the Left!">Left</span>
<button tooltip="some message"
  tooltip-position="below"
  tooltip-delay="1000">Hover Me
</button>
```

Md2Toast

Toast ist ein Dienst, der Benachrichtigungen in der Ansicht anzeigt.

Erzeugt und zeigt eine einfache Toastbenachrichtigung.

```
import {Md2Toast} from 'md2/toast/toast';

@Component({
  selector: "...",
})

export class ... {

  ...
  constructor(private toast: Md2Toast) { }
```

```
toastMe() {
  this.toast.show('Toast message...');

  --- or ---

  this.toast.show('Toast message...', 1000);
}

...

}
```

Md2Datepicker

Mit Datepicker kann der Benutzer Datum und Uhrzeit auswählen.

```
<md2-datepicker [(ngModel)]="date"></md2-datepicker>
```

Weitere Details finden Sie [hier](#)

Md2Accordion und Md2Collapse

Md2Collapse : Collapse ist eine Direktive, mit der der Benutzer die Sichtbarkeit des Abschnitts umschalten kann.

Beispiele

Ein Zusammenbruch hätte den folgenden Markup.

```
<div [collapse]="isCollapsed">
  Lorem Ipsum Content
</div>
```

Md2Accordion : Accordion ermöglicht es dem Benutzer, die Sichtbarkeit der verschiedenen Abschnitte zu ändern.

Beispiele

Ein Akkordeon hätte folgendes Markup.

```
<md2-accordion [multiple]="multiple">
  <md2-accordion-tab *ngFor="let tab of accordions"
    [header]="tab.title"
    [active]="tab.active"
    [disabled]="tab.disabled">
    {{tab.content}}
  </md2-accordion-tab>
  <md2-accordion-tab>
    <md2-accordion-header>Custom Header</md2-accordion-header>
    test content
  </md2-accordion-tab>
</md2-accordion>
```

Angulares Materialdesign online lesen: <https://riptutorial.com/de/angular2/topic/10005/angulares-materialdesign>

Kapitel 18: Animation

Examples

Übergang zwischen Nullzuständen

```
@Component({
  ...
  animations: [
    trigger('appear', [
      transition(':enter', [
        style({
          //style applied at the start of animation
        }),
        animate('300ms ease-in', style({
          //style applied at the end of animation
        }))
      ])
    ])
  ]
})
class AnimComponent {
}
]
```

Animieren zwischen mehreren Zuständen

Das `<div>` in dieser Vorlage wächst auf `50px` und dann auf `50px` und `100px` dann auf `20px` wenn Sie auf die Schaltfläche klicken.

Jedem `state` ist ein Stil zugeordnet, der in den `@Component` Metadaten beschrieben wird.

Die Logik für den jeweils aktiven `state` kann in der Komponentenlogik verwaltet werden. In diesem Fall enthält die `size` der Komponentenvariablen den Zeichenfolgenwert "small", "medium" oder "large".

Das `<div>` Element auf diesen Wert durch die reagiert `trigger` in den angegebenen `@Component` Metadaten: `[@size]="size"` .

```
@Component({
  template: '<div [@size]="size">Some Text</div><button
(click)="toggleSize()">TOGGLE</button>',
  animations: [
    trigger('size', [
      state('small', style({
        height: '20px'
      })),
      state('medium', style({
        height: '50px'
      })),
      state('large', style({
```

```
        height: '100px'
      })),
      transition('small => medium', animate('100ms')),
      transition('medium => large', animate('200ms')),
      transition('large => small', animate('300ms'))
    ]
  })
}
export class TestComponent {

  size: string;

  constructor(){
    this.size = 'small';
  }
  toggleSize(){
    switch(this.size) {
      case 'small':
        this.size = 'medium';
        break;
      case 'medium':
        this.size = 'large';
        break;
      case 'large':
        this.size = 'small';
    }
  }
}
```

Animation online lesen: <https://riptutorial.com/de/angular2/topic/8127/animation>

Kapitel 19: AOT-Compilierung mit Angular 2

Examples

1. Installieren Sie die Abhängigkeiten von Angular 2 mit dem Compiler

HINWEIS: Um optimale Ergebnisse zu erzielen, stellen Sie sicher, dass Ihr Projekt mit der Angular-CLI erstellt wurde.

```
npm install angular/{core,common,compiler,platform-browser,platform-browser-  
dynamic,http,router,forms,compiler-cli,tsc-wrapped,platform-server}
```

Sie müssen diesen Schritt nicht ausführen, wenn in Ihrem Projekt bereits Winkel 2 und alle diese Abhängigkeiten installiert sind. Stellen Sie einfach sicher, dass sich der `compiler` dort befindet.

2. Fügen Sie der Datei "tsconfig.json" die Option "angleCompilerOptions" hinzu

```
...  
"angularCompilerOptions": {  
  "genDir": "./ngfactory"  
}  
...
```

Dies ist der Ausgabeordner des Compilers.

3. Führen Sie den Winkelcompiler `ngc` aus

Von der Wurzel Ihres Projekts `./node_modules/.bin/ngc -p src wo src gesamte`
`./node_modules/.bin/ngc -p src 2` ist. Dadurch wird ein Ordner namens `ngfactory` in dem der
gesamte kompilierte Code `ngfactory` wird.

`"node_modules/.bin/ngc" -p src` für Windows

4. Ändern Sie die Datei "main.ts", um den Browser NgFactory und die statische Plattform zu verwenden

```
// this is the static platform browser, the usual counterpart is @angular/platform-browser-  
dynamic.  
import { platformBrowser } from '@angular/platform-browser';  
  
// this is generated by the angular compiler  
import { AppModuleNgFactory } from './ngfactory/app/app.module.ngfactory';  
  
// note the use of `bootstrapModuleFactory`, as opposed to `bootstrapModule`.  
platformBrowser().bootstrapModuleFactory(AppModuleNgFactory);
```

An diesem Punkt sollten Sie in der Lage sein, Ihr Projekt auszuführen. In diesem Fall wurde mein Projekt mit der Angular-CLI erstellt.

```
> ng serve
```

Warum brauchen wir Zusammenstellung, Zusammenstellung von Ereignissen und Beispiel?

Warum brauchen wir eine Kompilierung? Ans. Wir benötigen eine Zusammenstellung, um eine höhere Effizienz unserer Angular-Anwendungen zu erreichen.

Schauen Sie sich das folgende Beispiel an:

```
// ...
compile: function (el, scope) {
  var dirs = this._getElDirectives(el);
  var dir;
  var scopeCreated;
  dirs.forEach(function (d) {
    dir = Provider.get(d.name + Provider.DIRECTIVES_SUFFIX);
    if (dir.scope && !scopeCreated) {
      scope = scope.$new();
      scopeCreated = true;
    }
    dir.link(el, scope, d.value);
  });
  Array.prototype.slice.call(el.children).forEach(function (c) {
    this.compile(c, scope);
  }, this);
},
// ...
```

Verwenden Sie den obigen Code, um die Vorlage zu rendern.

```
<ul>
  <li *ngFor="let name of names"></li>
</ul>
```

Ist viel langsamer im Vergleich zu:

```
// ...
this._text_9 = this.renderer.createText(this._el_3, '\n', null);
this._text_10 = this.renderer.createText(parentRenderNode, '\n\n', null);
this._el_11 = this.renderer.createElement(parentRenderNode, 'ul', null);
this._text_12 = this.renderer.createText(this._el_11, '\n ', null);
this._anchor_13 = this.renderer.createTemplateAnchor(this._el_11, null);
this._appEl_13 = new import2.AppElement(13, 11, this, this._anchor_13);
this._TemplateRef_13_5 = new import17.TemplateRef_(this._appEl_13,
viewFactory_HomeComponent1);
this._NgFor_13_6 = new import15.NgFor(this._appEl_13.vcRef, this._TemplateRef_13_5,
this.parentInjector.get(import18.IterableDiffers), this.ref);
// ...
```

Ablauf von Ereignissen mit Vorausberechnung

Im Gegensatz dazu werden mit AoT die folgenden Schritte durchlaufen:

1. Entwicklung der Angular 2-Anwendung mit TypeScript.
2. Erstellung der Anwendung mit ngc.
3. Führt die Kompilierung der Vorlagen mit dem Angular-Compiler zu TypeScript durch.
4. Kompilierung des TypeScript-Codes in JavaScript.
5. Bündeln
6. Minifizierung
7. Einsatz.

Obwohl der obige Vorgang etwas komplizierter erscheint, führt der Benutzer nur die folgenden Schritte durch:

1. Laden Sie alle Assets herunter.
2. Angular Bootstraps.
3. Die Anwendung wird gerendert.

Wie Sie sehen, fehlt der dritte Schritt, was eine schnellere / bessere UX bedeutet. Außerdem werden Tools wie angle2-seed und angle-cli den Build-Prozess erheblich automatisieren.

Ich hoffe es könnte dir helfen! Vielen Dank!

AoT-Kompilierung mit Angular CLI verwenden

Die [Angular CLI](#)- Befehlszeilenschnittstelle unterstützt seit der Beta 17 AoT-Kompilierung.

Um Ihre App mit AoT-Kompilierung zu erstellen, führen Sie einfach Folgendes aus:

```
ng build --prod --aot
```

AOT-Kompilierung mit Angular 2 online lesen: <https://riptutorial.com/de/angular2/topic/6634/aot-kompilierung-mit-angular-2>

Kapitel 20: Attributanweisungen, um den Wert von Eigenschaften auf dem Hostknoten mithilfe des @ HostBinding-Dekors zu beeinflussen

Examples

@HostBinding

Mit dem @HostBinding-Dekorator können wir einen Eigenschaftswert für das Hostelement der Anweisung programmgesteuert festlegen. Es funktioniert ähnlich wie eine Eigenschaftsbindung, die in einer Vorlage definiert ist, außer es richtet sich speziell an das Hostelement. Die Bindung wird für jeden Änderungserkennungszyklus überprüft, sodass sie sich bei Bedarf dynamisch ändern kann. Nehmen wir beispielsweise an, wir möchten eine Direktive für Schaltflächen erstellen, die eine Klasse dynamisch hinzufügt, wenn Sie darauf drücken. Das könnte ungefähr so aussehen:

```
import { Directive, HostBinding, HostListener } from '@angular/core';

@Directive({
  selector: '[appButtonPress]'
})
export class ButtonPressDirective {
  @HostBinding('attr.role') role = 'button';
  @HostBinding('class.pressed') isPressed: boolean;

  @HostListener('mousedown') hasPressed() {
    this.isPressed = true;
  }
  @HostListener('mouseup') hasReleased() {
    this.isPressed = false;
  }
}
```

Beachten Sie, dass wir für beide Anwendungsfälle von @HostBinding einen String-Wert übergeben, für den die Eigenschaft relevant ist. Wenn wir dem Dekorateur keinen String zur Verfügung stellen, wird stattdessen der Name des Klassenmitglieds verwendet. In der ersten @HostBinding setzen wir das Rollenattribut statisch auf button. Im zweiten Beispiel wird die gedrückte Klasse angewendet, wenn isPressed true ist

Attributanweisungen, um den Wert von Eigenschaften auf dem Hostknoten mithilfe des @HostBinding-Dekors zu beeinflussen online lesen:

<https://riptutorial.com/de/angular2/topic/9455/attributanweisungen--um-den-wert-von-eigenschaften-auf-dem-hostknoten-mithilfe-des---hostbinding-dekors-zu-beeinflussen>

Kapitel 21: Beispiel für Routen wie / route / subroute für statische URLs

Examples

Grundlegendes Routenbeispiel mit Unterroutenbaum

app.module.ts

```
import {routes} from "./app.routes";

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule, mainModule.forRoot(), RouterModule.forRoot(routes)],
  providers: [],
  bootstrap: [AppComponent]
})

export class AppModule { }
```

app.routes.ts

```
import { Routes } from '@angular/router';
import {SubTreeRoutes} from "./subTree/subTreeRoutes.routes";

export const routes: Routes = [
  ...SubTreeRoutes,
  { path: '', redirectTo: 'home', pathMatch: 'full' }
];
```

subTreeRoutes.ts

```
import {Route} from '@angular/router';
import {exampleComponent} from "./example.component";

export const SubTreeRoutes: Route[] = [
  {
    path: 'subTree',
    children: [
      {path: '', component: exampleComponent}
    ]
  }
];
```

Beispiel für Routen wie / route / subroute für statische URLs online lesen:

<https://riptutorial.com/de/angular2/topic/8910/beispiel-fur-routen-wie---route---subroute-fur-statische-urls>

Kapitel 22: Beispiele für erweiterte Komponenten

Bemerkungen

Denken Sie daran, dass es bei Angular 2 um singuläre Verantwortung geht. Unabhängig davon, wie klein Ihre Komponente ist, widmen Sie jeder Komponente eine eigene Logik. Sei es eine Schaltfläche, eine originelle Ankerverbindung, eine Dialogkopfzeile oder sogar ein Unterelement von Sidenav.

Examples

Bildauswahl mit Vorschau

In diesem Beispiel erstellen wir eine Bildauswahl, die Ihr Bild vor dem Hochladen in der Vorschau anzeigt. Der Previewer unterstützt auch das Ziehen und Ablegen von Dateien in die Eingabe. In diesem Beispiel werde ich nur das Hochladen einzelner Dateien behandeln, aber Sie können ein wenig basteln, um das Hochladen mehrerer Dateien zu ermöglichen.

image-preview.html

Dies ist das HTML-Layout unserer Bildvorschau

```
<!-- Icon as placeholder when no file picked -->
<i class="material-icons">cloud_upload</i>

<!-- file input, accepts images only. Detect when file has been picked/changed with Angular's
native (change) event listener -->
<input type="file" accept="image/*" (change)="updateSource($event)">

<!-- img placeholder when a file has been picked. shows only when 'source' is not empty -->
<img *ngIf="source" [src]="source" src="">
```

image-preview.ts

Dies ist die Hauptdatei für unsere Komponente `<image-preview>`

```
import {
  Component,
  Output,
  EventEmitter,
} from '@angular/core';

@Component({
  selector: 'image-preview',
  styleUrls: [ './image-preview.css' ],
  templateUrl: './image-preview.html'
})
```

```

export class MtImagePreviewComponent {

  // Emit an event when a file has been picked. Here we return the file itself
  @Output() onChange: EventEmitter<File> = new EventEmitter<File>();

  constructor() {}

  // If the input has changed(file picked) we project the file into the img previewer
  updateSource($event: Event) {
    // We access he file with $event.target['files'][0]
    this.projectImage($event.target['files'][0]);
  }

  // Uses FileReader to read the file from the input
  source:string = '';
  projectImage(file: File) {
    let reader = new FileReader;
    // TODO: Define type of 'e'
    reader.onload = (e: any) => {
      // Simply set e.target.result as our <img> src in the layout
      this.source = e.target.result;
      this.onChange.emit(file);
    };
    // This will process our file and get it's attributes/data
    reader.readAsDataURL(file);
  }
}

```

eine andere.komponente.html

```

<form (ngSubmit)="submitPhoto()">
  <image-preview (onChange)="getFile($event)"></image-preview>
  <button type="submit">Upload</button>
</form>

```

Und das ist es. Viel einfacher als in AngularJS 1.x. Ich habe diese Komponente auf der Grundlage einer älteren Version erstellt, die ich in AngularJS 1.5.5 erstellt habe.

Tabellenwerte über die Eingabe herausfiltern

Importieren Sie `ReactiveFormsModule` und dann

```

import { Component, OnInit, OnDestroy } from '@angular/core';
import { FormControl } from '@angular/forms';
import { Subscription } from 'rxjs';

@Component({
  selector: 'component',
  template: `
    <input [formControl]="control" />
    <div *ngFor="let item of content">
      {{item.id}} - {{item.name}}
    </div>
  `
})
export class MyComponent implements OnInit, OnDestroy {

```

```
public control = new FormControl('');

public content: { id: number; name: string; }[];

private originalContent = [
  { id: 1, name: 'abc' },
  { id: 2, name: 'abce' },
  { id: 3, name: 'ced' }
];

private subscription: Subscription;

public ngOnInit() {
  this.subscription = this.control.valueChanges.subscribe(value => {
    this.content = this.originalContent.filter(item => item.name.startsWith(value));
  });
}

public ngOnDestroy() {
  this.subscription.unsubscribe();
}
}
```

Beispiele für erweiterte Komponenten online lesen:

<https://riptutorial.com/de/angular2/topic/5597/beispiele-fur-erweiterte-komponenten>

Kapitel 23: benutzerdefinierte ngx-bootstrap datepicker + input

Examples

benutzerdefinierte ngx-bootstrap datepicker

datepicker.component.html

```
<div (clickOutside)="onClickedOutside($event)" (blur)="onClickedOutside($event)">
  <div class="input-group date" [ngClass]="{'disabled-icon': disabledDatePicker == false}">
    <input (change)="changedDate()" type="text" [ngModel]="value" class="form-control"
    id="{{id}}" (focus)="openCloseDatePicker()" disabled="{{disabledInput}}" />
    <span id="openCloseDatePicker" class="input-group-addon"
    (click)="openCloseDatePicker()">
      <span class="glyphicon-calendar glyphicon"></span>
    </span>
  </div>

  <div class="dp-popup" *ngIf="showDatePicker">
    <datepicker [startingDay]="1" [startingDay]="dt" [minDate]="min" [(ngModel)]="dt"
    (selectionDone)="onSelectionDone($event)"></datepicker>
  </div>
</div>
```

datepicker.component.ts

```
import {Component, Input, EventEmitter, Output, OnChanges, SimpleChanges, ElementRef, OnInit}
from "@angular/core";
import {DatePipe} from "@angular/common";
import {NgModel} from "@angular/forms";
import * as moment from 'moment';

@Component({
  selector: 'custom-datepicker',
  templateUrl: 'datepicker.component.html',
  providers: [DatePipe, NgModel],
  host: {
    '(document:mousedown)': 'onClick($event)',
  }
})

export class DatepickerComponent implements OnChanges , OnInit{
  ngOnInit(): void {
    this.dt = null;
  }

  inputElement : ElementRef;
  dt: Date = null;
  showDatePicker: boolean = false;

  @Input() disabledInput : boolean = false;
```

```

@Input() disabledDatePicker: boolean = false;
@Input() value: string = null;
@Input() id: string;
@Input() min: Date = null;
@Input() max: Date = null;

@Output() dateModelChange = new EventEmitter();
constructor(el: ElementRef) {
  this.inputElement = el;
}

changedDate(){
  if(this.value === ''){
    this.dateModelChange.emit(null);
  }else if(this.value.split('/').length === 3){
    this.dateModelChange.emit(DatepickerComponent.convertToDate(this.value));
  }
}

clickOutside(event : Event){
  if(this.inputElement.nativeElement !== event.target) {
    console.log('click outside', event);
  }
}

onClick(event) {
  if (!this.inputElement.nativeElement.contains(event.target)) {
    this.close();
  }
}

ngOnChanges(changes: SimpleChanges): void {
  if (this.value !== null && this.value !== undefined && this.value.length > 0) {
    this.value = null;
    this.dt = null;
  }else {
    if(this.value !== null){
      this.dt = new Date(this.value);
      this.value = moment(this.value).format('MM/DD/YYYY');
    }
  }
}

private static transformDate(date: Date): string {
  return new DatePipe('pt-PT').transform(date, 'MM/dd/yyyy');
}

openCloseDatepicker(): void {
  if (!this.disabledDatePicker) {
    this.showDatepicker = !this.showDatepicker;
  }
}

open(): void {
  this.showDatepicker = true;
}

close(): void {
  this.showDatepicker = false;
}

private apply(): void {

```



```
    this.value = DatepickerComponent.transformDate(this.dt);
    this.dateModelChange.emit(this.dt);
  }

  onSelectionDone(event: Date): void {
    this.dt = event;
    this.apply();
    this.close();
  }

  onClickedOutside(event: Date): void {
    if (this.showDatepicker) {
      this.close();
    }
  }

  static convertToDate(val : string): Date {
    return new Date(val.replace('/', '-'));
  }
}
```

benutzerdefinierte ngx-bootstrap datepicker + input online lesen:

<https://riptutorial.com/de/angular2/topic/10549/benutzerdefinierte-ngx-bootstrap-datepicker-plus-input>

Kapitel 24: Bootstrap Leeres Modul in Winkel 2

Examples

Ein leeres Modul

```
import { NgModule } from '@angular/core';

@NgModule({
  declarations: [], // components your module owns.
  imports: [], // other modules your module needs.
  providers: [], // providers available to your module.
  bootstrap: [] // bootstrap this root component.
})
export class MyModule {}
```

Dies ist ein leeres Modul, das keine Deklarationen, Importe, Anbieter oder Komponenten für den Bootstrap enthält. Verwenden Sie dies als Referenz.

Ein Modul mit Vernetzung im Webbrowser.

```
// app.module.ts

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/http';
import { MyRootComponent } from './app.component';

@NgModule({
  declarations: [MyRootComponent],
  imports: [BrowserModule, HttpClientModule],
  bootstrap: [MyRootComponent]
})
export class MyModule {}
```

`MyRootComponent` ist die in `MyModule` gepackte `MyModule`. Es ist der Einstiegspunkt in Ihre Angular 2-Anwendung.

Bootstrapping deines Moduls

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { MyModule } from './app.module';

platformBrowserDynamic().bootstrapModule( MyModule );
```

In diesem Beispiel ist `MyModule` das Modul, das Ihre `MyModule` enthält. Durch Bootstrapping von `MyModule` Ihre Angular 2-App einsatzbereit.

Anwendungsstammmodul

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent }  from './app.component';

@NgModule({
  imports: [ BrowserModule ],
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Statisches Bootstrapping mit Factory-Klassen

Wir können eine Anwendung statisch bootstrappen, indem Sie die einfache ES5-Javascript-Ausgabe der generierten Factory-Klassen verwenden. Dann können wir diese Ausgabe verwenden, um die Anwendung zu booten:

```
import { platformBrowser } from '@angular/platform-browser';
import { AppModuleNgFactory } from './main.ngfactory';

// Launch with the app module factory.
platformBrowser().bootstrapModuleFactory(AppModuleNgFactory);
```

Dies führt dazu, dass das Anwendungspaket wesentlich kleiner ist, da die gesamte Vorlagenkompilierung bereits in einem Erstellungsschritt durchgeführt wurde, entweder mit `ngc` oder direktem Aufruf der internen Daten.

Bootstrap Leeres Modul in Winkel 2 online lesen:

<https://riptutorial.com/de/angular2/topic/5508/bootstrap-leeres-modul-in-winkel-2>

Kapitel 25: Brute Force Upgrade

Einführung

Wenn Sie die Angular-CLI-Version Ihres Projekts aktualisieren möchten, können Sie durch das Ändern der Angular-CLI-Versionsnummer in Ihrem Projekt schwer zu behebende Fehler und Fehler feststellen. Da die Angular-CLI eine Menge verbirgt, was im Build- und Bundles-Prozess vor sich geht, können Sie nicht viel tun, wenn dort Probleme auftreten.

Manchmal ist es am einfachsten, die Angular CLI-Version des Projekts zu aktualisieren, indem Sie mit der Angular CLI-Version, die Sie verwenden möchten, ein neues Projekt erstellen.

Bemerkungen

Da Angular 2 so modular und gekapselt ist, können Sie einfach alle Ihre Komponenten, Services, Pipes, Anweisungen kopieren und das NgModule so ausfüllen, wie es im alten Projekt war.

Examples

Gerüstbau eines neuen CLI-Projekts

```
ng new NewProject
```

oder

```
ng init NewProject
```

Brute Force Upgrade online lesen: <https://riptutorial.com/de/angular2/topic/9152/brute-force-upgrade>

Kapitel 26: CRUD in Angular2 mit Restful-API

Syntax

- `@Injectable ()` // Teilt dem Abhängigkeitsinjektor das Einfügen von Abhängigkeiten beim Erstellen einer Instanz dieses Dienstes mit.
- `request.subscribe ()` // Hier stellen Sie *tatsächlich* die Anfrage. Andernfalls wird Ihre Anfrage nicht gesendet. Sie möchten auch die Antwort in der Rückruffunktion lesen.
- Konstruktor (`privates WikiService: WikipediaService`) {} // Da sowohl unser Service als auch seine Abhängigkeiten vom Abhängigkeitseinspritzer injizierbar sind, empfiehlt es sich, den Service in die Komponente einzuspeisen, um die App zu testen.

Examples

Aus einer Restful-API in Angular2 lesen

Um die API-Logik von der Komponente zu trennen, erstellen wir den API-Client als separate Klasse. Diese Beispielklasse fordert eine Anfrage an die Wikipedia-API, um zufällige Wiki-Artikel abzurufen.

```
import { Http, Response } from '@angular/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs/Observable';
import 'rxjs/Rx';

@Injectable()
export class WikipediaService{
  constructor(private http: Http) {}

  getRandomArticles(numberOfArticles: number)
  {
    var request =
this.http.get("https://en.wikipedia.org/w/api.php?action=query&list=random&format=json&rnlimit="
+ numberOfArticles);
    return request.map((response: Response) => {
      return response.json();
    }, (error) => {
      console.log(error);
      //you want to implement your own error handling here.
    });
  }
}
```

Und haben eine Komponente, um unseren neuen API-Client zu nutzen.

```
import { Component, OnInit } from '@angular/core';
import { WikipediaService } from './wikipedia.Service';
```

```
@Component({
  selector: 'wikipedia',
  templateUrl: 'wikipedia.component.html'
})
export class WikipediaComponent implements OnInit {
  constructor(private wikiService: WikipediaService) { }

  private articles: any[] = null;
  ngOnInit() {
    var request = this.wikiService.getRandomArticles(5);
    request.subscribe((res) => {
      this.articles = res.query.random;
    });
  }
}
```

CRUD in Angular2 mit Restful-API online lesen:

<https://riptutorial.com/de/angular2/topic/7343/crud-in-angular2-mit-restful-api>

Kapitel 27: Debuggen der Angular2-Typoskriptanwendung mit Visual Studio Code

Examples

Launch.json-Setup für Ihren Arbeitsbereich

1. Aktivieren Sie Debug aus Menü - Ansicht> Debug
2. Es gibt einige Fehler beim Start-Debugging zurück, Popup-Benachrichtigung anzeigen und launch.json von dieser Popup-Benachrichtigung aus öffnen. kopiere und füge den folgenden Code in launch.json // new launch.json ein

Ihr alter Launch.json

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Launch Extension",
      "type": "extensionHost",
      "request": "launch",
      "runtimeExecutable": "${execPath}",
      "args": [
        "--extensionDevelopmentPath=${workspaceRoot}"
      ],
      "stopOnEntry": false,
      "sourceMaps": true,
      "outDir": "${workspaceRoot}/out",
      "preLaunchTask": "npm"
    }
  ]
}
```

Aktualisieren Sie nun Ihre launch.json wie unten beschrieben

neuer launch.json

// Erinnern Sie sich bitte an Ihren main.js-Pfad.

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Launch",
      "type": "node",
      "request": "launch",
      "program": "${workspaceRoot}/app/main.js", // put your main.js path
      "stopOnEntry": false,
      "args": [],
      "cwd": "${workspaceRoot}",
    }
  ]
}
```

```

    "preLaunchTask": null,
    "runtimeExecutable": null,
    "runtimeArgs": [
      "--nolazy"
    ],
    "env": {
      "NODE_ENV": "development"
    },
    "console": "internalConsole",
    "sourceMaps": false,
    "outDir": null
  },
  {
    "name": "Attach",
    "type": "node",
    "request": "attach",
    "port": 5858,
    "address": "localhost",
    "restart": false,
    "sourceMaps": false,
    "outDir": null,
    "localRoot": "${workspaceRoot}",
    "remoteRoot": null
  },
  {
    "name": "Attach to Process",
    "type": "node",
    "request": "attach",
    "processId": "${command.PickProcess}",
    "port": 5858,
    "sourceMaps": false,
    "outDir": null
  }
]
}

```

3. Jetzt funktioniert das Debuggen. Zeigen Sie das Popup-Fenster für das schrittweise Debuggen an

Debuggen der Angular2-Typoskriptanwendung mit Visual Studio Code online lesen:
<https://riptutorial.com/de/angular2/topic/7139/debuggen-der-angular2-typoskriptanwendung-mit-visual-studio-code>

Kapitel 28: Dienste und Abhängigkeitsinjektion

Examples

Beispieldienst

services / mein.service.ts

```
import { Injectable } from '@angular/core';

@Injectable()
export class MyService {
  data: any = [1, 2, 3];

  getData() {
    return this.data;
  }
}
```

Die Registrierung des Service Providers in der Bootstrap-Methode macht den Service global verfügbar.

main.ts

```
import { bootstrap } from '@angular/platform-browser-dynamic';
import { AppComponent } from 'app.component.ts';
import { MyService } from 'services/my.service';

bootstrap(AppComponent, [MyService]);
```

In der Version RC5 kann der globale Service Provider in der Moduldatei registriert werden. Um eine einzige Instanz Ihres Dienstes für Ihre gesamte Anwendung zu erhalten, muss der Dienst in der Liste der Anbieter im ngmodule Ihrer Anwendung angegeben werden. *app_module.ts*

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { routing, appRoutingProviders } from './app-routes/app.routes';
import { HttpClientModule } from '@angular/http';

import { AppComponent } from './app.component';
import { MyService } from 'services/my.service';

import { routing } from './app-resources/app-routes/app.routes';

@NgModule({
  declarations: [ AppComponent ],
  imports: [ BrowserModule,
            routing,
            RouterModule,
            HttpClientModule ],
```

```

    providers: [    appRoutingProviders,
                  MyService
    ],
    bootstrap:    [AppComponent],
  })
  export class AppModule {}

```

Verwendung in `MyComponent`

components / my.component.ts

Alternativer Ansatz zum Registrieren von Anwendungsanbietern in Anwendungskomponenten. Wenn wir bei der Wiedergabe der Komponente Anbieter auf Komponentenebene hinzufügen, wird eine neue Instanz des Diensts erstellt.

```

import { Component, OnInit } from '@angular/core';
import { MyService } from '../services/my.service';

@Component({
  ...
  providers:[MyService] //
})
export class MyComponent implements OnInit {
  data: any[];
  // Creates private variable myService to use, of type MyService
  constructor(private myService: MyService) { }

  ngOnInit() {
    this.data = this.myService.getData();
  }
}

```

Beispiel mit `Promise.resolve`

services / mein.service.ts

```

import { Injectable } from '@angular/core';

@Injectable()
export class MyService {
  data: any = [1, 2, 3];

  getData() {
    return Promise.resolve(this.data);
  }
}

```

`getData()` jetzt wie ein REST-Aufruf, der ein Promise erstellt, das sofort aufgelöst wird. Die Ergebnisse können in `.then()` und Fehler erkannt werden. Dies ist eine gute Praxis und Konvention für asynchrone Methoden.

components / my.component.ts

```

import { Component, OnInit } from '@angular/core';
import { MyService } from '../services/my.service';

@Component({...})
export class MyComponent implements OnInit {
  data: any[];
  // Creates private variable myService to use, of type MyService
  constructor(private myService: MyService) { }

  ngOnInit() {
    // Uses an "arrow" function to set data
    this.myService.getData().then(data => this.data = data);
  }
}

```

Testen eines Dienstes

Bei einem Dienst, mit dem sich ein Benutzer anmelden kann:

```

import 'rxjs/add/operator/toPromise';

import { Http } from '@angular/http';
import { Injectable } from '@angular/core';

interface LoginCredentials {
  password: string;
  user: string;
}

@Injectable()
export class AuthService {
  constructor(private http: Http) { }

  async signIn({ user, password }: LoginCredentials) {
    const response = await this.http.post('/login', {
      password,
      user,
    }).toPromise();

    return response.json();
  }
}

```

Es kann so getestet werden:

```

import { ConnectionBackend, Http, HttpModule, Response, ResponseOptions } from
 '@angular/http';
import { TestBed, async, inject } from '@angular/core/testing';

import { AuthService } from './auth.service';
import { MockBackend } from '@angular/http/testing';
import { MockConnection } from '@angular/http/testing';

describe('AuthService', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [HttpModule],
      providers: [

```

```

    AuthService,
    Http,
    { provide: ConnectionBackend, useClass: MockBackend },
  ]
});
});

it('should be created', inject([AuthService], (service: AuthService) => {
  expect(service).toBeTruthy();
}));

// Alternative 1
it('should login user if right credentials are passed', async(
  inject([AuthService], async (authService) => {
    const backend: MockBackend = TestBed.get(ConnectionBackend);
    const http: Http = TestBed.get(Http);

    backend.connections.subscribe((c: MockConnection) => {
      c.mockRespond(
        new Response(
          new ResponseOptions({
            body: {
              accessToken: 'abcdef',
            },
          }),
        ),
      );
    });

    const result = await authService.signIn({ password: 'ok', user: 'bruno' });

    expect(result).toEqual({
      accessToken: 'abcdef',
    });
  })
);

// Alternative 2
it('should login user if right credentials are passed', async () => {
  const backend: MockBackend = TestBed.get(ConnectionBackend);
  const http: Http = TestBed.get(Http);

  backend.connections.subscribe((c: MockConnection) => {
    c.mockRespond(
      new Response(
        new ResponseOptions({
          body: {
            accessToken: 'abcdef',
          },
        }),
      ),
    );
  });

  const authService: AuthService = TestBed.get(AuthService);

  const result = await authService.signIn({ password: 'ok', user: 'bruno' });

  expect(result).toEqual({
    accessToken: 'abcdef',
  });
};

```

```

});

// Alternative 3
it('should login user if right credentials are passed', async (done) => {
  const authService: AuthService = TestBed.get(AuthService);

  const backend: MockBackend = TestBed.get(ConnectionBackend);
  const http: Http = TestBed.get(Http);

  backend.connections.subscribe((c: MockConnection) => {
    c.mockRespond(
      new Response(
        new ResponseOptions({
          body: {
            accessToken: 'abcdef',
          },
        }),
      ),
    );
  });

  try {
    const result = await authService.signIn({ password: 'ok', user: 'bruno' });

    expect(result).toEqual({
      accessToken: 'abcdef',
    });

    done();
  } catch (err) {
    fail(err);
    done();
  }
});
});

```

Dienste und Abhängigkeitsinjektion online lesen:

<https://riptutorial.com/de/angular2/topic/4187/dienste-und-abhangigkeitsinjektion>

Kapitel 29: Dropzone in Angular2

Examples

Abwurfgebiet

Angular 2 Wrapper-Bibliothek für Dropzone.

```
npm install angle2-dropzone-wrapper --save-dev
```

Laden Sie das Modul für Ihr App-Modul

```
import { DropzoneModule } from 'angular2-dropzone-wrapper';
import { DropzoneConfigInterface } from 'angular2-dropzone-wrapper';

const DROPZONE_CONFIG: DropzoneConfigInterface = {
  // Change this to your upload POST address:
  server: 'https://example.com/post',
  maxFileSize: 10,
  acceptedFiles: 'image/*'
};

@NgModule({
  ...
  imports: [
    ...
    DropzoneModule.forRoot(DROPZONE_CONFIG)
  ]
})
```

KOMPONENTENVERWENDUNG

Ersetzen Sie einfach das Element, das andernfalls an Dropzone übergeben würde, durch die Dropzone-Komponente.

```
<dropzone [config]="config" [message]='Click or drag images here to upload'
(error)="onUploadError($event)" (success)="onUploadSuccess($event)"></dropzone>
```

Dropzone-Komponente erstellen

```
import {Component} from '@angular/core';
@Component({
  selector: 'app-new-media',
  templateUrl: './dropzone.component.html',
  styleUrls: ['./dropzone.component.scss']
})
export class DropZoneComponent {

  onUploadError(args: any) {
    console.log('onUploadError:', args);
  }
}
```

```
onUploadSuccess(args: any) {  
    console.log('onUploadSuccess:', args);  
}  
}
```

Dropzone in Angular2 online lesen: <https://riptutorial.com/de/angular2/topic/10010/dropzone-in-angular2>

Kapitel 30: Dynamisches Hinzufügen von Komponenten mithilfe von `ViewContainerRef.createComponent`

Examples

Eine Wrapper-Komponente, die dynamische Komponenten deklarativ hinzufügt

Eine benutzerdefinierte Komponente, die den Typ einer Komponente als Eingabe übernimmt und eine Instanz dieses Komponententyps in sich selbst erstellt. Wenn die Eingabe aktualisiert wird, wird die zuvor hinzugefügte dynamische Komponente entfernt und stattdessen die neue hinzugefügt.

```
@Component ({
  selector: 'dcl-wrapper',
  template: `

https://riptutorial.com/de/home



95


```



```

ngOnDestroy() {
  if (this.cmpRef) {
    this.cmpRef.destroy();
  }
}
}

```

So können Sie dynamische Komponenten wie

```
<dcl-wrapper [type]="someComponentType"></dcl-wrapper>
```

Plunker Beispiel

Dynamisches Hinzufügen einer Komponente zu einem bestimmten Ereignis (Klicken)

Hauptkomponenten-Datei:

```

//our root app component
import {Component, NgModule, ViewChild, ViewContainerRef, ComponentFactoryResolver,
ComponentRef} from '@angular/core'
import {BrowserModule} from '@angular/platform-browser'
import {ChildComponent} from './childComp.ts'

@Component({
  selector: 'my-app',
  template: `
    <div>
      <h2>Hello {{name}}</h2>
      <input type="button" value="Click me to add element" (click) = addElement()> // call the
function on click of the button
      <div #parent> </div> // Dynamic component will be loaded here
    </div>
  `,
})
export class App {
  name:string;

  @ViewChild('parent', {read: ViewContainerRef}) target: ViewContainerRef;
  private componentRef: ComponentRef<any>;

  constructor(private componentFactoryResolver: ComponentFactoryResolver) {
    this.name = 'Angular2'
  }

  addElement(){
    let childComponent =
this.componentFactoryResolver.resolveComponentFactory(ChildComponent);
    this.componentRef = this.target.createComponent(childComponent);
  }
}

```

childComp.ts:

```
import {Component} from '@angular/core';

@Component({
  selector: 'child',
  template: `
    <p>This is Child</p>
  `,
})
export class ChildComponent {
  constructor() {

  }
}
```

app.module.ts:

```
@NgModule({
  imports: [ BrowserModule ],
  declarations: [ App, ChildComponent ],
  bootstrap: [ App ],
  entryComponents: [ChildComponent] // define the dynamic component here in module.ts
})
export class AppModule {}
```

Plunker Beispiel

Dynamisch erstelltes Komponenten-Array für Template-HTML in Angular2

Wir können dynamische Komponenten erstellen und die Instanzen der Komponenten in ein Array bringen und schließlich als Vorlage darstellen.

Zum Beispiel können wir zwei Widgetkomponenten betrachten, ChartWidget und PatientWidget, die die Klasse WidgetComponent erweitert haben, die ich im Container hinzufügen wollte.

ChartWidget.ts

```
@Component({
  selector: 'chart-widget',
  templateUrl: 'chart-widget.component.html',
  providers: [{provide: WidgetComponent, useExisting: forwardRef(() => ChartWidget) }]
})

export class ChartWidget extends WidgetComponent implements OnInit {
  constructor(ngEl: ElementRef, renderer: Renderer) {
    super(ngEl, renderer);
  }
  ngOnInit() {}
  close() {
    console.log('close');
  }
  refresh() {
    console.log('refresh');
  }
  ...
}
```

chart-widget.component.html (mit dem primeng Panel)

```
<p-panel [style]="{'margin-bottom':'20px'}">
  <p-header>
    <div class="ui-helper-clearfix">
      <span class="ui-panel-title" style="font-size:14px;display:inline-block;margin-top:2px">Chart Widget</span>
      <div class="ui-toolbar-group-right">
        <button pButton type="button" icon="fa-window-minimize"
(click)="minimize()"></button>
        <button pButton type="button" icon="fa-refresh" (click)="refresh()"></button>
        <button pButton type="button" icon="fa-expand" (click)="expand()" ></button>
        <button pButton type="button" (click)="close()" icon="fa-window-close"></button>
      </div>
    </div>
  </p-header>
  some data
</p-panel>
```

DataWidget.ts

```
@Component({
  selector: 'data-widget',
  templateUrl: 'data-widget.component.html',
  providers: [{provide: WidgetComponent, useExisting: forwardRef(() =>DataWidget) }]
})

export class DataWidget extends WidgetComponent implements OnInit {
  constructor(ngEl: ElementRef, renderer: Renderer) {
    super(ngEl, renderer);
  }
  ngOnInit() {}
  close(){
    console.log('close');
  }
  refresh(){
    console.log('refresh');
  }
  ...
}
```

data-widget.component.html (wie Chart-Widget mit dem primeng Panel)

WidgetComponent.ts

```
@Component({
  selector: 'widget',
  template: '<ng-content></ng-content>'
})
export class WidgetComponent{
}
```

Wir können dynamische Komponenteninstanzen erstellen, indem Sie die bereits vorhandenen Komponenten auswählen. Zum Beispiel,

```
@Component({
```

```

    selector: 'dynamic-component',
    template: `<div #container><ng-content></ng-content></div>`
  })
  export class DynamicComponent {
    @ViewChild('container', {read: ViewContainerRef}) container: ViewContainerRef;

    public addComponent(ngItem: Type<WidgetComponent>): WidgetComponent {
      let factory = this.compFactoryResolver.resolveComponentFactory(ngItem);
      const ref = this.container.createComponent(factory);
      const newItem: WidgetComponent = ref.instance;
      this._elements.push(newItem);
      return newItem;
    }
  }
}

```

Schließlich verwenden wir es in der App-Komponente. `app.component.ts`

```

@Component({
  selector: 'app-root',
  templateUrl: './app/app.component.html',
  styleUrls: ['./app/app.component.css'],
  entryComponents: [ChartWidget, DataWidget],
})

export class AppComponent {
  private elements: Array<WidgetComponent>=[];
  private WidgetClasses = {
    'ChartWidget': ChartWidget,
    'DataWidget': DataWidget
  }
  @ViewChild(DynamicComponent) dynamicComponent:DynamicComponent;

  addComponent(widget: string ): void{
    let ref= this.dynamicComponent.addComponent(this.WidgetClasses[widget]);
    this.elements.push(ref);
    console.log(this.elements);

    this.dynamicComponent.resetContainer();
  }
}

```

`app.component.html`

```

<button (click)="addComponent('ChartWidget')">Add ChartWidget</button>
<button (click)="addComponent('DataWidget')">Add DataWidget</button>

<dynamic-component [hidden]="true" ></dynamic-component>

<hr>
Dynamic Components
<hr>
<widget *ngFor="let item of elements">
  <div>{{item}}</div>
  <div [innerHTML]="item._ngEl.nativeElement.innerHTML | sanitizeHtml">
  </div>
</widget>

```

<https://plnkr.co/edit/lugU2pPsSBd3XhPHiUP1?p=preview>

Eine Änderung von @yurzui zur Verwendung des Mausereignisses in den Widgets

view.directive.ts

importiere {ViewRef, Direktive, Eingabe, ViewContainerRef} von '@ angle / core';

```
@Directive({
  selector: '[view]'
})
export class ViewDirective {
  constructor(private vcRef: ViewContainerRef) {}

  @Input()
  set view(view: ViewRef) {
    this.vcRef.clear();
    this.vcRef.insert(view);
  }

  ngOnDestroy() {
    this.vcRef.clear();
  }
}
```

app.component.ts

```
private elements: Array<{ view: ViewRef, component: WidgetComponent}> = [];

...
addComponent(widget: string ): void{
  let component = this.dynamicComponent.addComponent(this.WidgetClasses[widget]);
  let view: ViewRef = this.dynamicComponent.container.detach(0);
  this.elements.push({view, component});

  this.dynamicComponent.resetContainer();
}
```

app.component.html

```
<widget *ngFor="let item of elements">
  <ng-container *view="item.view"></ng-container>
</widget>
```

<https://plnkr.co/edit/JHpIHR43SvJd0OxJVMfV?p=preview>

Dynamisches Hinzufügen von Komponenten mithilfe von ViewContainerRef.createComponent
online lesen: <https://riptutorial.com/de/angular2/topic/831/dynamisches-hinzufugen-von-komponenten-mithilfe-von-viewcontainerref-createcomponent>

Kapitel 31: eckiger Redux

Examples

Basic

app.module.ts

```
import {appStoreProviders} from "../app.store";
providers : [
  ...
  appStoreProviders,
  ...
]
```

app.store.ts

```
import {InjectionToken} from '@angular/core';
import {createStore, Store, compose, StoreEnhancer} from 'redux';
import {AppState, default as reducer} from "../app.reducer";

export const AppStore = new InjectionToken('App.store');

const devtools: StoreEnhancer<AppState> =
  window['devToolsExtension'] ?
  window['devToolsExtension']() : f => f;

export function createAppStore(): Store<AppState> {
  return createStore<AppState>(
    reducer,
    compose(devtools)
  );
}

export const appStoreProviders = [
  {provide: AppStore, useFactory: createAppStore}
];
```

app.reducer.ts

```
export interface AppState {
  example : string
}

const rootReducer: Reducer<AppState> = combineReducers<AppState>({
  example : string
});

export default rootReducer;
```

store.ts

```

export interface IAppState {
  example?: string;
}

export const INITIAL_STATE: IAppState = {
  example: null,
};

export function rootReducer(state: IAppState = INITIAL_STATE, action: Action): IAppState {
  switch (action.type) {
    case EXAMPLE_CHANGED:
      return Object.assign(state, state, (<UpdateAction>action));
    default:
      return state;
  }
}

```

actions.ts

```

import {Action} from "redux";
export const EXAMPLE_CHANGED = 'EXAMPLE CHANGED';

export interface UpdateAction extends Action {
  example: string;
}

```

Holen Sie sich den aktuellen Stand

```

import * as Redux from 'redux';
import {Inject, Injectable} from '@angular/core';

@Injectable()
export class exampleService {
  constructor(@Inject(AppStore) private store: Redux.Store<AppState>) {}
  getExampleState(){
    console.log(this.store.getState().example);
  }
}

```

Zustand ändern

```

import * as Redux from 'redux';
import {Inject, Injectable} from '@angular/core';

@Injectable()
export class exampleService {
  constructor(@Inject(AppStore) private store: Redux.Store<AppState>) {}
  setExampleState(){
    this.store.dispatch(updateExample("new value"));
  }
}

```

actions.ts

```

export interface UpdateExapleAction extends Action {

```

```
    example?: string;
  }

export const updateExample: ActionCreator<UpdateExapleAction> =
  (newVal) => ({
    type: EXAMPLE_CHANGED,
    example: newVal
  });
```

Fügen Sie das Redux-Chrome-Tool hinzu

app.store.ts

```
import {InjectionToken} from '@angular/core';
import {createStore, Store, compose, StoreEnhancer} from 'redux';
import {AppState, default as reducer} from "../app.reducer";

export const AppStore = new InjectionToken('App.store');

const devtools: StoreEnhancer<AppState> =
  window['devToolsExtension'] ?
  window['devToolsExtension']() : f => f;

export function createAppStore(): Store<AppState> {
  return createStore<AppState>(
    reducer,
    compose(devtools)
  );
}

export const appStoreProviders = [
  {provide: AppStore, useFactory: createAppStore}
];
```

Installieren Sie die Redux DevTools Chromverlängerung

eckiger Redux online lesen: <https://riptutorial.com/de/angular2/topic/10652/eckiger-redux>

Kapitel 32: Ermitteln von Größenänderungsereignissen

Examples

Eine Komponente, die das Fenster zur Größenänderung des Fensters überwacht.

Angenommen, wir haben eine Komponente, die bei einer bestimmten Fensterbreite ausgeblendet wird.

```
import { Component } from '@angular/core';

@Component({
  ...
  template: `
    <div>
      <p [hidden]="!visible" (window:resize)="onResize($event)" >Now you see me...</p>
      <p>now you dont!</p>
    </div>
  `
  ...
})
export class MyComponent {
  visible: boolean = false;
  breakpoint: number = 768;

  constructor() {
  }

  onResize(event) {
    const w = event.target.innerWidth;
    if (w >= this.breakpoint) {
      this.visible = true;
    } else {
      // whenever the window is less than 768, hide this component.
      this.visible = false;
    }
  }
}
```

Ein `p` Tag in unserer Vorlage wird ausgeblendet, wenn `visible false` ist. `visible` ändert den Wert, wenn der `onResize` Ereignishandler aufgerufen wird. Der Aufruf erfolgt bei jedem `window:resize` wird ein Ereignis `window:resize`.

Ermitteln von Größenänderungsereignissen online lesen:

<https://riptutorial.com/de/angular2/topic/5276/ermitteln-von-gro-enanderungsereignissen>

Kapitel 33: Erstellen einer Angular-npm-Bibliothek

Einführung

So veröffentlichen Sie Ihr NgModule in TypeScript in der npm-Registrierung. Einrichten des npm-Projekts, Typoscript-Compilers, Rollups und Continuous Integration.

Examples

Minimalmodul mit Serviceklasse

Dateistruktur

```
/
  -src/
    awesome.service.ts
    another-awesome.service.ts
    awesome.module.ts
  -index.ts
  -tsconfig.json
  -package.json
  -rollup.config.js
  -.npmignore
```

Service und Modul

Platziere deine großartige Arbeit hier.

src / awesome.service.ts:

```
export class AwesomeService {
  public doSomethingAwesome(): void {
    console.log("I am so awesome!");
  }
}
```

src / awesome.module.ts:

```
import { NgModule } from '@angular/core'
import { AwesomeService } from './awesome.service';
import { AnotherAwesomeService } from './another-awesome.service';

@NgModule({
  providers: [AwesomeService, AnotherAwesomeService]
```

```
)  
export class AwesomeModule {}
```

Machen Sie Ihr Modul und Ihren Service von außen zugänglich.

/index.ts:

```
export { AwesomeService } from './src/awesome.service';  
export { AnotherAwesomeService } from './src/another-awesome.service';  
export { AwesomeModule } from './src/awesome.module';
```

Zusammenstellung

In `compilerOptions.paths` müssen Sie alle externen Module angeben, die Sie in Ihrem Paket verwendet haben.

/tsconfig.json

```
{  
  "compilerOptions": {  
    "baseUrl": ".",  
    "declaration": true,  
    "stripInternal": true,  
    "experimentalDecorators": true,  
    "strictNullChecks": false,  
    "noImplicitAny": true,  
    "module": "es2015",  
    "moduleResolution": "node",  
    "paths": {  
      "@angular/core": ["node_modules/@angular/core"],  
      "rxjs/*": ["node_modules/rxjs/*"]  
    },  
    "rootDir": ".",  
    "outDir": "dist",  
    "sourceMap": true,  
    "inlineSources": true,  
    "target": "es5",  
    "skipLibCheck": true,  
    "lib": [  
      "es2015",  
      "dom"  
    ]  
  },  
  "files": [  
    "index.ts"  
  ],  
  "angularCompilerOptions": {  
    "strictMetadataEmit": true  
  }  
}
```

Geben Sie Ihre Externals erneut an

/rollup.config.js

```

export default {
  entry: 'dist/index.js',
  dest: 'dist/bundles/awesome.module.umd.js',
  sourceMap: false,
  format: 'umd',
  moduleName: 'ng.awesome.module',
  globals: {
    '@angular/core': 'ng.core',
    'rxjs': 'Rx',
    'rxjs/Observable': 'Rx',
    'rxjs/ReplaySubject': 'Rx',
    'rxjs/add/operator/map': 'Rx.Observable.prototype',
    'rxjs/add/operator/mergeMap': 'Rx.Observable.prototype',
    'rxjs/add/observable/fromEvent': 'Rx.Observable',
    'rxjs/add/observable/of': 'Rx.Observable'
  },
  external: ['@angular/core', 'rxjs']
}

```

NPM-Einstellungen

Nun können wir einige Anweisungen für npm geben

/package.json

```

{
  "name": "awesome-angular-module",
  "version": "1.0.4",
  "description": "Awesome angular module",
  "main": "dist/bundles/awesome.module.umd.min.js",
  "module": "dist/index.js",
  "typings": "dist/index.d.ts",
  "scripts": {
    "test": "",
    "transpile": "ngc",
    "package": "rollup -c",
    "minify": "uglifyjs dist/bundles/awesome.module.umd.js --screw-ie8 --compress --mangle --
comments --output dist/bundles/awesome.module.umd.min.js",
    "build": "rimraf dist && npm run transpile && npm run package && npm run minify",
    "prepublishOnly": "npm run build"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/maciejtreder/awesome-angular-module.git"
  },
  "keywords": [
    "awesome",
    "angular",
    "module",
    "minimal"
  ],
  "author": "Maciej Treder <contact@maciejtreder.com>",
  "license": "MIT",
  "bugs": {
    "url": "https://github.com/maciejtreder/awesome-angular-module/issues"
  },
  "homepage": "https://github.com/maciejtreder/awesome-angular-module#readme",

```

```
"devDependencies": {
  "@angular/compiler": "^4.0.0",
  "@angular/compiler-cli": "^4.0.0",
  "rimraf": "^2.6.1",
  "rollup": "^0.43.0",
  "typescript": "^2.3.4",
  "uglify-js": "^3.0.21"
},
"dependencies": {
  "@angular/core": "^4.0.0",
  "rxjs": "^5.3.0"
}
}
```

Wir können auch angeben, welche Dateien von npm ignoriert werden sollen

.npmignore

```
node_modules
npm-debug.log
Thumbs.db
.DS_Store
src
!dist/src
plugin
!dist/plugin
*.ngsummary.json
*.iml
rollup.config.js
tsconfig.json
*.ts
!*d.ts
.idea
```

Kontinuierliche Integration

Schließlich können Sie den Continuous Integration Build einrichten

.travis.yml

```
language: node_js
node_js:
- node

deploy:
  provider: npm
  email: contact@maciejtreder.com
  api_key:
    secure: <your api key>
  on:
    tags: true
    repo: maciejtreder/awesome-angular-module
```

Die Demo finden Sie hier: <https://github.com/maciejtreder/awesome-angular-module>

Erstellen einer Angular-npm-Bibliothek online lesen:

<https://riptutorial.com/de/angular2/topic/10704/erstellen-einer-angular-npm-bibliothek>

Kapitel 34: Erstellen Sie ein Angular 2+ NPM-Paket

Einführung

Manchmal müssen wir einige Komponenten für einige Apps freigeben. Die Veröffentlichung in npm ist eine der besten Möglichkeiten, dies zu tun.

Es gibt einige Tricks, die wir wissen müssen, um eine normale Komponente als npm-Paket verwenden zu können, ohne die Struktur als Inlining externer Stile zu ändern.

Sie können hier ein minimales Beispiel [sehen](#)

Examples

Einfachstes Paket

Hier stellen wir einige minimale Arbeitsabläufe vor, um ein Angular 2+ npm-Paket zu erstellen und zu veröffentlichen.

Konfigurationsdateien

Wir benötigen einige Konfigurationsdateien, um `git`, `npm`, `gulp` und `typescript` mitzuteilen, wie sie vorgehen sollen.

`.gitignore`

Zuerst erstellen wir eine `.gitignore` Datei, um die Versionierung unerwünschter Dateien und Ordner zu vermeiden. Der Inhalt ist:

```
npm-debug.log
node_modules
jspm_packages
.idea
build
```

`.npmignore`

Zweitens erstellen wir eine `.npmignore` -Datei, um zu vermeiden, dass unerwünschte Dateien und Ordner veröffentlicht werden. Der Inhalt ist:

```
examples
node_modules
src
```

gulpfile.js

Wir müssen eine `gulpfile.js` erstellen, um Gulp mitzuteilen, wie er unsere Anwendung kompiliert. Dieser Teil ist notwendig, da vor der Veröffentlichung unseres Pakets alle externen Vorlagen und Stile minimiert und integriert werden müssen. Der Inhalt ist:

```
var gulp = require('gulp');
var embedTemplates = require('gulp-angular-embed-templates');
var inlineNg2Styles = require('gulp-inline-ng2-styles');

gulp.task('js:build', function () {
  gulp.src('src/*.ts') // also can use *.js files
    .pipe(embedTemplates({sourceType:'ts'}))
    .pipe(inlineNg2Styles({ base: '/src' }))
    .pipe(gulp.dest('./dist'));
});
```

index.d.ts

Die Datei `index.d.ts` wird beim Importieren eines externen Moduls von `index.d.ts` verwendet. Es hilft dem Editor bei der automatischen Vervollständigung und Funktionstipps.

```
export * from './lib';
```

index.js

Dies ist der Paketeinstiegspunkt. Wenn Sie dieses Paket mit NPM installieren und in Ihre Anwendung importieren, müssen Sie nur den Paketnamen übergeben, und Ihre Anwendung erfährt, wo sich eine EXPORTED-Komponente Ihres Pakets befindet.

```
exports.AngularXMinimalNpmPackageModule = require('./lib').AngularXMinimalNpmPackageModule;
```

Wir haben den `lib` Ordner verwendet, weil die Ausgabe beim Kompilieren des Codes im `/lib` Ordner abgelegt wird.

package.json

Diese Datei wird zur Konfiguration Ihrer npm-Publikation verwendet und definiert die erforderlichen Pakete, um zu funktionieren.

```
{
  "name": "angular-x-minimal-npm-package",
  "version": "0.0.18",
  "description": "An Angular 2+ Data Table that uses HTTP to create, read, update and delete data from an external API such REST.",
  "main": "index.js",
  "scripts": {
    "watch": "tsc -p src -w",
    "build": "gulp js:build && rm -rf lib && tsc -p dist"
  },
}
```



```

"repository": {
  "type": "git",
  "url": "git+https://github.com/vinagreti/angular-x-minimal-npm-package.git"
},
"keywords": [
  "Angular",
  "Angular2",
  "Datatable",
  "Rest"
],
"author": "bruno@tzadi.com",
"license": "MIT",
"bugs": {
  "url": "https://github.com/vinagreti/angular-x-minimal-npm-package/issues"
},
"homepage": "https://github.com/vinagreti/angular-x-minimal-npm-package#readme",
"devDependencies": {
  "gulp": "3.9.1",
  "gulp-angular-embed-templates": "2.3.0",
  "gulp-inline-ng2-styles": "0.0.1",
  "typescript": "2.0.0"
},
"dependencies": {
  "@angular/common": "2.4.1",
  "@angular/compiler": "2.4.1",
  "@angular/core": "2.4.1",
  "@angular/http": "2.4.1",
  "@angular/platform-browser": "2.4.1",
  "@angular/platform-browser-dynamic": "2.4.1",
  "rxjs": "5.0.2",
  "zone.js": "0.7.4"
}
}

```

dist / tsconfig.json

Erstellen Sie einen dist-Ordner und legen Sie diese Datei darin ab. Diese Datei wird verwendet, um Typescript mitzuteilen, wie Sie Ihre Anwendung kompilieren. Woher bekomme ich den Typescript-Ordner und wo die kompilierten Dateien abgelegt werden sollen?

```

{
  "compilerOptions": {
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "mapRoot": "",
    "rootDir": ".",
    "target": "es5",
    "lib": ["es6", "es2015", "dom"],
    "inlineSources": true,
    "stripInternal": true,
    "module": "commonjs",
    "moduleResolution": "node",
    "removeComments": true,
    "sourceMap": true,
    "outDir": "../lib",
    "declaration": true
  }
}

```

Nach dem Erstellen der Konfigurationsdateien müssen wir unsere Komponente und unser Modul erstellen. Diese Komponente erhält einen Klick und zeigt eine Nachricht an. Es wird wie ein HTML-Tag `<angular-x-minimal-npm-package></angular-x-minimal-npm-package>` . Installieren Sie einfach dieses npm-Paket und laden Sie sein Modul in das Modell, das Sie verwenden möchten.

src / angle-x-minimal-npm-package.component.ts

```
import {Component} from '@angular/core';
@Component({
  selector: 'angular-x-minimal-npm-package',
  styleUrls: ['./angular-x-minimal-npm-package.component.scss'],
  templateUrl: './angular-x-minimal-npm-package.component.html'
})
export class AngularXMinimalNpmPackageComponent {
  message = "Click Me ...";
  onClick() {
    this.message = "Angular 2+ Minimal NPM Package. With external scss and html!";
  }
}
```

src / angle-x-minimal-npm-package.component.html

```
<div>
  <h1 (click)="onClick()">{{message}}</h1>
</div>
```

src / angle-x-data-table.component.css

```
h1{
  color: red;
}
```

src / angle-x-minimal-npm-package.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

import { AngularXMinimalNpmPackageComponent } from './angular-x-minimal-npm-
package.component';

@NgModule({
  imports: [ CommonModule ],
  declarations: [ AngularXMinimalNpmPackageComponent ],
  exports: [ AngularXMinimalNpmPackageComponent ],
  entryComponents: [ AngularXMinimalNpmPackageComponent ],
})
export class AngularXMinimalNpmPackageModule {}
```

Danach müssen Sie Ihr Paket kompilieren, erstellen und veröffentlichen.

Bauen und kompilieren

Für Build verwenden wir `gulp` und zum Kompilieren `tsc`. Der Befehl wird in der `package.json`-Datei `scripts.build` Option `scripts.build`. Wir haben dieses Set `gulp js:build && rm -rf lib && tsc -p dist`. Dies sind unsere Kettenaufgaben, die die Arbeit für uns erledigen werden.

Führen Sie zum Erstellen und Kompilieren den folgenden Befehl im Stammverzeichnis Ihres Pakets aus:

```
npm run build
```

Dadurch wird die Kette ausgelöst, und am Ende befinden sich der Build im Ordner `/dist` und das kompilierte Paket im Ordner `/lib`. Aus diesem Grund `index.js` wir in `index.js` den Code aus dem Ordner `/lib` und nicht aus `/src` exportiert.

Veröffentlichen

Jetzt müssen wir nur noch unser Paket veröffentlichen, damit wir es über npm installieren können. Führen Sie dazu einfach den Befehl aus:

```
npm publish
```

Das ist alles!!!

Erstellen Sie ein Angular 2+ NPM-Paket online lesen:

<https://riptutorial.com/de/angular2/topic/8790/erstellen-sie-ein-angular-2plus-npm-paket>

Kapitel 35: EventEmitter-Dienst

Examples

Klassenübersicht

```
class EventEmitter extends Subject {
  constructor(isAsync?: boolean)
  emit(value?: T)
  subscribe(generatorOrNext?: any, error?: any, complete?: any) : any
}
```

Klassenkomponente

```
@Component({
  selector: 'zippy',
  template: `
<div class="zippy">
  <div (click)="toggle()">Toggle</div>
  <div [hidden]="!visible">
    <ng-content></ng-content>
  </div>
</div>`})
export class Zippy {
  visible: boolean = true;
  @Output() open: EventEmitter<any> = new EventEmitter();
  @Output() close: EventEmitter<any> = new EventEmitter();
  toggle() {
    this.visible = !this.visible;
    if (this.visible) {
      this.open.emit(null);
    } else {
      this.close.emit(null);
    }
  }
}
```

Ereignisse ausstatten

```
<zippy (open)="onOpen($event)" (close)="onClose($event)"></zippy>
```

Die Veranstaltung abfangen

Erstellen Sie einen Service

```
import {EventEmitter} from 'angular2/core';
export class NavService {
  navchange: EventEmitter<number> = new EventEmitter();
  constructor() {}
  emitNavChangeEvent(number) {
    this.navchange.emit(number);
  }
}
```

```

    }
    getNavChangeEmitter() {
        return this.navchange;
    }
}

```

Erstellen Sie eine Komponente, um die Service-

```

import {Component} from 'angular2/core';
import {NavService} from '../services/NavService';

@Component({
  selector: 'obs-comp',
  template: `obs component, item: {{item}}`
})
export class ObservingComponent {
  item: number = 0;
  subscription: any;
  constructor(private navService:NavService) {}
  ngOnInit() {
    this.subscription = this.navService.getNavChangeEmitter()
      .subscribe(item => this.selectedNavItem(item));
  }
  selectedNavItem(item: number) {
    this.item = item;
  }
  ngOnDestroy() {
    this.subscription.unsubscribe();
  }
}

@Component({
  selector: 'my-nav',
  template: `
    <div class="nav-item" (click)="selectedNavItem(1)">nav 1 (click me)</div>
    <div class="nav-item" (click)="selectedNavItem(2)">nav 2 (click me)</div>
  `,
})
export class Navigation {
  item = 1;
  constructor(private navService:NavService) {}
  selectedNavItem(item: number) {
    console.log('selected nav item ' + item);
    this.navService.emitNavChangeEvent(item);
  }
}

```

Live-Beispiel

Ein Live-Beispiel dafür finden Sie [hier](#) .

EventEmitter-Dienst online lesen: <https://riptutorial.com/de/angular2/topic/9159/eventemitter-dienst>

Kapitel 36: Fass

Einführung

Mit einem Fass können Sie den Export mehrerer ES2015-Module in einem einzigen ES2015-Komfortmodul zusammenfassen. Das Fass selbst ist eine ES2015-Moduldatei, die ausgewählte Exporte anderer ES2015-Module erneut exportiert.

Examples

Fass verwenden

Beispielsweise würde ein Verbraucher ohne Fass drei Importanweisungen benötigen:

```
import { HeroComponent } from '../heroes/hero.component.ts';
import { Hero }          from '../heroes/hero.model.ts';
import { HeroService }  from '../heroes/hero.service.ts';
```

Wir können ein Fass hinzufügen, indem Sie eine Datei in demselben Komponentenordner erstellen. In diesem Fall heißt der Ordner "Helden" namens `index.ts` (unter Verwendung der Konventionen), der alle diese Elemente exportiert:

```
export * from './hero.model.ts'; // re-export all of its exports
export * from './hero.service.ts'; // re-export all of its exports
export { HeroComponent } from './hero.component.ts'; // re-export the named thing
```

Nun kann ein Verbraucher das, was er braucht, aus dem Fass importieren.

```
import { Hero, HeroService } from '../heroes/index';
```

Dies kann jedoch zu einer sehr langen Linie werden. Was könnte weiter reduziert werden.

```
import * as h from '../heroes/index';
```

Das ist ziemlich reduziert! Das `* as h` importiert alle Module und Aliasnamen als `h`

Fass online lesen: <https://riptutorial.com/de/angular2/topic/10717/fass>

Kapitel 37: Fauler Laden eines Moduls

Examples

Lazy Loading Beispiel

Durch **Lazy-Loading-** Module verkürzen wir die Startzeit. Beim langsamen Laden muss unsere Anwendung nicht alles auf einmal laden, sondern nur das laden, was der Benutzer erwartet, wenn die App zum ersten Mal geladen wird. Module, die nur langsam geladen werden, werden nur geladen, wenn der Benutzer zu seinen Routen navigiert.

app / app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { EagerComponent } from './eager.component';
import { routing } from './app.routing';
@NgModule({
  imports: [
    BrowserModule,
    routing
  ],
  declarations: [
    AppComponent,
    EagerComponent
  ],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

app / app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'my-app',
  template: `<h1>My App</h1>    <nav>
    <a routerLink="eager">Eager</a>
    <a routerLink="lazy">Lazy</a>
  </nav>
  <router-outlet></router-outlet>
`
})
export class AppComponent {}
```

app / app.routing.ts

```
import { ModuleWithProviders } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { EagerComponent } from './eager.component';
const routes: Routes = [
  { path: '', redirectTo: 'eager', pathMatch: 'full' },
```

```
{ path: 'eager', component: EagerComponent },
{ path: 'lazy', loadChildren: './lazy.module' }
];
export const routing: ModuleWithProviders = RouterModule.forRoot(routes);
```

app / eager.component.ts

```
import { Component } from '@angular/core';
@Component({
  template: `

Eager Component</p>`
})
export class EagerComponent {}


```

Es gibt nichts Besonderes an LazyModule, außer dass es über ein eigenes Routing und eine Komponente namens LazyComponent verfügt.

app / lazy.module.ts

```
import { NgModule } from '@angular/core';
import { LazyComponent } from './lazy.component';
import { routing } from './lazy.routing';
@NgModule({
  imports: [routing],
  declarations: [LazyComponent]
})
export class LazyModule {}
```

app / lazy.routing.ts

```
import { ModuleWithProviders } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { LazyComponent } from './lazy.component';
const routes: Routes = [
  { path: '', component: LazyComponent }
];
export const routing: ModuleWithProviders = RouterModule.forChild(routes);
```

app / lazy.component.ts

```
import { Component } from '@angular/core';
@Component({
  template: `

Lazy Component</p>`
})
export class LazyComponent {}


```

Faules Laden eines Moduls online lesen: <https://riptutorial.com/de/angular2/topic/7751/faules-laden-eines-moduls>

Kapitel 38: Funktionsmodule

Examples

Ein Funktionsmodul

```
// my-feature.module.ts
import { CommonModule } from '@angular/common';
import { NgModule }      from '@angular/core';

import { MyComponent } from './my.component';
import { MyDirective } from './my.directive';
import { MyPipe }      from './my.pipe';
import { MyService }   from './my.service';

@NgModule({
  imports:      [ CommonModule ],
  declarations: [ MyComponent, MyDirective, MyPipe ],
  exports:     [ MyComponent ],
  providers:   [ MyService ]
})
export class MyFeatureModule { }
```

Nun in Ihrem root (normalerweise `app.module.ts`):

```
// app.module.ts
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent }  from './app.component';
import { MyFeatureModule } from './my-feature.module';

@NgModule({
  // import MyFeatureModule in root module
  imports:      [ BrowserModule, MyFeatureModule ],
  declarations: [ AppComponent ],
  bootstrap:   [ AppComponent ]
})
export class AppModule { }
```

Funktionsmodule online lesen: <https://riptutorial.com/de/angular2/topic/6551/funktionsmodule>

Kapitel 39: Geräteprüfung

Examples

Grundeinheitstest

Komponentendatei

```
@Component ({
  selector: 'example-test-compnent',
  template: '<div>
    <div>{{user.name}}</div>
    <div>{{user.fname}}</div>
    <div>{{user.email}}</div>
  </div>'
})

export class ExampleTestComponent implements OnInit{

  let user :User = null;
  ngOnInit(): void {
    this.user.name = 'name';
    this.user.fname= 'fname';
    this.user.email= 'email';
  }
}
```

Testdatei

```
describe('Example unit test component', () => {
  let component: ExampleTestComponent ;
  let fixture: ComponentFixture<ExampleTestComponent >;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ExampleTestComponent]
    }).compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(ExampleTestComponent );
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('ngOnInit should change user object values', () => {
    expect(component.user).toBeNull(); // check that user is null on initialize
    component.ngOnInit(); // run ngOnInit

    expect(component.user.name).toEqual('name');
    expect(component.user.fname).toEqual('fname');
```

```
    expect(component.user.email).toEqual('email');  
  });  
});
```

Geräteprüfung online lesen: <https://riptutorial.com/de/angular2/topic/8955/gerateprufung>

Kapitel 40: Häufig eingebaute Richtlinien und Dienste

Einführung

@ angle / common - häufig benötigte Richtlinien und Dienste @ angle / core - das Winkel-Kern-Framework

Examples

Standortklasse

Location ist ein Dienst, den Anwendungen zur Interaktion mit der Browser-URL verwenden können. Abhängig davon, welche LocationStrategy verwendet wird, verbleibt Location entweder am Pfad der URL oder am Hash-Segment der URL.

Der Standort ist für die Normalisierung der URL gegenüber der Basis-Href der Anwendung verantwortlich.

```
import {Component} from '@angular/core';
import {Location} from '@angular/common';

@Component({
  selector: 'app-component'
})
class AppCmp {

  constructor(_location: Location) {

    //Changes the browsers URL to the normalized version of the given URL,
    //and pushes a new item onto the platform's history.
    _location.go('/foo');

  }

  backClicked() {
    //Navigates back in the platform's history.
    this._location.back();
  }

  forwardClicked() {
    //Navigates forward in the platform's history.
    this._location.back();
  }
}
```

AsyncPipe

Die async-Pipe abonniert ein Observable oder Promise und gibt den neuesten Wert zurück, den

sie ausgegeben hat. Wenn ein neuer Wert ausgegeben wird, markiert die asynchrone Pipe die Komponente, die auf Änderungen geprüft werden soll. Wenn die Komponente zerstört wird, wird die asynchrone Pipe automatisch abbestellt, um mögliche Speicherlecks zu vermeiden.

```
@Component({
  selector: 'async-observable-pipe',
  template: '<div><code>observable|async</code>: Time: {{ time | async }}</div>'
})
export class AsyncObservablePipeComponent {
  time = new Observable<string>((observer: Subscriber<string>) => {
    setInterval(() => observer.next(new Date().toString()), 1000);
  });
}
```

Anzeige der aktuellen Version von `angular2`, die in Ihrem Projekt verwendet wird

Zur Anzeige der aktuellen Version können Sie **VERSION** aus dem `@angular/core`-Paket verwenden.

```
import { Component, VERSION } from '@angular/core';

@Component({
  selector: 'my-app',
  template: `<h1>Hello {{name}}</h1>
<h2>Current Version: {{ver}}</h2>
`,
})
export class AppComponent {
  name = 'Angular2';
  ver = VERSION.full;
}
```

Währungsrohr

Mit der Währungspipe können Sie mit Ihren Daten als reguläre Zahlen arbeiten, diese jedoch mit der Standardwährungsformatierung (Währungssymbol, Dezimalstellen usw.) in der Ansicht anzeigen.

```
@Component({
  selector: 'currency-pipe',
  template: `<div>
    <p>A: {{myMoney | currency:'USD':false}}</p>
    <p>B: {{yourMoney | currency:'USD':true:'4.2-2'}}</p>
  </div>`
})
export class CurrencyPipeComponent {
  myMoney: number = 100000.653;
  yourMoney: number = 5.3495;
}
```

Das Rohr hat drei optionale Parameter:

- **Währungscode** : Ermöglicht die Angabe des Währungscode nach ISO 4217.

- **symbolDisplay** : Boolescher **Wert, der** angibt, ob das Währungssymbol verwendet werden soll
- **digitInfo** : Hier können Sie angeben, wie die Dezimalstellen angezeigt werden sollen.

Weitere Informationen zur Currency Pipe:

<https://angular.io/docs/ts/latest/api/common/index/CurrencyPipe-pipe.html>

Häufig eingebaute Richtlinien und Dienste online lesen:

<https://riptutorial.com/de/angular2/topic/8252/haufig-eingebaute-richtlinien-und-dienste>

Kapitel 41: HTTP Interceptor

Bemerkungen

Was wir mit der `HttpServiceLayer`-Klasse tun, ist, die `Http`-Klasse von `angular` zu erweitern und unsere eigene Logik hinzuzufügen.

Wir fügen dann diese Klasse in die `Bootstrap`-Klasse der Anwendung ein und teilen dem Angreifer mit, dass wir die `Http`-Klasse importieren würden, in der Rückseite, um den `HttpServiceLayer` einzufügen.

Überall im Code können wir einfach importieren

```
import { Http } from '@angular/http';
```

Unsere Klasse wird jedoch für jeden Anruf verwendet.

Examples

Einfache Klasse Erweiterung der `Http`-Klasse von `Winkeln`

```
import { Http, Request, RequestOptionsArgs, Response, RequestOptions, ConnectionBackend, Headers } from '@angular/http';
import { Router } from '@angular/router';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/observable/empty';
import 'rxjs/add/observable/throw';
import 'rxjs/add/operator/catch';
import { ApplicationConfiguration } from '../application-configuration/application-configuration';

/**
 * This class extends the Http class from angular and adds automatically the server URL (if in development mode) and 2 headers by default:
 * Headers added: 'Content-Type' and 'X-AUTH-TOKEN'.
 * 'Content-Type' can be set in any other service, and if set, it will NOT be overwritten in this class any more.
 */
export class HttpServiceLayer extends Http {

  constructor(backend: ConnectionBackend, defaultOptions: RequestOptions, private _router: Router, private appConfig: ApplicationConfiguration) {
    super(backend, defaultOptions);
  }

  request(url: string | Request, options?: RequestOptionsArgs): Observable<Response> {
    this.getRequestOptionArgs(options);
    return this.intercept(super.request(this.appConfig.getServerAddress() + url, options));
  }

  /**
   * This method checks if there are any headers added and if not created the headers map and
```

```

ads 'Content-Type' and 'X-AUTH-TOKEN'
* 'Content-Type' is not overwritten if it is already available in the headers map
*/
getRequestOptionsArgs(options?: RequestOptionsArgs): RequestOptionsArgs {
  if (options == null) {
    options = new RequestOptions();
  }
  if (options.headers == null) {
    options.headers = new Headers();
  }

  if (!options.headers.get('Content-Type')) {
    options.headers.append('Content-Type', 'application/json');
  }

  if (this.appConfig.getAuthToken() != null) {
    options.headers.append('X-AUTH-TOKEN', this.appConfig.getAuthToken());
  }

  return options;
}

/**
* This method as the name suggests intercepts the request and checks if there are any errors.
* If an error is present it will be checked what error there is and if it is a general one
then it will be handled here, otherwise, will be
* thrown up in the service layers
*/
intercept(observable: Observable<Response>): Observable<Response> {

  // return observable;
  return observable.catch((err, source) => {
    if (err.status == 401) {
      this._router.navigate(['/login']);
      //return observable;
      return Observable.empty();
    } else {
      //return observable;
      return Observable.throw(err);
    }
  });
}
}

```

Verwendung unserer Klasse anstelle von Angulars Http

Nach der Erweiterung der Http-Klasse müssen wir winkelt anweisen, diese Klasse anstelle der Http-Klasse zu verwenden.

Dazu müssen wir in unserem Hauptmodul (oder je nach Bedarf nur ein bestimmtes Modul) in den Abschnitt Provider schreiben:

```

export function httpServiceFactory(xhrBackend: XHRBackend, requestOptions: RequestOptions,
router: Router, appConfig: ApplicationConfiguration) {
  return new HttpServiceLayer(xhrBackend, requestOptions, router, appConfig);
}

import { HttpModule, Http, Request, RequestOptionsArgs, Response, XHRBackend, RequestOptions,

```



```

ConnectionBackend, Headers } from '@angular/http';
import { Router } from '@angular/router';

@NgModule({
  declarations: [ ... ],
  imports: [ ... ],
  exports: [ ... ],
  providers: [
    ApplicationConfiguration,
    {
      provide: Http,
      useFactory: httpServiceFactory,
      deps: [XHRBackend, RequestOptions, Router, ApplicationConfiguration]
    }
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

Hinweis: ApplicationConfiguration ist nur ein Dienst, den ich für die Dauer der Anwendung verwende

Simple HttpClient AuthToken Interceptor (Angular 4.3+)

```

import { Injectable } from '@angular/core';
import { HttpEvent, HttpRequest, HttpInterceptor, HttpHandler } from '@angular/common/http';
import { UserService } from '../services/user.service';
import { Observable } from 'rxjs/Observable';

@Injectable()
export class AuthHeaderInterceptor implements HttpInterceptor {

  constructor(private userService: UserService) {
  }

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    if (this.userService.isAuthenticated()) {
      req = req.clone({
        setHeaders: {
          Authorization: `Bearer ${this.userService.token}`
        }
      });
    }
    return next.handle(req);
  }
}

```

Bereitstellen von Interceptor (some-module.module.ts)

```
{provide: HTTP_INTERCEPTORS, useClass: AuthHeaderInterceptor, multi: true},
```

HTTP Interceptor online lesen: <https://riptutorial.com/de/angular2/topic/1413/http-interceptor>

Kapitel 42: Installieren von Drittanbieter-Plugins mit angle-cli@1.0.0-beta.10

Bemerkungen

Es ist möglich, andere Bibliotheken nach diesem Ansatz zu installieren. Möglicherweise müssen jedoch Modultyp, Hauptdatei und Standarderweiterung angegeben werden.

```
'lodash': {  
  format: 'cjs',  
  defaultExtension: 'js',  
  main: 'index.js'  
}
```

```
'moment': {  
  main: 'moment.js'  
}
```

Examples

Hinzufügen einer Jquery-Bibliothek zum angle-cli-Projekt

1. Jquery über npm installieren:

```
npm install jquery --save
```

Installieren Sie Typisierungen für die Bibliothek:

Führen Sie folgende Schritte aus, um Typisierungen für eine Bibliothek hinzuzufügen:

```
typings install jquery --global --save
```

2. Fügen Sie der vorder-cli-build.js-Datei jevery zum vendorNpmFiles-Array hinzu:

Dies ist erforderlich, damit das Buildsystem die Datei aufnimmt. Nach dem Setup sollte die angle-cli-build.js so aussehen:

Durchsuchen Sie die `node_modules` und suchen Sie nach Dateien und Ordnern, die Sie dem Herstellerordner hinzufügen möchten.

```
var Angular2App = require('angular-cli/lib/broccoli/angular2-app');  
  
module.exports = function(defaults) {  
  return new Angular2App(defaults, {
```

```
vendorNpmFiles: [  
  // ...  
  'jquery/dist/*.js'  
  
  ]  
});  
};
```

3. Konfigurieren Sie SystemJS-Zuordnungen, um zu wissen, wo Sie nach JQuery suchen müssen:

Die SystemJS-Konfiguration befindet sich in system-config.ts. Nachdem die benutzerdefinierte Konfiguration abgeschlossen ist, sollte der zugehörige Abschnitt folgendermaßen aussehen:

```
/** Map relative paths to URLs. */  
const map: any = {  
  'jquery': 'vendor/jquery'  
};  
  
/** User packages configuration. */  
const packages: any = {  
  
  // no need to add anything here for jquery  
  
};
```

4. Fügen Sie in Ihrer src / index.html diese Zeile hinzu

```
<script src="vendor/jquery/dist/jquery.min.js" type="text/javascript"></script>
```

Ihre anderen Optionen sind:

```
<script src="vendor/jquery/dist/jquery.js" type="text/javascript"></script>
```

oder

```
<script src="/vendor/jquery/dist/jquery.slim.js" type="text/javascript"></script>
```

und

```
<script src="/vendor/jquery/dist/jquery.slim.min.js" type="text/javascript"></script>
```

5. Importieren und Verwenden der JQuery-Bibliothek in Ihren Projektquelldateien:

Importieren Sie die JQuery-Bibliothek in Ihre .ts-Quelldateien wie folgt:

```

declare var $:any;

@Component({
})
export class YourComponent {
  ngOnInit() {
    $(".button").click(function(){
      // now you can DO, what ever you want
    });
    console.log();
  }
}

```

Wenn Sie die Schritte richtig ausgeführt haben, sollten Sie jetzt die JQuery-Bibliothek in Ihrem Projekt verwenden. Genießen!

Fügen Sie eine Drittanbieter-Bibliothek hinzu, die keine Typisierung hat

Beachten Sie, dass dies nur für eckige Kli bis Version 1.0.0-beta.10 gilt!

Einige Bibliotheken oder Plugins haben möglicherweise keine Typisierung. Andernfalls kann TypeScript sie nicht überprüfen und verursacht daher Kompilierungsfehler. Diese Bibliotheken können weiterhin verwendet werden, unterscheiden sich jedoch von importierten Modulen.

1. Fügen Sie einen Skriptverweis für die Bibliothek auf Ihrer Seite hinzu (`index.html`).

```

<script src="//cdn.somewhe.re/lib.min.js" type="text/javascript"></script>
<script src="/local/path/to/lib.min.js" type="text/javascript"></script>

```

- Diese Skripte sollten eine globale (zB `THREE`, `mapbox`, `$` usw.) hinzufügen oder mit einer globalen `mapbox`

2. Verwenden Sie in der Komponente, die diese benötigt, eine `declare`, um eine Variable zu initialisieren, die dem von der Bibliothek verwendeten globalen Namen entspricht. Dadurch kann TypeScript wissen, dass es bereits initialisiert wurde. ¹

```

declare var <globalname>: any;

```

Einige Bibliotheken werden an das `window` angehängt, das erweitert werden müsste, um in der App zugänglich zu sein.

```

interface WindowIntercom extends Window { Intercom: any; }
declare var window: WindowIntercom;

```

3. Verwenden Sie bei Bedarf die `lib` in Ihren Komponenten.

```

@Component { ... }
export class AppComponent implements AfterViewInit {
  ...
  ngAfterViewInit() {
    var geometry = new THREE.BoxGeometry( 1, 1, 1 );
  }
}

```

```
    window.Intercom('boot', { ... }  
  }  
}
```

- ANMERKUNG: Einige Bibliotheken können mit dem DOM interagieren und sollten in der entsprechenden [Komponentenlebenszyklusmethode](#) verwendet werden.

Installieren von Drittanbieter-Plugins mit `angle-cli@1.0.0-beta.10` online lesen:

<https://riptutorial.com/de/angular2/topic/2328/installieren-von-drittanbieter-plugins-mit-angle-cli-1-0-0-beta-10>

Kapitel 43: Komponenten

Einführung

Winkelkomponenten sind Elemente, die aus einer Vorlage bestehen, die Ihre Anwendung rendert.

Examples

Eine einfache Komponente

Um eine Komponente zu erstellen, fügen wir `@Component` Dekorator in einer Klasse hinzu, wobei einige Parameter übergeben werden:

- `providers` : Ressourcen, die in den Komponentenkonstruktor eingefügt werden
- `selector` : Der Abfrageselektor, der das Element im HTML-Code findet und durch die Komponente ersetzt
- `styles` : Inline-Styles. ANMERKUNG: Verwenden Sie diesen Parameter NICHT mit Requirement, er funktioniert bei der Entwicklung, aber wenn Sie die Anwendung in der Produktion erstellen, gehen alle Ihre Stile verloren
- `styleUrls` : Array von `styleUrls` zu Style-Dateien
- `template` : Zeichenfolge, die Ihren HTML-Code enthält
- `templateUrl` : Pfad zu einer HTML-Datei

Es gibt andere Parameter, die Sie konfigurieren können, aber die aufgelisteten sind die, die Sie am häufigsten verwenden werden.

Ein einfaches Beispiel:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-required',
  styleUrls: ['required.component.scss'],
  // template: `This field is required.`,
  templateUrl: 'required.component.html',
})
export class RequiredComponent { }
```

Vorlagen & Stile

Vorlagen sind HTML-Dateien, die Logik enthalten können.

Sie können eine Vorlage auf zwei Arten angeben:

Vorlage als Dateipfad übergeben

```
@Component ({
  templateUrl: 'hero.component.html',
})
```

Vorlage als Inline-Code übergeben

```
@Component ({
  template: `<div>My template here</div>`,
})
```

Vorlagen können Stile enthalten. Die in `@Component` deklarierten `@Component` unterscheiden sich von Ihrer Anwendungsstildatei. `@Component` in der Komponente angewendeten Stile sind auf diesen Bereich beschränkt. Angenommen, Sie fügen Folgendes hinzu:

```
div { background: red; }
```

Alle `div`s im Inneren des Bauteils wird rot sein, aber wenn Sie andere Komponenten haben, andere `div`s in Ihrem HTML werden sie nicht verändert werden.

Der generierte Code sieht folgendermaßen aus:

```
<style>div[_ngcontent-c1] { background: red; }</style>
```

Sie können einer Komponente auf zwei Arten Stile hinzufügen:

Ein Array von Dateipfaden übergeben

```
@Component ({
  styleUrls: ['hero.component.css'],
})
```

Übergeben eines Arrays von Inline-Codes

```
styles: [ `div { background: lime; }` ]
```

Sie sollten `styles` mit `require` da dies beim Erstellen Ihrer Anwendung für die Produktion nicht funktioniert.

Eine Komponente testen

hero.component.html

```
<form (ngSubmit)="submit($event)" [formGroup]="form" novalidate>
  <input type="text" FormControlName="name" />
  <button type="submit">Show hero name</button>
</form>
```

hero.component.ts

```
import { FormControl, FormGroup, Validators } from '@angular/forms';

import { Component } from '@angular/core';

@Component({
  selector: 'app-hero',
  templateUrl: 'hero.component.html',
})
export class HeroComponent {
  public form = new FormGroup({
    name: new FormControl('', Validators.required),
  });

  submit(event) {
    console.log(event);
    console.log(this.form.controls.name.value);
  }
}
```

hero.component.spec.ts

```
import { ComponentFixture, TestBed, async } from '@angular/core/testing';

import { HeroComponent } from './hero.component';
import { ReactiveFormsModule } from '@angular/forms';

describe('HeroComponent', () => {
  let component: HeroComponent;
  let fixture: ComponentFixture<HeroComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [HeroComponent],
      imports: [ReactiveFormsModule],
    }).compileComponents();

    fixture = TestBed.createComponent(HeroComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  }));

  it('should be created', () => {
    expect(component).toBeTruthy();
  });

  it('should log hero name in the console when user submit form', async(() => {
    const heroName = 'Saitama';
    const element = <HTMLFormElement>fixture.debugElement.nativeElement.querySelector('form');

    spyOn(console, 'log').and.callThrough();

    component.form.controls['name'].setValue(heroName);

    element.querySelector('button').click();

    fixture.whenStable().then(() => {
      fixture.detectChanges();
    });
  }));
});
```



```

        expect(console.log).toHaveBeenCalledWith(heroName);
    });
});

it('should validate name field as required', () => {
    component.form.controls['name'].setValue('');
    expect(component.form.invalid).toBeTruthy();
});
});

```

Komponenten verschachteln

Komponenten werden in ihrem jeweiligen `selector`, sodass Sie diese zum Schachteln von Komponenten verwenden können.

Wenn Sie eine Komponente haben, die eine Nachricht anzeigt:

```

import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-required',
  template: `{{name}} is required.`
})
export class RequiredComponent {
  @Input()
  public name: String = '';
}

```

Sie können es in einer anderen Komponente verwenden, indem Sie `app-required` (Selector dieser Komponente) verwenden:

```

import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-sample',
  template: `
    <input type="text" name="heroName" />
    <app-required name="Hero Name"></app-required>
  `
})
export class RequiredComponent {
  @Input()
  public name: String = '';
}

```

Komponenten online lesen: <https://riptutorial.com/de/angular2/topic/10838/komponenten>

Kapitel 44: Komponenteninteraktionen

Syntax

- `<element [variableName]="value"></element> //Declaring input to child when using @Input() method.`
- `<element (childOutput)="parentFunction($event)"></element> //Declaring output from child when using @Output() method.`
- `@Output() pageNumberClicked = new EventEmitter(); //Used for sending output data from child component when using @Output() method.`
- `this.pageNumberClicked.emit(pageNum); //Used to trigger data output from child component. when using @Output() method.`
- `@ViewChild(ComponentClass) //Property decorator is required when using ViewChild.`

Parameter

Name	Wert
Seitenzahl	Wird verwendet, um der untergeordneten Komponente die Anzahl der Seiten anzugeben, die erstellt werden sollen.
pageNumberClicked	Name der Ausgabevariable in der untergeordneten Komponente.
pageChanged	Funktion bei übergeordneter Komponente, die auf Ausgabe untergeordneter Komponenten wartet.

Examples

Parent - Child-Interaktion mit den Eigenschaften @Input & @Output

Wir haben eine `DataListComponent`, die Daten zeigt, die wir aus einem Dienst abrufen. `DataListComponent` hat auch eine `PagerComponent` als untergeordnetes Element.

`PagerComponent` erstellt eine Seitennummernliste basierend auf der Gesamtzahl der von `DataListComponent` abgerufenen Seiten. `PagerComponent` informiert die `DataListComponent` auch, wenn der Benutzer über die Output-Eigenschaft auf eine Seitennummer klickt.

```
import { Component, NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { DataListService } from '../dataList.service';
import { PagerComponent } from '../pager.component';

@Component({
  selector: 'datalist',
  template: `
    <table>
    <tr *ngFor="let person of personsData">
      <td>{{person.name}}</td>
      <td>{{person.surname}}</td>
```

```

        </tr>
    </table>

    <pager [pageCount]="pageCount" (pageNumberClicked)="pageChanged($event)"></pager>
    `
})
export class DataListComponent {
    private personsData = null;
    private pageCount: number;

    constructor(private dataListService: DataListService) {
        var response = this.dataListService.getData(1); //Request first page from the service
        this.personsData = response.persons;
        this.pageCount = response.totalCount / 10; //We will show 10 records per page.
    }

    pageChanged(pageNumber: number){
        var response = this.dataListService.getData(pageNumber); //Request data from the
service with new page number
        this.personsData = response.persons;
    }
}

@NgModule({
    imports: [CommonModule],
    exports: [],
    declarations: [DataListComponent, PagerComponent],
    providers: [DataListService],
})
export class DataListModule { }

```

PagerComponent listet alle Seitennummern auf. Wir legen für jedes von ihnen ein Klickereignis fest, damit wir dem Elternteil die angeklickte Seitennummer mitteilen können.

```

import { Component, Input, Output, EventEmitter } from '@angular/core';

@Component({
    selector: 'pager',
    template: `
    <div id="pager-wrapper">
        <span *ngFor="#page of pageCount" (click)="pageClicked(page)">{{page}}</span>
    </div>
    `
})
export class PagerComponent {
    @Input() pageCount: number;
    @Output() pageNumberClicked = new EventEmitter();
    constructor() { }

    pageClicked(pageNum) {
        this.pageNumberClicked.emit(pageNum); //Send clicked page number as output
    }
}

```

Parent - Child-Interaktion mit ViewChild

ViewChild bietet eine Möglichkeit der Interaktion von Eltern zu Kindern. Wenn ViewChild verwendet wird, erfolgt keine Rückmeldung oder Ausgabe von untergeordneten Elementen.

Wir haben eine `DataListComponent`, die einige Informationen anzeigt. `DataListComponent` hat `PagerComponent` als untergeordnetes Element. Wenn der Benutzer eine Suche in `DataListComponent` durchführt, erhält er Daten von einem Dienst und fordert `PagerComponent` auf, das Seitenlayout basierend auf der neuen Anzahl von Seiten zu aktualisieren.

```
import { Component, NgModule, ViewChild } from '@angular/core';
import { CommonModule } from '@angular/common';
import { DataListService } from '../dataList.service';
import { PagerComponent } from '../pager.component';

@Component({
  selector: 'datalist',
  template: `<input type='text' [(ngModel)]="searchText" />
    <button (click)="getData()">Search</button>
    <table>
    <tr *ngFor="let person of personsData">
      <td>{{person.name}}</td>
      <td>{{person.surname}}</td>
    </tr>
    </table>
    <pager></pager>
  `
})
export class DataListComponent {
  private personsData = null;
  private searchText: string;

  @ViewChild(PagerComponent)
  private pagerComponent: PagerComponent;

  constructor(private dataListService: DataListService) {}

  getData() {
    var response = this.dataListService.getData(this.searchText);
    this.personsData = response.data;
    this.pagerComponent.setPaging(this.personsData / 10); //Show 10 records per page
  }
}

@NgModule({
  imports: [CommonModule],
  exports: [],
  declarations: [DataListComponent, PagerComponent],
  providers: [DataListService],
})
export class DataListModule { }
```

Auf diese Weise können Sie Funktionen aufrufen, die an untergeordneten Komponenten definiert sind.

Die untergeordnete Komponente ist erst verfügbar, wenn die übergeordnete Komponente gerendert wird. `AfterViewInit` Sie versuchen, vor dem `AfterViewInit` Lebenszyklus-Hook der Eltern auf das Kind zuzugreifen, wird eine Ausnahme ausgelöst.

Bidirektionale Eltern-Kind-Interaktion über einen Dienst

Dienst, der für die Kommunikation verwendet wird:

```
import { Injectable } from '@angular/core';
import { Subject } from 'rxjs/Subject';

@Injectable()
export class ComponentCommunicationService {

  private componentChangeSource = new Subject();
  private newDateCreationSource = new Subject<Date>();

  componentChanged$ = this.componentChangeSource.asObservable();
  dateCreated$ = this.newDateCreationSource.asObservable();

  refresh() {
    this.componentChangeSource.next();
  }

  broadcastDate(date: Date) {
    this.newDateCreationSource.next(date);
  }
}
```

Übergeordnete Komponente:

```
import { Component, Inject } from '@angular/core';
import { ComponentCommunicationService } from './component-refresh.service';

@Component({
  selector: 'parent',
  template: `
<button (click)="refreshSubscribed()">Refresh</button>
<h1>Last date from child received: {{lastDate}}</h1>
<child-component></child-component>
`
})
export class ParentComponent implements OnInit {

  lastDate: Date;
  constructor(private communicationService: ComponentCommunicationService) { }

  ngOnInit() {
    this.communicationService.dateCreated$.subscribe(newDate => {
      this.lastDate = newDate;
    });
  }

  refreshSubscribed() {
    this.communicationService.refresh();
  }
}
```

Kindkomponente:

```
import { Component, OnInit, Inject } from '@angular/core';
import { ComponentCommunicationService } from './component-refresh.service';

@Component({
```

```

    selector: 'child-component',
    template: `
    <h1>Last refresh from parent: {{lastRefreshed}}</h1>
    <button (click)="sendNewDate()">Send new date</button>
    `
  })
  export class ChildComponent implements OnInit {

    lastRefreshed: Date;
    constructor(private communicationService: ComponentCommunicationService) { }

    ngOnInit() {
      this.communicationService.componentChanged$.subscribe(event => {
        this.onRefresh();
      });
    }

    sendNewDate() {
      this.communicationService.broadcastDate(new Date());
    }

    onRefresh() {
      this.lastRefreshed = new Date();
    }
  }
}

```

AppModule:

```

@NgModule({
  declarations: [
    ParentComponent,
    ChildComponent
  ],
  providers: [ComponentCommunicationService],
  bootstrap: [AppComponent] // not included in the example
})
export class AppModule {}

```

Komponenteninteraktionen online lesen:

<https://riptutorial.com/de/angular2/topic/7400/komponenteninteraktionen>

Kapitel 45: Komponenteninteraktionen

Einführung

Teilen Sie Informationen zwischen verschiedenen Richtlinien und Komponenten.

Examples

Übergabe der Daten von einem Elternteil an ein Kind mit Eingabe

HeroChildComponent verfügt über zwei Eingabeeigenschaften, die normalerweise mit @Input-Dekorationen versehen sind.

```
import { Component, Input } from '@angular/core';
import { Hero } from './hero';
@Component({
  selector: 'hero-child',
  template: `
    <h3>{{hero.name}} says:</h3>
    <p>I, {{hero.name}}, am at your service, {{masterName}}.</p>
  `
})
export class HeroChildComponent {
  @Input() hero: Hero;
  @Input('master') masterName: string;
}
```

Änderungen der Eingabeeigenschaften mit einem Setter abfangen

Verwenden Sie einen Eingabeeigenschaften-Setter, um einen Wert vom übergeordneten Element abzufangen und darauf zu reagieren.

Der Setter der Namenseingangs-eigenschaft in der untergeordneten NameChildComponent schneidet den Leerraum aus einem Namen ab und ersetzt einen leeren Wert durch den Standardtext.

```
import { Component, Input } from '@angular/core';
@Component({
  selector: 'name-child',
  template: '<h3>{{name}}</h3>'
})
export class NameChildComponent {
  private _name = '';
  @Input()
  set name(name: string) {
    this._name = (name && name.trim()) || '<no name set>';
  }
  get name(): string { return this._name; }
}
```

Hier ist die NameParentComponent, die Namensvariationen zeigt, einschließlich eines Namens

mit allen Leerzeichen:

```
import { Component } from '@angular/core';
@Component({
  selector: 'name-parent',
  template: `
    <h2>Master controls {{names.length}} names</h2>
    <name-child *ngFor="let name of names" [name]="name"></name-child>
  `
})
export class NameParentComponent {
  // Displays 'Mr. IQ', '<no name set>', 'Bombasto'
  names = ['Mr. IQ', ' ', ' Bombasto '];
}
```

Elternteil hört auf Kindereignis

Die untergeordnete Komponente macht eine EventEmitter-Eigenschaft verfügbar, mit der Ereignisse ausgelöst werden, wenn etwas passiert. Das übergeordnete Element bindet an diese Ereignisseigenschaft und reagiert auf diese Ereignisse.

Die EventEmitter-Eigenschaft des Kindes ist eine Ausgabeeigenschaft, die normalerweise mit einer @Output-Verzierung versehen ist, wie in dieser VoterComponent zu sehen ist:

```
import { Component, EventEmitter, Input, Output } from '@angular/core';
@Component({
  selector: 'my-voter',
  template: `
    <h4>{{name}}</h4>
    <button (click)="vote(true)" [disabled]="voted">Agree</button>
    <button (click)="vote(false)" [disabled]="voted">Disagree</button>
  `
})
export class VoterComponent {
  @Input() name: string;
  @Output() onVoted = new EventEmitter<boolean>();
  voted = false;
  vote(agree: boolean) {
    this.onVoted.emit(agree);
    this.voted = true;
  }
}
```

Durch Klicken auf eine Schaltfläche wird eine wahr oder falsch ausgegeben (die boolesche Nutzlast).

Die übergeordnete VoteTakerComponent bindet einen Ereignishandler (onVoted), der auf die untergeordneten Ereignisnutzdaten (\$ event) reagiert und einen Leistungsindikator aktualisiert.

```
import { Component } from '@angular/core';
@Component({
  selector: 'vote-taker',
  template: `
    <h2>Should mankind colonize the Universe?</h2>
    <h3>Agree: {{agree}}, Disagree: {{disagree}}</h3>
  `
})
export class VoteTakerComponent {
  @Input() name: string;
  @Output() onVoted = new EventEmitter<boolean>();
  voted = false;
  vote(agree: boolean) {
    this.onVoted.emit(agree);
    this.voted = true;
  }
}
```



```

    <my-voter *ngFor="let voter of voters"
      [name]="voter"
      (onVoted)="onVoted($event)">
    </my-voter>
  `
  `
})
export class VoteTakerComponent {
  agreed = 0;
  disagreed = 0;
  voters = ['Mr. IQ', 'Ms. Universe', 'Bombasto'];
  onVoted(agreed: boolean) {
    agreed ? this.agreed++ : this.disagreed++;
  }
}

```

Das Elternteil interagiert mit dem Kind über die lokale Variable

Eine übergeordnete Komponente kann keine Datenbindung verwenden, um untergeordnete Eigenschaften zu lesen oder untergeordnete Methoden aufzurufen. Beides können Sie tun, indem Sie eine Vorlagenreferenzvariable für das untergeordnete Element erstellen und dann auf diese Variable innerhalb der übergeordneten Vorlage verweisen, wie im folgenden Beispiel gezeigt.

Wir haben eine untergeordnete CountdownTimerComponent, die wiederholt auf null herunter zählt und eine Rakete startet. Es verfügt über Start- und Stopp-Methoden, die die Uhr steuern, und es wird eine Countdown-Statusmeldung in einer eigenen Vorlage angezeigt.

```

import { Component, OnDestroy, OnInit } from '@angular/core';
@Component({
  selector: 'countdown-timer',
  template: '<p>{{message}}</p>'
})
export class CountdownTimerComponent implements OnInit, OnDestroy {
  intervalId = 0;
  message = '';
  seconds = 11;
  clearTimer() { clearInterval(this.intervalId); }
  ngOnInit() { this.start(); }
  ngOnDestroy() { this.clearTimer(); }
  start() { this.countDown(); }
  stop() {
    this.clearTimer();
    this.message = `Holding at T-${this.seconds} seconds`;
  }
  private countDown() {
    this.clearTimer();
    this.intervalId = window.setInterval(() => {
      this.seconds -= 1;
      if (this.seconds === 0) {
        this.message = 'Blast off!';
      } else {
        if (this.seconds < 0) { this.seconds = 10; } // reset
        this.message = `T-${this.seconds} seconds and counting`;
      }
    }, 1000);
  }
}

```

Sehen wir uns die CountdownLocalVarParentComponent an, die die Timer-Komponente hostet.

```
import { Component } from '@angular/core';
import { CountdownTimerComponent } from './countdown-timer.component';
@Component({
  selector: 'countdown-parent-lv',
  template: `
    <h3>Countdown to Liftoff (via local variable)</h3>
    <button (click)="timer.start()">Start</button>
    <button (click)="timer.stop()">Stop</button>
    <div class="seconds">{{timer.seconds}}</div>
    <countdown-timer #timer></countdown-timer>
  `,
  styleUrls: ['demo.css']
})
export class CountdownLocalVarParentComponent { }
```

Die übergeordnete Komponente kann weder an die Start- und Stop-Methoden des Kindes noch an die Sekunden-Eigenschaft der Daten binden.

Wir können eine lokale Variable (#timer) auf das Tag () setzen, das die untergeordnete Komponente darstellt. Dies gibt uns einen Verweis auf die untergeordnete Komponente selbst und die Möglichkeit, innerhalb der übergeordneten Vorlage auf ihre Eigenschaften oder Methoden zuzugreifen.

In diesem Beispiel verbinden wir übergeordnete Schaltflächen mit dem Start und Stopp des Kindes und verwenden die Interpolation, um die Sekunden-Eigenschaft des Kindes anzuzeigen.

Hier sehen wir, wie Eltern und Kind zusammenarbeiten.

Übergeordnete ruft eine ViewChild auf

Der Ansatz der lokalen Variablen ist einfach und leicht. Dies ist jedoch begrenzt, da die Parent-Child-Verkabelung vollständig innerhalb der Parent-Vorlage erfolgen muss. Die übergeordnete Komponente selbst hat keinen Zugriff auf das untergeordnete Element.

Die lokale Variablentechnik kann nicht verwendet werden, wenn eine Instanz der übergeordneten Komponentenkategorie untergeordnete Komponentenwerte lesen oder schreiben oder untergeordnete Komponentenmethoden aufrufen muss.

Wenn die übergeordnete Komponentenkategorie diese Art von Zugriff erfordert, injizieren wir die untergeordnete Komponente als ViewChild in die übergeordnete Komponente.

Wir werden diese Technik anhand des gleichen Countdown-Timers veranschaulichen. Wir werden sein Aussehen oder Verhalten nicht ändern. Die untergeordnete CountdownTimerComponent ist ebenfalls dieselbe.

Wir wechseln ausschließlich zu Demonstrationszwecken von der lokalen Variablen zur ViewChild-Technik. Hier ist das übergeordnete CountdownViewChildParentComponent:

```
import { AfterViewInit, ViewChild } from '@angular/core';
import { Component } from '@angular/core';
```

```

import { CountdownTimerComponent } from './countdown-timer.component';
@Component({
  selector: 'countdown-parent-vc',
  template: `
<h3>Countdown to Liftoff (via ViewChild)</h3>
<button (click)="start()">Start</button>
<button (click)="stop()">Stop</button>
<div class="seconds">{{ seconds() }}</div>
<countdown-timer></countdown-timer>
`,
  styleUrls: ['demo.css']
})
export class CountdownViewChildParentComponent implements AfterViewInit {
  @ViewChild(CountdownTimerComponent)
  private timerComponent: CountdownTimerComponent;
  seconds() { return 0; }
  ngAfterViewInit() {
    // Redefine `seconds()` to get from the `CountdownTimerComponent.seconds` ...
    // but wait a tick first to avoid one-time devMode
    // unidirectional-data-flow-violation error
    setTimeout(() => this.seconds = () => this.timerComponent.seconds, 0);
  }
  start() { this.timerComponent.start(); }
  stop() { this.timerComponent.stop(); }
}

```

Es dauert etwas mehr Arbeit, um die untergeordnete Ansicht in die übergeordnete Komponentenklasse zu bringen.

Wir importieren Verweise auf den ViewChild-Dekorator und den Lebenszyklus-Hook AfterViewInit.

Wir injizieren die untergeordnete CountdownTimerComponent-Eigenschaft über die @ViewChild-Eigenschaftendekoration in die private timerComponent-Eigenschaft.

Die lokale Variable #timer wurde aus den Komponenten-Metadaten entfernt. Stattdessen binden wir die Schaltflächen an die Start- und Stop-Methoden der übergeordneten Komponente und präsentieren die tickenden Sekunden in einer Interpolation um die Sekunden-Methode der übergeordneten Komponente.

Diese Methoden greifen direkt auf die eingespritzte Zeitgeberkomponente zu.

Der ngAfterViewInit-Lebenszyklus-Hook ist eine wichtige Falte. Die Timerkomponente ist erst verfügbar, nachdem Angular die übergeordnete Ansicht angezeigt hat. Wir zeigen also zunächst 0 Sekunden an.

Dann ruft Angular den Lebenszyklus-Hook ngAfterViewInit auf. Zu diesem Zeitpunkt ist es zu spät, um die Anzeige der Countdown-Sekunden in der übergeordneten Ansicht zu aktualisieren. Die unidirektionale Datenflussregel von Angular verhindert, dass wir die übergeordneten Ansichten im selben Zyklus aktualisieren. Wir müssen eine Runde warten, bevor wir die Sekunden anzeigen können.

Wir verwenden setTimeout, um einen Tick zu warten, und überarbeiten die Sekunden-Methode, sodass zukünftige Werte aus der Timerkomponente übernommen werden.

Eltern und Kinder kommunizieren über einen Dienst

Eine übergeordnete Komponente und ihre untergeordneten Elemente teilen sich einen Dienst, dessen Schnittstelle die bidirektionale Kommunikation innerhalb der Familie ermöglicht.

Der Umfang der Dienstinstanz ist die übergeordnete Komponente und ihre untergeordneten Elemente. Komponenten außerhalb dieser Komponenten-Unterstruktur haben keinen Zugriff auf den Dienst oder deren Kommunikation.

Dieser `MissionService` verbindet die `MissionControlComponent` mit mehreren `AstronautComponent`-untergeordneten Objekten.

```
import { Injectable } from '@angular/core';
import { Subject } from 'rxjs/Subject';
@Injectable()
export class MissionService {
  // Observable string sources
  private missionAnnouncedSource = new Subject<string>();
  private missionConfirmedSource = new Subject<string>();
  // Observable string streams
  missionAnnounced$ = this.missionAnnouncedSource.asObservable();
  missionConfirmed$ = this.missionConfirmedSource.asObservable();
  // Service message commands
  announceMission(mission: string) {
    this.missionAnnouncedSource.next(mission);
  }
  confirmMission(astronaut: string) {
    this.missionConfirmedSource.next(astronaut);
  }
}
```

Die `MissionControlComponent` stellt die Instanz des Dienstes bereit, die sie mit ihren untergeordneten Objekten (über das Metadaten-Array des Providers) gemeinsam nutzt, und fügt diese Instanz über ihren Konstruktor in sich ein:

```
import { Component } from '@angular/core';
import { MissionService } from './mission.service';
@Component({
  selector: 'mission-control',
  template: `
    <h2>Mission Control</h2>
    <button (click)="announce()">Announce mission</button>
    <my-astronaut *ngFor="let astronaut of astronauts"
      [astronaut]="astronaut">
    </my-astronaut>
    <h3>History</h3>
    <ul>
      <li *ngFor="let event of history">{{event}}</li>
    </ul>
  `,
  providers: [MissionService]
})
export class MissionControlComponent {
  astronauts = ['Lovell', 'Swigert', 'Haise'];
  history: string[] = [];
  missions = ['Fly to the moon!'];
```

```

        'Fly to mars!',
        'Fly to Vegas!'];
nextMission = 0;
constructor(private missionService: MissionService) {
    missionService.missionConfirmed$.subscribe(
        astronaut => {
            this.history.push(`${astronaut} confirmed the mission`);
        });
}
announce() {
    let mission = this.missions[this.nextMission++];
    this.missionService.announceMission(mission);
    this.history.push(`Mission "${mission}" announced`);
    if (this.nextMission >= this.missions.length) { this.nextMission = 0; }
}
}

```

Die AstronautComponent fügt den Dienst auch in den Konstruktor ein. Jede AstronautComponent ist ein untergeordnetes Element der MissionControlComponent und empfängt daher die Dienstinstanz ihres übergeordneten Objekts:

```

import { Component, Input, OnDestroy } from '@angular/core';
import { MissionService } from './mission.service';
import { Subscription } from 'rxjs/Subscription';
@Component({
    selector: 'my-astronaut',
    template: `
        <p>
            {{astronaut}}: <strong>{{mission}}</strong>
            <button
                (click)="confirm()"
                [disabled]="!announced || confirmed">
                Confirm
            </button>
        </p>
    `
})
export class AstronautComponent implements OnDestroy {
    @Input() astronaut: string;
    mission = '<no mission announced>';
    confirmed = false;
    announced = false;
    subscription: Subscription;
    constructor(private missionService: MissionService) {
        this.subscription = missionService.missionAnnounced$.subscribe(
            mission => {
                this.mission = mission;
                this.announced = true;
                this.confirmed = false;
            });
    }
    confirm() {
        this.confirmed = true;
        this.missionService.confirmMission(this.astronaut);
    }
    ngOnDestroy() {
        // prevent memory leak when component destroyed
        this.subscription.unsubscribe();
    }
}

```

```
}
```

Beachten Sie, dass wir das Abonnement erfassen und das Abonnement abbestellen, wenn die AstronautComponent zerstört wird. Dies ist ein Schritt zum Schutz vor Speicherlecks. In dieser App gibt es kein tatsächliches Risiko, da die Lebensdauer einer AstronautComponent der Lebensdauer der App selbst entspricht. Das wäre in einer komplexeren Anwendung nicht immer der Fall.

Wir fügen diesen Schutz nicht zur MissionControlComponent hinzu, da er als übergeordnetes Element die Lebensdauer des MissionService steuert. Das Verlaufsprotokoll zeigt, dass Nachrichten in beide Richtungen zwischen der übergeordneten MissionControlComponent-Komponente und der untergeordneten AstronautComponent-Komponente gesendet werden, was durch den Dienst erleichtert wird:

Komponenteninteraktionen online lesen:

<https://riptutorial.com/de/angular2/topic/9454/komponenteninteraktionen>

Kapitel 46: Konfigurieren der ASP.net Core-Anwendung für die Arbeit mit Angular 2 und TypeScript

Einführung

SZENARIO: ASP.NET Core-Hintergrund Angular 2-Front-End-Angular 2-Komponenten mit Asp.net-Core-Controllern

So kann Angular 2 über die Asp.Net Core App implementiert werden. Wir können MVC-Controller auch von Angular 2-Komponenten aufrufen, wobei das MVC-Ergebnis View unterstützt, das Angular 2 unterstützt.

Examples

Asp.Net Core + Angular2 + Gulp

Startup.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using CoreAngular000.Data;
using CoreAngular000.Models;
using CoreAngular000.Services;
using Microsoft.Extensions.FileProviders;
using System.IO;

namespace CoreAngular000
{
    public class Startup
    {
        public Startup(IHostingEnvironment env)
        {
            var builder = new ConfigurationBuilder()
                .SetBasePath(env.ContentRootPath)
                .AddJsonFile("appsettings.json", optional: false, reloadOnChange:
true)
                .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional:
true);

            if (env.IsDevelopment())
```

```

    {

        builder.AddUserSecrets<Startup>();
    }

    builder.AddEnvironmentVariables();
    Configuration = builder.Build();
}

public IConfigurationRoot Configuration { get; }

public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>()
        .AddDefaultTokenProviders();

    services.AddMvc();

    // Add application services.
    services.AddTransient<IEmailSender, AuthMessageSender>();
    services.AddTransient<ISmsSender, AuthMessageSender>();
}

public void Configure(IApplicationBuilder app, IHostingEnvironment env,
ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole(Configuration.GetSection("Logging"));
    loggerFactory.AddDebug();

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseDatabaseErrorPage();
        app.UseBrowserLink();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseDefaultFiles();
    app.UseStaticFiles();
    app.UseStaticFiles(new StaticFileOptions
    {
        FileProvider = new
PhysicalFileProvider(Path.Combine(env.ContentRootPath, "node_modules"),
        RequestPath = "/node_modules"
    });

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}

```



```
    }  
  }  
}
```

tsConfig.json

```
{  
  "compilerOptions": {  
    "diagnostics": true,  
    "emitDecoratorMetadata": true,  
    "experimentalDecorators": true,  
    "lib": [ "es2015", "dom" ],  
    "listFiles": true,  
    "module": "commonjs",  
    "moduleResolution": "node",  
    "noImplicitAny": true,  
    "outDir": "wwwroot",  
    "removeComments": false,  
    "rootDir": "wwwroot",  
    "sourceMap": true,  
    "suppressImplicitAnyIndexErrors": true,  
    "target": "es5"  
  },  
  "exclude": [  
    "node_modules",  
    "wwwroot/lib/"  
  ]  
}
```

Package.json

```
{  
  "name": "angular dependencies and web dev package",  
  "version": "1.0.0",  
  "description": "Angular 2 MVC. Samuel Maicas Template",  
  "scripts": {},  
  "dependencies": {  
    "@angular/common": "~2.4.0",  
    "@angular/compiler": "~2.4.0",  
    "@angular/core": "~2.4.0",  
    "@angular/forms": "~2.4.0",  
    "@angular/http": "~2.4.0",  
    "@angular/platform-browser": "~2.4.0",  
    "@angular/platform-browser-dynamic": "~2.4.0",  
    "@angular/router": "~3.4.0",  
    "angular-in-memory-web-api": "~0.2.4",  
    "systemjs": "0.19.40",  
    "core-js": "^2.4.1",  
    "rxjs": "5.0.1",  
    "zone.js": "^0.7.4"  
  },  
  "devDependencies": {  
    "del": "^2.2.2",  
    "gulp": "^3.9.1",  
    "gulp-concat": "^2.6.1",  
    "gulp-cssmin": "^0.1.7",  
    "gulp-htmlmin": "^3.0.0",  
    "gulp-uglify": "^2.1.2",  
    "merge-stream": "^1.0.1",  
  }  
}
```

```
"tslint": "^3.15.1",
"typescript": "~2.0.10"
},
"repository": {}
}
```

bundleconfig.json

```
[
  {
    "outputFileName": "wwwroot/css/site.min.css",
    "inputFiles": [
      "wwwroot/css/site.css"
    ]
  },
  {
    "outputFileName": "wwwroot/js/site.min.js",
    "inputFiles": [
      "wwwroot/js/site.js"
    ],
    "minify": {
      "enabled": true,
      "renameLocals": true
    },
    "sourceMap": false
  }
]
```

Konvertieren Sie bundleconfig.json in gulpfile (RightClick bundleconfig.json im Solution Explorer, Bundler & Minifier> In Gulp konvertieren)

Ansichten / Startseite / Index.cshtml

```
@{
    ViewData["Title"] = "Home Page";
}
<div>{{ nombre }}</div>
```

Für den Ordner "wwwroot" verwenden Sie <https://github.com/angular/quickstart> seed. Sie benötigen: **index.html** **main.ts**, **systemjs-angle-loader.js**, **systemjs.config.js**, **tsconfig.json** und den **App-Ordner**

wwwroot / Index.html

```
<html>
<head>
  <title>SMTemplate Angular2 & ASP.NET Core</title>
  <base href="/">
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">

  <script src="node_modules/core-js/client/shim.min.js"></script>

  <script src="node_modules/zone.js/dist/zone.js"></script>
  <script src="node_modules/systemjs/dist/system.src.js"></script>
```

```

<script src="systemjs.config.js"></script>
<script>
  System.import('main.js').catch(function(err){ console.error(err); });
</script>
</head>

<body>
  <my-app>Loading AppComponent here ...</my-app>
</body>
</html>

```

Sie können Controller von templateUrl als Controller aufrufen. `wwwroot / app / app.component.ts`

```

import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  templateUrl: '/home/index',
})
export class AppComponent { nombre = 'Samuel Maicas'; }

```

[Seed] Asp.Net Core + Angular2 + Gulp auf Visual Studio 2017

1. Samen herunterladen
2. Führen Sie Dotnet Restore aus
3. Führen Sie npm install aus

Immer. Genießen.

<https://github.com/SamML/CoreAngular000>

MVC <-> Winkel 2

Gewusst wie: ANGULAR 2 HTML / JS COMPONENT VON ASP.NET Core CONTROLLER ANRUFEN:

Wir rufen den HTML-Code auf und geben stattdessen View () zurück.

```
return File("~/html/About.html", "text/html");
```

Und laden Sie die Winkelkomponente in die HTML-Datei. Hier können wir entscheiden, ob wir mit demselben oder einem anderen Modul arbeiten wollen. Kommt auf die Situation an.

`wwwroot / html / About.html`

```

<!DOCTYPE html>
<html>
  <head>
    <title>About Page</title>
    <base href="/">
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

```

```

    <link href="../../../css/site.min.css" rel="stylesheet" type="text/css"/>

    <script src="../../../node_modules/core-js/client/shim.min.js"></script>

    <script src="../../../node_modules/zone.js/dist/zone.js"></script>
    <script src="../../../node_modules/systemjs/dist/system.src.js"></script>

    <script src="../../../systemjs.config.js"></script>
    <script>
      System.import('../main.js').catch(function(err){ console.error(err); });
    </script>
  </head>

  <body>
    <aboutpage>Loading AppComponent here ...</aboutpage>
  </body>
</html>

```

(*) Dieser Seed muss bereits die gesamte Ressourcenliste laden

Gewusst wie: RUFEN Sie den ASP.NET Core Controller an, um eine MVC-Ansicht mit Angular2-Unterstützung anzuzeigen:

```

import { Component } from '@angular/core';

@Component({
  selector: 'aboutpage',
  templateUrl: '/home/about',
})
export class AboutComponent {

}

```

Konfigurieren der ASP.net Core-Anwendung für die Arbeit mit Angular 2 und TypeScript online lesen: <https://riptutorial.com/de/angular2/topic/9543/konfigurieren-der-asp-net-core-anwendung-fur-die-arbeit-mit-angular-2-und-typescript>

Kapitel 47: Lebenszyklus-Haken

Bemerkungen

Verfügbarkeit von Veranstaltungen

`AfterViewInit` und `AfterViewChecked` sind nur **in Komponenten** und **nicht in Direktiven** verfügbar.

Events bestellen

- `OnChanges` (mehrmals)
- `OnInit` (einmalig)
- `DoCheck` (mehrmals)
- `AfterContentInit` (einmalig)
- `AfterContentChecked` (mehrmals)
- `AfterViewInit` (einmalig) (nur Komponente)
- `AfterViewChecked` (mehrmals) (nur Komponente)
- `OnDestroy` (einmalig)

Lesen Sie weiter

- [Winkeldokumentation - Lebenszyklushaken](#)

Examples

OnInit

Wird ausgelöst, wenn Komponenten- oder Anweisungseigenschaften initialisiert wurden.

(Vor denen der Kinderrichtlinien)

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'so-oninit-component',
  templateUrl: 'oninit-component.html',
  styleUrls: ['oninit-component.'],
})
class OnInitComponent implements OnInit {

  ngOnInit(): void {
    console.log('Component is ready !');
  }
}
```

OnDestroy

Wird ausgelöst, wenn die Komponenten- oder Direktiveninstanz zerstört wird.

```
import { Component, OnDestroy } from '@angular/core';

@Component({
  selector: 'so-ondestroy-component',
  templateUrl: 'ondestroy-component.html',
  styleUrls: ['ondestroy-component.']
})
class OnDestroyComponent implements OnDestroy {

  ngOnDestroy(): void {
    console.log('Component was destroyed !');
  }
}
```

OnChange

Wird ausgelöst, wenn eine oder mehrere der Komponenten- oder Anweisungseigenschaften geändert wurden

```
import { Component, OnChanges, Input } from '@angular/core';

@Component({
  selector: 'so-onchanges-component',
  templateUrl: 'onchanges-component.html',
  styleUrls: ['onchanges-component.']
})
class OnChangesComponent implements OnChanges {
  @Input() name: string;
  message: string;

  ngOnChanges(changes: SimpleChanges): void {
    console.log(changes);
  }
}
```

Bei Änderung wird das Ereignis protokolliert

```
name: {
  currentValue: 'new name value',
  previousValue: 'old name value'
},
message: {
  currentValue: 'new message value',
  previousValue: 'old message value'
}
```

AfterContentInit

Feuer, nachdem die Initialisierung des Inhalts der Komponente oder der Direktive abgeschlossen ist.

(Gleich nach OnInit)

```
import { Component, AfterContentInit } from '@angular/core';

@Component({
  selector: 'so-aftercontentinit-component',
  templateUrl: 'aftercontentinit-component.html',
  styleUrls: ['aftercontentinit-component.'],
})
class AfterContentInitComponent implements AfterContentInit {

  ngAfterContentInit(): void {
    console.log('Component content have been loaded!');
  }
}
```

AfterContentChecked

Feuer, nachdem die Ansicht vollständig initialisiert wurde.

(Nur für Komponenten verfügbar)

```
import { Component, AfterContentChecked } from '@angular/core';

@Component({
  selector: 'so-aftercontentchecked-component',
  templateUrl: 'aftercontentchecked-component.html',
  styleUrls: ['aftercontentchecked-component.'],
})
class AfterContentCheckedComponent implements AfterContentChecked {

  ngAfterContentChecked(): void {
    console.log('Component content have been checked!');
  }
}
```

AfterViewInit

Wird ausgelöst, nachdem die Komponentenansicht und eine ihrer untergeordneten Ansichten initialisiert wurde. Dies ist ein nützlicher Lebenszyklus-Hook für Plugins außerhalb des Angular 2-Ökosystems. Mit dieser Methode können Sie beispielsweise eine jQuery-Datumsauswahl basierend auf der von Angular 2 gerenderten Markierung initialisieren.

```
import { Component, AfterViewInit } from '@angular/core';

@Component({
  selector: 'so-afterviewinit-component',
  templateUrl: 'afterviewinit-component.html',
  styleUrls: ['afterviewinit-component.'],
})
class AfterViewInitComponent implements AfterViewInit {

  ngAfterViewInit(): void {
    console.log('This event fire after the content init have been loaded!');
  }
}
```

```
}
```

AfterViewChecked

Feuer, nachdem die Überprüfung der Ansicht der Komponente abgeschlossen ist.

(Nur für Komponenten verfügbar)

```
import { Component, AfterViewChecked } from '@angular/core';

@Component({
  selector: 'so-afterviewchecked-component',
  templateUrl: 'afterviewchecked-component.html',
  styleUrls: ['afterviewchecked-component.'],
})
class AfterViewCheckedComponent implements AfterViewChecked {

  ngAfterViewChecked(): void {
    console.log('This event fire after the content have been checked!');
  }
}
```

DoCheck

Ermöglicht das Abhören von Änderungen nur für angegebene Eigenschaften

```
import { Component, DoCheck, Input } from '@angular/core';

@Component({
  selector: 'so-docheck-component',
  templateUrl: 'docheck-component.html',
  styleUrls: ['docheck-component.'],
})
class DoCheckComponent implements DoCheck {
  @Input() elements: string[];
  differ: any;
  ngDoCheck(): void {
    // get value for elements property
    const changes = this.differ.diff(this.elements);

    if (changes) {
      changes.forEachAddedItem(res => console.log('Added', r.item));
      changes.forEachRemovedItem(r => console.log('Removed', r.item));
    }
  }
}
```

Lebenszyklus-Haken online lesen: <https://riptutorial.com/de/angular2/topic/2935/lebenszyklus-haken>

Kapitel 48: Module

Einführung

Winkelmodule sind Container für verschiedene Teile Ihrer App.

Sie können verschachtelte Module haben, Ihr `app.module` verschachtelt bereits andere Module wie `BrowserModule` und Sie können `RouterModule` usw. hinzufügen.

Examples

Ein einfaches Modul

Ein Modul ist eine Klasse mit dem `@NgModule` Dekorator. Um ein Modul zu erstellen, fügen wir `@NgModule` einigen Parametern hinzu:

- `bootstrap` : Die Komponente, die das Stammverzeichnis Ihrer Anwendung sein wird. Diese Konfiguration ist nur auf Ihrem Root-Modul vorhanden
- `declarations` : Ressourcen, die das Modul deklariert. Wenn Sie eine neue Komponente hinzufügen, müssen Sie die Deklarationen aktualisieren (`ng generate component` führt dies automatisch durch).
- `exports` : Der Modulexport wird in anderen Modulen verwendet
- `imports` : Ressourcen, die das Modul aus anderen Modulen verwendet (nur Modulklassen werden akzeptiert)
- `providers` : Ressourcen, die in eine Komponente injiziert werden können

Ein einfaches Beispiel:

```
import { AppComponent } from './app.component';
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

@NgModule({
  bootstrap: [AppComponent]
  declarations: [AppComponent],
  exports: [],
  imports: [BrowserModule],
  providers: [],
})
export class AppModule { }
```

Verschachteln von Modulen

Die Module können mit Hilfe der geschachtelt werden `imports` Parameter von `@NgModule` Dekorateur.

Wir können in unserer Anwendung ein `core.module` erstellen, das generische Dinge wie eine `ReservePipe` (eine Pipe, die eine Zeichenfolge `ReservePipe`) enthält, und diese in diesem Modul

bündeln:

```
import { CommonModule } from '@angular/common';
import { NgModule } from '@angular/core';
import { ReversePipe } from '../reverse.pipe';

@NgModule({
  imports: [
    CommonModule
  ],
  exports: [ReversePipe], // export things to be imported in another module
  declarations: [ReversePipe],
})
export class CoreModule { }
```

Dann in der `app.module` :

```
import { CoreModule } from 'app/core/core.module';

@NgModule({
  declarations: [...], // ReversePipe is available without declaring here
                        // because CoreModule exports it
  imports: [
    CoreModule,        // import things from CoreModule
    ...
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Module online lesen: <https://riptutorial.com/de/angular2/topic/10840/module>

Kapitel 49: NgModel testen

Einführung

Ist ein Beispiel dafür, wie Sie eine Komponente in Angular2 mit einem ngModel testen können.

Examples

Grundtest

```
import { BrowserModule } from '@angular/platform-browser';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/http';

import { Component, DebugElement } from '@angular/core';
import { dispatchEvent } from "@angular/platform-browser/testing/browser_util";
import { TestBed, ComponentFixture } from '@angular/core/testing';
import { By } from "@angular/platform-browser";

import { MyComponentModule } from 'ng2-my-component';
import { MyComponent } from './my-component';

describe('MyComponent:', () => {

  const template = `
    <div>
      <my-component type="text" [(ngModel)]="value" name="TestName" size="9" min="3" max="8"
placeholder="testPlaceholder" disabled=false required=false></my-component>
    </div>
  `;

  let fixture:any;
  let element:any;
  let context:any;

  beforeEach(() => {

    TestBed.configureTestingModule({
      declarations: [InlineEditorComponent],
      imports: [
        FormsModule,
        InlineEditorModule]
    });
    fixture = TestBed.overrideComponent(InlineEditorComponent, {
      set: {
        selector:"inline-editor-test",
        template: template
      }})
      .createComponent(InlineEditorComponent);
    context = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should change value of the component', () => {
```

```
let input = fixture.nativeElement.querySelector("input");
input.value = "Username";
dispatchEvent(input, 'input');
fixture.detectChanges();

fixture.whenStable().then(() => {
  //this button dispatch event for save the text in component.value
  fixture.nativeElement.querySelectorAll('button')[0].click();
  expect(context.value).toBe("Username");
});
});
});
```

NgModel testen online lesen: <https://riptutorial.com/de/angular2/topic/8693/ngmodel-testen>

Kapitel 50: ngrx

Einführung

Ngrx ist eine leistungsstarke Bibliothek, die Sie mit **Angular2 verwenden können** . Die Idee dahinter ist , zwei Konzepte zu verschmelzen , die gut zusammen spielen eine **reaktive App** mit einem vorhersagbaren **Zustand Behältern** haben: - [Redux] [1] - [RxJs] [2] Die wichtigsten Vorteile: - Gemeinsame Nutzung von Daten in der App zwischen Komponenten wird einfacher - Das Testen Ihrer App-Kernlogik besteht darin, reine Funktionen zu testen, ohne auf Angular2 angewiesen zu sein (sehr einfach!) [1]: <http://redux.js.org> [2]: <http://interactivex.io/rxjs>

Examples

Vollständiges Beispiel: Anmelden / Abmelden eines Benutzers

Voraussetzungen

Dieses Thema behandelt **nicht** Redux und / oder Ngrx:

- Sie müssen mit Redux vertraut sein
- Verstehe zumindest die Grundlagen von RxJs und Observable Pattern

Zuerst definieren wir ein Beispiel von Anfang an und spielen mit etwas Code:

Als Entwickler möchte ich:

1. `IUser` eine `IUser` Schnittstelle, die die Eigenschaften eines `User`
2. Deklarieren Sie die Aktionen, die wir später verwenden werden, um den `User` im `Store`
3. Definieren Sie den Anfangszustand des `UserReducer`
4. Erstellen Sie den Reduzierer `UserReducer`
5. Importieren Sie unseren `UserReducer` in unser Hauptmodul, um den `Store` zu erstellen
6. Verwenden Sie Daten aus dem `Store` , um Informationen in unserer Ansicht anzuzeigen

Spoiler-Alarm : Wenn Sie die Demo sofort ausprobieren oder den Code lesen möchten, bevor wir überhaupt anfangen, ist hier ein Plunkr (Embed- [View](#) oder [Run-View](#)).

1) Definieren Sie die `IUser` Schnittstelle

Ich mag es, meine Schnittstellen in zwei Teile aufzuteilen:

- Die Eigenschaften erhalten wir von einem Server
- Die Eigenschaften, die wir nur für die Benutzeroberfläche definieren (sollte sich beispielsweise eine Schaltfläche drehen)

Und hier ist das Interface, `IUser` ich verwenden werde:

user.interface.ts

```
export interface IUser {
  // from server
  username: string;
  email: string;

  // for UI
  isConnected: boolean;
  isConnecting: boolean;
};
```

2) Deklarieren Sie die Aktionen zur Manipulation des `User`

Jetzt müssen wir darüber nachdenken, mit welchen Aktionen unsere **Reduzierer** umgehen sollen. Sagen wir hier:

user.actions.ts

```
export const UserActions = {
  // when the user clicks on login button, before we launch the HTTP request
  // this will allow us to disable the login button during the request
  USR_IS_CONNECTING: 'USR_IS_CONNECTING',
  // this allows us to save the username and email of the user
  // we assume those data were fetched in the previous request
  USR_IS_CONNECTED: 'USR_IS_CONNECTED',

  // same pattern for disconnecting the user
  USR_IS_DISCONNECTING: 'USR_IS_DISCONNECTING',
  USR_IS_DISCONNECTED: 'USR_IS_DISCONNECTED'
};
```

Bevor wir diese Aktionen verwenden, lassen Sie mich erklären, warum wir einen Dienst benötigen, um **einige** dieser Aktionen für uns auszuführen:

Angenommen, wir möchten einen Benutzer verbinden. Wir werden also auf einen Login-Button klicken und Folgendes wird passieren:

- Klicken Sie auf die Schaltfläche
- Die Komponente fängt das Ereignis ab und ruft `userService.login`
- `userService.login` Methode `dispatch` ein Ereignis unseren Speicher Eigenschaft zu **aktualisieren**: `user.isConnected`
- Ein HTTP-Aufruf wird ausgelöst (wir verwenden ein `setTimeout` in der Demo, um das **async-Verhalten** zu simulieren)
- Sobald der `HTTP` Aufruf abgeschlossen ist, werden wir eine weitere Aktion auslösen, um unseren Shop zu warnen, dass ein Benutzer angemeldet ist

user.service.ts

```
@Injectable()
export class UserService {
  constructor(public store$: Store<AppState>) { }

  login(username: string) {
    // first, dispatch an action saying that the user's trying to connect
    // so we can lock the button until the HTTP request finish
    this.store$.dispatch({ type: UserActions.USR_IS_CONNECTING });

    // simulate some delay like we would have with an HTTP request
    // by using a timeout
    setTimeout(() => {
      // some email (or data) that you'd have get as HTTP response
      let email = `${username}@email.com`;

      this.store$.dispatch({ type: UserActions.USR_IS_CONNECTED, payload: { username, email }
    });
  }, 2000);
}

logout() {
  // first, dispatch an action saying that the user's trying to connect
  // so we can lock the button until the HTTP request finish
  this.store$.dispatch({ type: UserActions.USR_IS_DISCONNECTING });

  // simulate some delay like we would have with an HTTP request
  // by using a timeout
  setTimeout(() => {
    this.store$.dispatch({ type: UserActions.USR_IS_DISCONNECTED });
  }, 2000);
}
}
```

3) Definieren Sie den Anfangszustand des

UserReducer

user.state.ts

```
export const UserFactory: IUser = () => {
  return {
    // from server
    username: null,
    email: null,

    // for UI
    isConnected: false,
    isConnecting: false,
    isDisconnecting: false
  };
};
```

4) Erstellen Sie den Reduzierer `UserReducer`

Ein Reduzierer benötigt 2 Argumente:

- Der aktuelle Zustand
- Eine `Action` vom Typ `Action<{type: string, payload: any}>`

Zur Erinnerung: Ein Reduzierer muss irgendwann initialisiert werden

Da wir in Teil 3 den Standardzustand unseres Reduzierers definiert haben, können wir ihn so verwenden:

`user.reducer.ts`

```
export const UserReducer: ActionReducer<IUser> = (user: IUser, action: Action) => {
  if (user === null) {
    return userFactory();
  }

  // ...
}
```

Hoffentlich gibt es eine einfachere Möglichkeit, dies zu schreiben, indem Sie unsere `factory` Funktion verwenden, um ein Objekt zurückzugeben, und innerhalb des Reduzierers einen (ES6) [Standardparameterwert verwenden](#) :

```
export const UserReducer: ActionReducer<IUser> = (user: IUser = UserFactory(), action: Action)
=> {
  // ...
}
```

Dann müssen wir alle Aktionen in unserem Reduzierer zu handhaben : **TIP:** Verwenden Sie [ES6](#) `Object.assign` Funktion unseres Staates unveränderlich zu halten

```
export const UserReducer: ActionReducer<IUser> = (user: IUser = UserFactory(), action: Action)
=> {
  switch (action.type) {
    case UserActions.USR_IS_CONNECTING:
      return Object.assign({}, user, { isConnecting: true });

    case UserActions.USR_IS_CONNECTED:
      return Object.assign({}, user, { isConnecting: false, isConnected: true, username:
action.payload.username });

    case UserActions.USR_IS_DISCONNECTING:
      return Object.assign({}, user, { isDisconnecting: true });

    case UserActions.USR_IS_DISCONNECTED:
      return Object.assign({}, user, { isDisconnecting: false, isConnected: false });
  }
}
```



```
    default:
      return user;
  }
};
```

5) Importieren Sie unseren `UserReducer` in unser Hauptmodul, um den `Store` zu erstellen

app.module.ts

```
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    // angular modules
    // ...

    // declare your store by providing your reducers
    // (every reducer should return a default state)
    StoreModule.provideStore({
      user: UserReducer,
      // of course, you can put as many reducers here as you want
      // ...
    }),

    // other modules to import
    // ...
  ]
});
```

6) Verwenden Sie Daten aus dem `Store`, um Informationen in unserer Ansicht anzuzeigen

Alles ist jetzt auf der Logikseite fertig und wir müssen nur das zeigen, was wir wollen, in zwei Komponenten:

- `UserComponent` : **[Dumme Komponente]** Wir übergeben das Benutzerobjekt einfach mit der `@Input` Eigenschaft und der `async` Pipe aus dem Speicher. Auf diese Weise erhält die Komponente den Benutzer erst, wenn er verfügbar ist (und der `user` ist vom Typ `IUser` und nicht vom Typ `Observable<IUser>`!).
- `LoginComponent` **[Smart-Komponente]** Wir injizieren den `Store` direkt in diese Komponente und arbeiten nur für den `user` als `Observable`.

user.component.ts

```
@Component({
  selector: 'user',
```

```

styles: [
  '.table { max-width: 250px; }',
  '.truthy { color: green; font-weight: bold; }',
  '.falsy { color: red; }'
],
template: `
  <h2>User information :</h2>

  <table class="table">
    <tr>
      <th>Property</th>
      <th>Value</th>
    </tr>

    <tr>
      <td>username</td>
      <td [class.truthy]="user.username" [class.falsy]="!user.username">
        {{ user.username ? user.username : 'null' }}
      </td>
    </tr>

    <tr>
      <td>email</td>
      <td [class.truthy]="user.email" [class.falsy]="!user.email">
        {{ user.email ? user.email : 'null' }}
      </td>
    </tr>

    <tr>
      <td>isConnecting</td>
      <td [class.truthy]="user.isConnecting" [class.falsy]="!user.isConnecting">
        {{ user.isConnecting }}
      </td>
    </tr>

    <tr>
      <td>isConnected</td>
      <td [class.truthy]="user.isConnected" [class.falsy]="!user.isConnected">
        {{ user.isConnected }}
      </td>
    </tr>

    <tr>
      <td>isDisconnecting</td>
      <td [class.truthy]="user.isDisconnecting" [class.falsy]="!user.isDisconnecting">
        {{ user.isDisconnecting }}
      </td>
    </tr>
  </table>
`
,
export class UserComponent {
  @Input() user;

  constructor() { }
}

```

login.component.ts

```
@Component({
```

```

selector: 'login',
template: `
  <form
    *ngIf="!(user | async).isConnected"
    #loginForm="ngForm"
    (ngSubmit)="login(loginForm.value.username) "
  >
    <input
      type="text"
      name="username"
      placeholder="Username"
      [disabled]="(user | async).isConnecting"
      ngModel
    >

    <button
      type="submit"
      [disabled]="(user | async).isConnecting || (user | async).isConnected"
    >Log me in</button>
  </form>

  <button
    *ngIf="(user | async).isConnected"
    (click)="logout () "
    [disabled]="(user | async).isDisconnecting"
  >Log me out</button>
`
})
export class LoginComponent {
  public user: Observable<IUser>;

  constructor(public store$: Store<AppState>, private userService: UserService) {
    this.user = store$.select('user');
  }

  login(username: string) {
    this.userService.login(username);
  }

  logout () {
    this.userService.logout ();
  }
}

```

Wie `Ngrx` eine Zusammenführung ist `Redux` und `RxJs` Konzepte, kann es sehr schwierig sein, die In eine outs am Anfang zu verstehen. Dies ist jedoch ein leistungsfähiges Muster, mit dem Sie, wie wir in diesem Beispiel gesehen haben, eine *reaktive App haben können* und Ihre Daten problemlos freigeben können. Vergiss nicht, dass es einen [Plunkr](#) gibt und du kannst ihn für deine eigenen Tests zusammenfassen!

Ich hoffe, es war hilfreich, obwohl das Thema ziemlich lang ist, Prost!

[ngrx online lesen: https://riptutorial.com/de/angular2/topic/8086/ngrx](https://riptutorial.com/de/angular2/topic/8086/ngrx)

Kapitel 51: Optimieren des Renderns mit ChangeDetectionStrategy

Examples

Standard vs OnPush

Betrachten Sie die folgende Komponente mit einer Eingabe `myInput` und einem internen Wert namens `someInternalValue`. Beide werden in der Vorlage einer Komponente verwendet.

```
import {Component, Input} from '@angular/core';

@Component ({
  template:`
  <div>
    <p>{{myInput}}</p>
    <p>{{someInternalValue}}</p>
  </div>
  `
})
class MyComponent {
  @Input () myInput: any;

  someInternalValue: any;

  // ...
}
```

Standardmäßig wird die `changeDetection:` im Komponentendekorator auf `ChangeDetectionStrategy.Default` . im Beispiel implizit. In dieser Situation lösen alle Änderungen an den Werten in der Vorlage ein erneutes Rendern von `MyComponent` . Mit anderen Worten, wenn ich `myInput` oder `someInternalValue` **Winkel 2, wird Energie** `someInternalValue` und die Komponente **erneut** `someInternalValue` .

Nehmen wir jedoch an, dass wir nur dann neu rendern möchten, wenn sich die Eingaben ändern. Berücksichtigen Sie die folgende Komponente mit `changeDetection: ChangeDetectionStrategy.OnPush`

```
import {Component, ChangeDetectionStrategy, Input} from '@angular/core';

@Component ({
  changeDetection: ChangeDetectionStrategy.OnPush
  template:`
  <div>
    <p>{{myInput}}</p>
    <p>{{someInternalValue}}</p>
  </div>
  `
})
class MyComponent {
  @Input () myInput: any;
```

```
someInternalValue: any;  
  
// ...  
}
```

Durch Festlegen von `changeDetection: auf ChangeDetectionStrategy.OnPush` wird `MyComponent` nur dann neu `MyComponent`, wenn sich die Eingaben ändern. In diesem Fall muss `myInput` einen neuen Wert von seinem übergeordneten `myInput` erhalten, um ein erneutes Rendern auszulösen.

Optimieren des Renderns mit `ChangeDetectionStrategy` online lesen:

<https://riptutorial.com/de/angular2/topic/2644/optimieren-des-renderns-mit-changedetectionstrategy>

Kapitel 52: OrderBy Pipe

Einführung

Wie schreibe ich Order Pipe und benutze es.

Examples

Das Rohr

Die Pipe-Implementierung

```
import {Pipe, PipeTransform} from '@angular/core';

@Pipe({
  name: 'orderBy',
  pure: false
})
export class OrderBy implements PipeTransform {

  value:string[] =[];

  static _orderByComparator(a:any, b:any):number{

    if(a === null || typeof a === 'undefined') a = 0;
    if(b === null || typeof b === 'undefined') b = 0;

    if((isNaN(parseFloat(a)) || !isFinite(a)) || (isNaN(parseFloat(b)) || !isFinite(b))){
      //Isn't a number so lowercase the string to properly compare
      if(a.toLowerCase() < b.toLowerCase()) return -1;
      if(a.toLowerCase() > b.toLowerCase()) return 1;
    }else{
      //Parse strings as numbers to compare properly
      if(parseFloat(a) < parseFloat(b)) return -1;
      if(parseFloat(a) > parseFloat(b)) return 1;
    }

    return 0; //equal each other
  }

  transform(input:any, config:string = '+'): any{

    //make a copy of the input's reference
    this.value = [...input];
    let value = this.value;

    if(!Array.isArray(value)) return value;

    if(!Array.isArray(config) || (Array.isArray(config) && config.length === 1)){
      let propertyToCheck:string = !Array.isArray(config) ? config : config[0];
      let desc = propertyToCheck.substr(0, 1) === '-';

      //Basic array
      if(!propertyToCheck || propertyToCheck === '-' || propertyToCheck === '+'){
```

```

    return !desc ? value.sort() : value.sort().reverse();
  } else {
    let property:string = propertyToCheck.substr(0, 1) === '+' ||
propertyToCheck.substr(0, 1) === '-'
      ? propertyToCheck.substr(1)
      : propertyToCheck;

    return value.sort(function(a:any,b:any){
      return !desc
        ? OrderBy._orderByComparator(a[property], b[property])
        : -OrderBy._orderByComparator(a[property], b[property]);
    });
  }
} else {
  //Loop over property of the array in order and sort
  return value.sort(function(a:any,b:any){
    for(let i:number = 0; i < config.length; i++){
      let desc = config[i].substr(0, 1) === '-';
      let property = config[i].substr(0, 1) === '+' || config[i].substr(0, 1) === '-'
        ? config[i].substr(1)
        : config[i];

      let comparison = !desc
        ? OrderBy._orderByComparator(a[property], b[property])
        : -OrderBy._orderByComparator(a[property], b[property]);

      //Don't return 0 yet in case of needing to sort by next property
      if(comparison !== 0) return comparison;
    }

    return 0; //equal each other
  });
}
}
}
}

```

Verwendung der Pipe in der HTML - Reihenfolge aufsteigend nach Vornamen

```

<table>
  <thead>
    <tr>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Age</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let user of users | orderBy : ['firstName']">
      <td>{{user.firstName}}</td>
      <td>{{user.lastName}}</td>
      <td>{{user.age}}</td>
    </tr>
  </tbody>
</table>

```

So verwenden Sie die Pipe in der HTML - Reihenfolge, absteigend nach Vorname

```
<table>
  <thead>
    <tr>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Age</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let user of users | orderBy : ['-firstName']">
      <td>{{user.firstName}}</td>
      <td>{{user.lastName}}</td>
      <td>{{user.age}}</td>
    </tr>
  </tbody>
</table>
```

OrderBy Pipe online lesen: <https://riptutorial.com/de/angular2/topic/8969/orderby-pipe>

Kapitel 53: Pfeifen

Einführung

Das Rohr | Das Zeichen wird zum Anwenden von Pipes in Angular 2 verwendet. Pipes sind Filtern in AngularJS sehr ähnlich, da beide dazu beitragen, die Daten in ein bestimmtes Format umzuwandeln.

Parameter

Funktion / Parameter	Erläuterung
@Pipe ({Name, rein})	Metadaten für Pipe müssen unmittelbar vor der Pipe-Klasse liegen
name: <i>string</i>	was Sie in der Vorlage verwenden werden
rein: <i>boolean</i>	Wenn der Standardwert "true" ist, markieren Sie dies als "false", damit Ihre Pipe öfter neu bewertet wird
transform (value, args []?)	Die Funktion, die zum Umwandeln der Werte in der Vorlage aufgerufen wird
Wert: <i>beliebig</i>	den Wert, den Sie transformieren möchten
args: <i>any []</i>	die Argumente, die Sie möglicherweise in Ihre Transformation aufnehmen müssen. Markieren Sie optionale Argumente mit dem ? Operator wie so transform (Wert, Arg1, Arg2?)

Bemerkungen

Dieses Thema behandelt [Angular2 Pipes](#) , einen Mechanismus zum Umwandeln und Formatieren von Daten innerhalb von HTML-Vorlagen in einer Angular2-Anwendung.

Examples

Verkettung von Rohren

Rohre können angekettet sein.

```
<p>Today is {{ today | date:'fullDate' | uppercase}}.</p>
```

Kundenspezifische Rohre

my.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({name: 'myPipe'})
export class MyPipe implements PipeTransform {

  transform(value:any, args?: any):string {
    let transformedValue = value; // implement your transformation logic here
    return transformedValue;
  }

}
```

meine.komponente.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-component',
  template: `{{ value | myPipe }}`
})
export class MyComponent {

  public value:any;

}
```

mein.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { MyComponent } from './my.component';
import { MyPipe } from './my.pipe';

@NgModule({
  imports: [
    BrowserModule,
  ],
  declarations: [
    MyComponent,
    MyPipe
  ],
})
export class MyModule { }
```

Eingebaute Pfeifen

Angular2 wird mit ein paar eingebauten Rohren geliefert:

Rohr	Verwendungszweck	Beispiel
DatePipe	date	{{ dateObj date }} // output is 'Jun 15, 2015'
UpperCasePipe	uppercase	{{ value uppercase }} // output is 'SOMETEXT'
LowerCasePipe	lowercase	{{ value lowercase }} // output is 'sometext'
CurrencyPipe	currency	{{ 31.00 currency:'USD':true }} // output is '\$31'
PercentPipe	percent	{{ 0.03 percent }} //output is %3

Da sind andere. [Hier finden Sie](#) ihre Dokumentation.

Beispiel

hotel-reservation.component.ts

```
import { Component } from '@angular/core';

@Component({
  moduleId: module.id,
  selector: 'hotel-reservation',
  templateUrl: './hotel-reservation.template.html'
})
export class HotelReservationComponent {
  public fName: string = 'Joe';
  public lName: string = 'SCHMO';
  public reservationMade: string = '2016-06-22T07:18-08:00'
  public reservationFor: string = '2025-11-14';
  public cost: number = 99.99;
}
```

hotel-reservation.template.html

```
<div>
  <h1>Welcome back {{fName | uppercase}} {{lName | lowercase}}</h1>
  <p>
    On {reservationMade | date} at {reservationMade | date:'shortTime'} you
    reserved room 205 for {reservationDate | date} for a total cost of
    {cost | currency}.
  </p>
</div>
```

Ausgabe

```
Welcome back JOE schmo
On Jun 26, 2016 at 7:18 you reserved room 205 for Nov 14, 2025 for a total cost of
$99.99.
```

Debuggen mit JsonPipe

Das JsonPipe kann zum Debuggen des Zustands eines beliebigen internen verwendet werden.

Code

```
@Component({
  selector: 'json-example',
  template: `<div>
    <p>Without JSON pipe:</p>
    <pre>{{object}}</pre>
    <p>With JSON pipe:</p>
    <pre>{{object | json}}</pre>
  </div>`
})
export class JsonPipeExample {
  object: Object = {foo: 'bar', baz: 'qux', nested: {xyz: 3, numbers: [1, 2, 3, 4, 5]}};
}
```

Ausgabe

```
Without JSON Pipe:
object
With JSON pipe:
{object:{foo: 'bar', baz: 'qux', nested: {xyz: 3, numbers: [1, 2, 3, 4, 5]}}
```

Weltweit verfügbares Custom Pipe

Um eine benutzerdefinierte Pipe für die gesamte Anwendung verfügbar zu machen, erweitern Sie PLATFORM_PIPES während des Anwendungs-Bootstraps.

```
import { bootstrap } from '@angular/platform-browser-dynamic';
import { provide, PLATFORM_PIPES } from '@angular/core';

import { AppComponent } from './app.component';
import { MyPipe } from './my.pipe'; // your custom pipe

bootstrap(AppComponent, [
  provide(PLATFORM_PIPES, {
    useValue: [
      MyPipe
    ],
    multi: true
  })
]);
```

Tutorial hier: <https://scotch.io/tutorials/create-a-globally-available-custom-pipe-in-angular-2>

Benutzerdefiniertes Rohr erstellen

app / pipes.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';
```

```

@Pipe({name: 'truthy'})
export class Truthy implements PipeTransform {
  transform(value: any, truthy: string, falsey: string): any {
    if (typeof value === 'boolean'){return value ? truthy : falsey;}
    else return value
  }
}

```

app / meine-Komponente.Komponente.ts

```

import { Truthy } from './pipes.pipe';

@Component({
  selector: 'my-component',
  template: `
    <p>{{value | truthy:'enabled':'disabled'}}</p>
  `,
  pipes: [Truthy]
})
export class MyComponent{ }

```

Async-Werte mit Async-Pipe auspacken

```

import { Component } from '@angular/core';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/observable/of';

@Component({
  selector: 'async-stuff',
  template: `
    <h1>Hello, {{ name | async }}</h1>
    Your Friends are:
    <ul>
      <li *ngFor="let friend of friends | async">
        {{friend}}
      </li>
    </ul>
  `,
})
class AsyncStuffComponent {
  name = Promise.resolve('Misko');
  friends = Observable.of(['Igor']);
}

```

Wird:

```

<h1>Hello, Misko</h1>
Your Friends are:
<ul>
  <li>
    Igor
  </li>
</ul>

```

Bestehendes Rohr verlängern

```

import { Pipe, PipeTransform } from '@angular/core';
import { DatePipe } from '@angular/common'

@Pipe({name: 'ifDate'})
export class IfDate implements PipeTransform {
  private datePipe: DatePipe = new DatePipe();

  transform(value: any, pattern?:string) : any {
    if (typeof value === 'number') {return value}
    try {
      return this.datePipe.transform(value, pattern)
    } catch(err) {
      return value
    }
  }
}

```

Stateful Pipes

Angular 2 bietet zwei verschiedene Arten von Rohren an - zustandslos und zustandsbehaftet. Pipes sind standardmäßig zustandslos. Stateful-Pipes können jedoch implementiert werden, indem die `pure` -Eigenschaft auf `false` . Wie Sie im Parameterabschnitt sehen können, können Sie einen `name` angeben und angeben, ob die Pipe rein sein soll oder nicht, dh stateful oder zustandslos. Während Daten durch eine zustandslose Pipe (eine reine Funktion) fließen, **die sich** an nichts erinnert, können Daten von Stateful-Pipe verwaltet und gespeichert werden. Ein gutes Beispiel für eine Stateful-Pipe ist die `AsyncPipe` , die von Angular 2 bereitgestellt wird.

Wichtig

Beachten Sie, dass die meisten Rohre in die Kategorie zustandslose Rohre fallen sollten. Dies ist aus Leistungsgründen wichtig, da Angular zustandslose Rohre für den Änderungsdetektor optimieren kann. Verwenden Sie daher Stateful-Pipes vorsichtig. Im Allgemeinen bietet die Optimierung von Rohren in Angular 2 eine wesentliche Verbesserung der Leistung gegenüber Filtern in Angular 1.x. In Winkel 1 musste der Digest-Zyklus immer alle Filter erneut ausführen, auch wenn sich die Daten überhaupt nicht geändert haben. In Winkel 2 kann der Änderungsdetektor nach der Berechnung eines Pipe-Werts diese Pipe nicht erneut ausführen, wenn sich die Eingabe nicht ändert.

Implementierung einer Stateful Pipe

```

import {Pipe, PipeTransform, OnDestroy} from '@angular/core';

@Pipe({
  name: 'countdown',
  pure: false
})
export class CountdownPipe implements PipeTransform, OnDestroy {
  private interval: any;
  private remainingTime: number;

  transform(value: number, interval: number = 1000): number {
    if (!parseInt(value, 10)) {
      return null;
    }
  }
}

```

```

}

if (typeof this.remainingTime !== 'number') {
  this.remainingTime = parseInt(value, 10);
}

if (!this.interval) {
  this.interval = setInterval(() => {
    this.remainingTime--;

    if (this.remainingTime <= 0) {
      this.remainingTime = 0;
      clearInterval(this.interval);
      delete this.interval;
    }
  }, interval);
}

return this.remainingTime;
}

ngOnDestroy(): void {
  if (this.interval) {
    clearInterval(this.interval);
  }
}
}

```

Sie können die Pipe dann wie gewohnt verwenden:

```

{{ 1000 | countdown:50 }}
{{ 300 | countdown }}

```

Es ist wichtig, dass Ihre Pipe auch die `OnDestroy` Schnittstelle implementiert, damit Sie aufräumen können, sobald Ihre Pipe zerstört ist. In dem obigen Beispiel muss das Intervall gelöscht werden, um Speicherverluste zu vermeiden.

Dynamisches Rohr

Anwendungsfall: Eine Tabellensicht besteht aus verschiedenen Spalten mit unterschiedlichen Datenformaten, die mit unterschiedlichen Pipes umgewandelt werden müssen.

table.component.ts

```

...
import { DYNAMIC_PIPES } from '../pipes/dynamic.pipe.ts';

@Component({
  ...
  pipes: [DYNAMIC_PIPES]
})
export class TableComponent {
  ...

  // pipes to be used for each column
  table.pipes = [ null, null, null, 'humanizeDate', 'statusFromBoolean' ],

```

```

table.header = [ 'id', 'title', 'url', 'created', 'status' ],
table.rows = [
  [ 1, 'Home', 'home', '2016-08-27T17:48:32', true ],
  [ 2, 'About Us', 'about', '2016-08-28T08:42:09', true ],
  [ 4, 'Contact Us', 'contact', '2016-08-28T13:28:18', false ],
  ...
]
...
}

```

dynamische.rohr.ts

```

import {
  Pipe,
  PipeTransform
} from '@angular/core';
// Library used to humanize a date in this example
import * as moment from 'moment';

@Pipe({name: 'dynamic'})
export class DynamicPipe implements PipeTransform {

  transform(value:string, modifier:string) {
    if (!modifier) return value;
    // Evaluate pipe string
    return eval('this.' + modifier + '(\'' + value + '\')')
  }

  // Returns 'enabled' or 'disabled' based on input value
  statusFromBoolean(value:string):string {
    switch (value) {
      case 'true':
      case '1':
        return 'enabled';
      default:
        return 'disabled';
    }
  }

  // Returns a human friendly time format e.g: '14 minutes ago', 'yesterday'
  humanizeDate(value:string):string {
    // Humanize if date difference is within a week from now else returns 'December 20,
    2016' format
    if (moment().diff(moment(value), 'days') < 8) return moment(value).fromNow();
    return moment(value).format('MMMM Do YYYY');
  }
}

export const DYNAMIC_PIPES = [DynamicPipe];

```

table.component.html

```

<table>
  <thead>
    <td *ngFor="let head of data.header">{{ head }}</td>
  </thead>
  <tr *ngFor="let row of table.rows; let i = index">
    <td *ngFor="let column of row">{{ column | dynamic:table.pipes[i] }}</td>
  </tr>
</table>

```



```
</tr>
</table>
```

Ergebnis

ID	Page Title	Page URL	Created	Status
1	Home	home	4 minutes ago	Enabled
2	About Us	about	Yesterday	Enabled
4	Contact Us	contact	Yesterday	Disabled

Eine Pfeife testen

Gegeben eine Pipe, die eine Zeichenkette umkehrt

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({ name: 'reverse' })
export class ReversePipe implements PipeTransform {
  transform(value: string): string {
    return value.split('').reverse().join('');
  }
}
```

Es kann getestet werden, die Spec-Datei auf diese Weise zu konfigurieren

```
import { TestBed, inject } from '@angular/core/testing';

import { ReversePipe } from './reverse.pipe';

describe('ReversePipe', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      providers: [ReversePipe],
    });
  });

  it('should be created', inject([ReversePipe], (reversePipe: ReversePipe) => {
    expect(reversePipe).toBeTruthy();
  }));

  it('should reverse a string', inject([ReversePipe], (reversePipe: ReversePipe) => {
    expect(reversePipe.transform('abc')).toEqual('cba');
  }));
});
```

Pfeifen online lesen: <https://riptutorial.com/de/angular2/topic/1165/pfeifen>

Kapitel 54: Richtlinien

Syntax

- `<input [value]="value">` - Bindet den `name` Attributwert-Klassenmitglieds.
- `<div [attr.data-note]="note">` - Bindet das Attribut `data-note` an die variable `note` .
- `<p green></p>` - Benutzerdefinierte Anweisung

Bemerkungen

Die Hauptinformationsquelle zu Angular-2-Richtlinien ist die offizielle Dokumentation <https://angular.io/docs/ts/latest/guide/attribute-directives.html>

Examples

Attribut-Direktive

```
<div [class.active]="isActive"></div>

<span [style.color]='red'></span>

<p [attr.data-note]='This is value for data-note attribute'>A lot of text here</p>
```

Komponente ist eine Direktive mit Vorlage

```
import { Component } from '@angular/core';
@Component({
  selector: 'my-app',
  template: `
    <h1>Angular 2 App</h1>
    <p>Component is directive with template</p>
  `
})
export class AppComponent {
}
```

Strukturelle Richtlinien

```
<div *ngFor="let item of items">{{ item.description }}</div>

<span *ngIf="isVisible"></span>
```

Zollrichtlinie

```
import {Directive, ElementRef, Renderer} from '@angular/core';

@Directive({
  selector: '[green]',
})

class GreenDirective {
  constructor(private _elementRef: ElementRef,
              private _renderer: Renderer) {
    _renderer.setStyle(_elementRef.nativeElement, 'color', 'green');
  }
}
```

Verwendungszweck:

```
<p green>A lot of green text here</p>
```

* ngFor

form1.component.ts:

```
import { Component } from '@angular/core';

// Defines example component and associated template
@Component({
  selector: 'example',
  template: `
    <div *ngFor="let f of fruit"> {{f}} </div>
    <select required>
      <option *ngFor="let f of fruit" [value]="f"> {{f}} </option>
    </select>
  `
})

// Create a class for all functions, objects, and variables
export class ExampleComponent {
  // Array of fruit to be iterated by *ngFor
  fruit = ['Apples', 'Oranges', 'Bananas', 'Limes', 'Lemons'];
}
```

Ausgabe:

```
<div>Apples</div>
<div>Oranges</div>
<div>Bananas</div>
<div>Limes</div>
<div>Lemons</div>
<select required>
  <option value="Apples">Apples</option>
  <option value="Oranges">Oranges</option>
  <option value="Bananas">Bananas</option>
  <option value="Limes">Limes</option>
  <option value="Lemons">Lemons</option>
</select>
```

In seiner einfachsten Form besteht `*ngFor` aus zwei Teilen: `let variableName of object/array`

Im Falle von `fruit = ['Apples', 'Oranges', 'Bananas', 'Limes', 'Lemons'];`.

Äpfel, Orangen usw. sind die Werte in der `fruit` der Reihe.

`[value]="f"` ist gleich der aktuellen `fruit (f)`, die `*ngFor` hat.

Im Gegensatz zu AngularJS hat Angular2 die Verwendung von `ng-options` für `<select>` und `ng-repeat` für alle anderen allgemeinen Wiederholungen nicht fortgesetzt.

`*ngFor` ist sehr ähnlich zu `ng-repeat` mit leicht `*ngFor` Syntax.

Verweise:

Angular2 | [Daten anzeigen](#)

Angular2 | [ngFor](#)

Angular2 | [Formen](#)

Direkt in die Zwischenablage kopieren

In diesem Beispiel erstellen wir eine Anweisung, um einen Text in die Zwischenablage zu kopieren, indem Sie auf ein Element klicken

copy-text.directive.ts

```
import {
  Directive,
  Input,
  HostListener
} from "@angular/core";

@Directive({
  selector: '[text-copy]'
})
export class TextCopyDirective {

  // Parse attribute value into a 'text' variable
  @Input('text-copy') text:string;

  constructor() {
  }

  // The HostListener will listen to click events and run the below function, the
  // HostListener supports other standard events such as mouseenter, mouseleave etc.
  @HostListener('click') copyText() {

    // We need to create a dummy textarea with the text to be copied in the DOM
    var textArea = document.createElement("textarea");

    // Hide the textarea from actually showing
    textArea.style.position = 'fixed';
    textArea.style.top = '-999px';
    textArea.style.left = '-999px';
```

```

    textArea.style.width = '2em';
    textArea.style.height = '2em';
    textArea.style.padding = '0';
    textArea.style.border = 'none';
    textArea.style.outline = 'none';
    textArea.style.boxShadow = 'none';
    textArea.style.background = 'transparent';

    // Set the texarea's content to our value defined in our [text-copy] attribute
    textArea.value = this.text;
    document.body.appendChild(textArea);

    // This will select the textarea
    textArea.select();

    try {
        // Most modern browsers support execCommand('copy'|'cut'|'paste'), if it doesn't
it should throw an error
        var successful = document.execCommand('copy');
        var msg = successful ? 'successful' : 'unsuccessful';
        // Let the user know the text has been copied, e.g toast, alert etc.
        console.log(msg);
    } catch (err) {
        // Tell the user copying is not supported and give alternative, e.g alert window
with the text to copy
        console.log('unable to copy');
    }

    // Finally we remove the textarea from the DOM
    document.body.removeChild(textArea);
}
}

export const TEXT_COPY_DIRECTIVES = [TextCopyDirective];

```

some-page.component.html

Denken Sie daran, TEXT_COPY_DIRECTIVES in das directives-Array Ihrer Komponente einzufügen

```

...
<!-- Insert variable as the attribute's value, let textToBeCopied = 'http://facebook.com/'
-->
<button [text-copy]="textToBeCopied">Copy URL</button>
<button [text-copy]="'https://www.google.com/'">Copy URL</button>
...

```

Testen einer benutzerdefinierten Direktive

Gegeben eine Direktive, die Text zu Mausereignissen hervorhebt

```

import { Directive, ElementRef, HostListener, Input } from '@angular/core';

@Directive({ selector: '[appHighlight]' })
export class HighlightDirective {
    @Input('appHighlight') // tslint:disable-line no-input-rename
    highlightColor: string;
}

```

```

constructor(private el: ElementRef) { }

@HostListener('mouseenter')
onMouseEnter() {
  this.highlight(this.highlightColor || 'red');
}

@HostListener('mouseleave')
onMouseLeave() {
  this.highlight(null);
}

private highlight(color: string) {
  this.el.nativeElement.style.backgroundColor = color;
}
}

```

Es kann so getestet werden

```

import { ComponentFixture, ComponentFixtureAutoDetect, TestBed } from '@angular/core/testing';

import { Component } from '@angular/core';
import { HighlightDirective } from './highlight.directive';

@Component({
  selector: 'app-test-container',
  template: `
    <div>
      <span id="red" appHighlight>red text</span>
      <span id="green" [appHighlight]="'green'">green text</span>
      <span id="no">no color</span>
    </div>
  `
})
class ContainerComponent { }

const mouseEvents = {
  get enter() {
    const mouseenter = document.createEvent('MouseEvent');
    mouseenter.initEvent('mouseenter', true, true);
    return mouseenter;
  },
  get leave() {
    const mouseleave = document.createEvent('MouseEvent');
    mouseleave.initEvent('mouseleave', true, true);
    return mouseleave;
  },
};

describe('HighlightDirective', () => {
  let fixture: ComponentFixture<ContainerComponent>;
  let container: ContainerComponent;
  let element: HTMLElement;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [ContainerComponent, HighlightDirective],
      providers: [
        { provide: ComponentFixtureAutoDetect, useValue: true },

```

```

    ],
  });

  fixture = TestBed.createComponent(ContainerComponent);
  // fixture.detectChanges(); // without the provider
  container = fixture.componentInstance;
  element = fixture.nativeElement;
});

it('should set background-color to empty when mouse leaves with directive without
arguments', () => {
  const targetElement = <HTMLSpanElement>element.querySelector('#red');

  targetElement.dispatchEvent(mouseEvents.leave);
  expect(targetElement.style.backgroundColor).toEqual('');
});

it('should set background-color to empty when mouse leaves with directive with arguments',
() => {
  const targetElement = <HTMLSpanElement>element.querySelector('#green');

  targetElement.dispatchEvent(mouseEvents.leave);
  expect(targetElement.style.backgroundColor).toEqual('');
});

it('should set background-color red with no args passed', () => {
  const targetElement = <HTMLSpanElement>element.querySelector('#red');

  targetElement.dispatchEvent(mouseEvents.enter);
  expect(targetElement.style.backgroundColor).toEqual('red');
});

it('should set background-color green when passing green parameter', () => {
  const targetElement = <HTMLSpanElement>element.querySelector('#green');

  targetElement.dispatchEvent(mouseEvents.enter);
  expect(targetElement.style.backgroundColor).toEqual('green');
});
});

```

Richtlinien online lesen: <https://riptutorial.com/de/angular2/topic/2202/richtlinien>

Kapitel 55: Richtlinien und Komponenten: @Input @Output

Syntax

1. Einwegbindung von der übergeordneten Komponente an die verschachtelte Komponente:
[Eigenschaftsname]
2. Einwegbindung von verschachtelten Komponenten an übergeordnete Komponenten:
(propertyName)
3. Zweiwege-Bindung (auch bekannt als Bananenbox): [(propertyName)]

Examples

Eingabebeispiel

@input ist nützlich, um Daten zwischen Komponenten zu binden

Importieren Sie es zuerst in Ihre Komponente

```
import { Input } from '@angular/core';
```

Fügen Sie dann die Eingabe als Eigenschaft Ihrer Komponentenkategorie hinzu

```
@Input() car: any;
```

Nehmen wir an, der Selektor Ihrer Komponente ist 'car-component'. Fügen Sie beim Aufruf der Komponente das Attribut 'car'

```
<car-component [car]="car"></car-component>
```

Jetzt ist Ihr Auto als Attribut in Ihrem Objekt (this.car) verfügbar.

Vollständiges Beispiel:

1. car.entity.ts

```
export class CarEntity {  
  constructor(public brand : string, public color : string) {  
  }  
}
```

2. car.component.ts

```
import { Component, Input } from '@angular/core';  
import { CarEntity } from "../car.entity";
```



```

@Component({
  selector: 'car-component',
  template: require('./templates/car.html'),
})

export class CarComponent {
  @Input() car: CarEntity;

  constructor() {
    console.log('gros');
  }
}

```

3. garage.component.ts

```

import { Component } from '@angular/core';
import { CarEntity } from "../car.entity";
import { CarComponent } from "../car.component";

@Component({
  selector: 'garage',
  template: require('./templates/garage.html'),
  directives: [CarComponent]
})

export class GarageComponent {
  public cars : Array<CarEntity>;

  constructor() {
    var carOne : CarEntity = new CarEntity('renault', 'blue');
    var carTwo : CarEntity = new CarEntity('fiat', 'green');
    var carThree : CarEntity = new CarEntity('citroen', 'yellow');
    this.cars = [carOne, carTwo, carThree];
  }
}

```

4. garage.html

```

<div *ngFor="let car of cars">
  <car-component [car]="car"></car-component>
</div>

```

5. car.html

```

<div>
  <span>{{ car.brand }}</span> |
  <span>{{ car.color }}</span>
</div>

```

Angular2 @Input und @Output in einer verschachtelten Komponente

Eine Button-Direktive, die ein @Input() akzeptiert, um ein @Input() festzulegen, bis die Schaltfläche deaktiviert wird. Die übergeordnete Komponente kann ein Ereignis abhören, das bei Erreichen des @Output über @Output :

```

import { Component, Input, Output, EventEmitter } from '@angular/core';

@Component({
  selector: 'limited-button',
  template: `<button (click)="onClick()"
              [disabled]="disabled">
              <ng-content></ng-content>
            </button>`,
  directives: []
})

export class LimitedButton {
  @Input() clickLimit: number;
  @Output() limitReached: EventEmitter<number> = new EventEmitter();

  disabled: boolean = false;

  private clickCount: number = 0;

  onClick() {
    this.clickCount++;
    if (this.clickCount === this.clickLimit) {
      this.disabled = true;
      this.limitReached.emit(this.clickCount);
    }
  }
}

```

Übergeordnete Komponente, die die Button-Direktive verwendet und eine Benachrichtigung ausgibt, wenn das Klicklimit erreicht ist:

```

import { Component } from '@angular/core';
import { LimitedButton } from './limited-button.component';

@Component({
  selector: 'my-parent-component',
  template: `<limited-button [clickLimit]="2"
                          (limitReached)="onLimitReached($event)">
              You can only click me twice
            </limited-button>`,
  directives: [LimitedButton]
})

export class MyParentComponent {
  onLimitReached(clickCount: number) {
    alert('Button disabled after ' + clickCount + ' clicks.');
```

Angular2 @Input mit asynchronen Daten

Manchmal müssen Sie Daten asynchron abrufen, bevor Sie sie an eine zu verwendende untergeordnete Komponente übergeben. Wenn die untergeordnete Komponente versucht, die Daten zu verwenden, bevor sie empfangen wurden, wird ein Fehler ausgegeben. Sie können `ngOnChanges`, um Änderungen in den `@Input` Komponenten zu erkennen und zu warten, bis sie definiert sind, bevor Sie darauf `@Input`.

Übergeordnete Komponente mit asynchronem Aufruf an einen Endpunkt

```
import { Component, OnChanges, OnInit } from '@angular/core';
import { Http, Response } from '@angular/http';
import { ChildComponent } from './child.component';

@Component ({
  selector : 'parent-component',
  template : `
    <child-component [data]="asyncData"></child-component>
  `
})
export class ParentComponent {

  asyncData : any;

  constructor(
    private _http : Http
  ){}

  ngOnInit () {
    this._http.get('some.url')
      .map(this.extractData)
      .subscribe(this.handleData)
      .catch(this.handleError);
  }

  extractData (res:Response) {
    let body = res.json();
    return body.data || { };
  }

  handleData (data:any) {
    this.asyncData = data;
  }

  handleError (error:any) {
    console.error(error);
  }
}
```

Untergeordnete Komponente mit asynchronen Daten als Eingabe

Diese untergeordnete Komponente übernimmt die asynchronen Daten als Eingabe. Daher muss es warten, bis die Daten vorhanden sind, bevor sie verwendet werden. Wir verwenden `ngOnChanges`, das immer dann ausgelöst wird, wenn sich die Eingabe einer Komponente ändert. Überprüfen Sie, ob die Daten vorhanden sind, und verwenden Sie sie, falls dies der Fall ist. Beachten Sie, dass die Vorlage für das untergeordnete Element nicht angezeigt wird, wenn eine Eigenschaft, die von den übergebenen Daten abhängig ist, nicht `true` ist.

```
import { Component, OnChanges, Input } from '@angular/core';

@Component ({
  selector : 'child-component',
  template : `
    <p *ngIf="doesDataExist">Hello child</p>
  `
})
export class ChildComponent {

  doesDataExist: boolean = false;

  @Input('data') data : any;

  // Runs whenever component @Inputs change
  ngOnChanges () {
    // Check if the data exists before using it
    if (this.data) {
      this.useData(data);
    }
  }

  // contrived example to assign data to reliesOnData
  useData (data) {
    this.doesDataExist = true;
  }
}
```

Richtlinien und Komponenten: @Input @Output online lesen:

<https://riptutorial.com/de/angular2/topic/3046/richtlinien-und-komponenten---input--output>

Kapitel 56: Routing

Examples

Grundlegendes Routing

Der Router ermöglicht die Navigation von einer Ansicht zur anderen basierend auf den Benutzerinteraktionen mit der Anwendung.

Im Folgenden werden die Schritte zum Implementieren des grundlegenden Routings in Angular 2 beschrieben.

Grundlegende Vorsichtsmaßnahme : Stellen Sie sicher, dass Sie das Tag haben

```
<base href='/>
```

als erstes Kind unter Ihrem Head-Tag in Ihrer index.html-Datei. Dieses Tag weist darauf hin, dass Ihr App-Ordner der Anwendungsstamm ist. Angular 2 würde dann wissen, um Ihre Links zu organisieren.

Zuerst überprüfen Sie, ob Sie in package.json auf korrekte / aktuelle Routing-Abhängigkeiten verweisen.

```
"dependencies": {  
  .....  
  "@angular/router": "3.0.0-beta.1",  
  .....  
}
```

Der zweite Schritt besteht darin, die Route gemäß ihrer Klassendefinition zu definieren.

```
class Route {  
  path : string  
  pathMatch : 'full'|'prefix'  
  component : Type|string  
  .....  
}
```

Importieren Sie in einer `route/routes.ts` (`route/routes.ts`) alle Komponenten, die Sie für verschiedene Routingpfade konfigurieren müssen. Ein leerer Pfad bedeutet, dass die Ansicht standardmäßig geladen wird. ":" im Pfad gibt dynamische Parameter an, die an die geladene Komponente übergeben werden.

Die Routen werden der Anwendung über die Abhängigkeitsinjektion zur Verfügung gestellt. Die `ProviderRouter`-Methode wird mit dem Parameter `RouterConfig` als Parameter aufgerufen, damit sie in die Komponenten eingefügt werden kann, um routingspezifische Aufgaben aufzurufen.

```
import { provideRouter, RouterConfig } from '@angular/router';
```

```

import { BarDetailComponent } from '../components/bar-detail.component';
import { DashboardComponent } from '../components/dashboard.component';
import { LoginComponent } from '../components/login.component';
import { SignupComponent } from '../components/signup.component';

export const appRoutes: RouterConfig = [
  { path: '', pathMatch: 'full', redirectTo: 'login' },
  { path: 'dashboard', component: DashboardComponent },
  { path: 'bars/:id', component: BarDetailComponent },
  { path: 'login', component: LoginComponent },
  { path: 'signup', component: SignupComponent }
];

export const APP_ROUTER_PROVIDER = [provideRouter(appRoutes)];

```

Der dritte Schritt besteht darin, den Routenanbieter zu booten.

In Ihrer `main.ts` (es kann ein beliebiger Name sein. Grundsätzlich sollte Ihre Hauptdatei in `systemjs.config` definiert sein.)

```

import { bootstrap } from '@angular/platform-browser-dynamic';
import { AppComponent } from './components/app.component';
import { APP_ROUTER_PROVIDER } from './routes/routes';

bootstrap(AppComponent, [ APP_ROUTER_PROVIDER ]).catch(err => console.error(err));

```

Der vierte Schritt ist das Laden / Anzeigen der Routerkomponenten basierend auf dem Pfad, auf den zugegriffen wird. Die Direktive wird verwendet, um den Winkel anzugeben, wo die Komponente geladen werden soll. Um den Import von `ROUTER_DIRECTIVES` zu verwenden.

```

import { ROUTER_DIRECTIVES } from '@angular/router';

@Component({
  selector: 'demo-app',
  template: `
    .....
    <div>
      <router-outlet></router-outlet>
    </div>
    .....
  `,
  // Add our router directives we will be using
  directives: [ROUTER_DIRECTIVES]
})

```

Der fünfte Schritt besteht darin, die anderen Routen zu verknüpfen. Standardmäßig lädt `RouterOutlet` die Komponente, für die in der `RouterConfig` ein leerer Pfad angegeben ist. Die `RouterLink`-Anweisung wird mit dem HTML-Ankertag verwendet, um die an Routen angeschlossenen Komponenten zu laden. `RouterLink` generiert das `href`-Attribut, das zum Generieren von Links verwendet wird. Für Ex:

```

import { Component } from '@angular/core';
import { ROUTER_DIRECTIVES } from '@angular/router';

@Component({

```

```

selector: 'demo-app',
template: `
  <a [routerLink]="['/login']">Login</a>
  <a [routerLink]="['/signup']">Signup</a>
  <a [routerLink]="['/dashboard']">Dashboard</a>
  <div>
    <router-outlet></router-outlet>
  </div>
`,
// Add our router directives we will be using
directives: [ROUTER_DIRECTIVES]
})
export class AppComponent { }

```

Nun sind wir gut mit dem Routing zum statischen Pfad. RouterLink unterstützt den dynamischen Pfad auch, indem zusätzliche Parameter zusammen mit dem Pfad übergeben werden.

{Komponente} aus '@ angle / core' importieren; importiere {ROUTER_DIRECTIVES} von '@ angle / router';

```

@Component({
  selector: 'demo-app',
  template: `
    <ul>
      <li *ngFor="let bar of bars | async">
        <a [routerLink]="['/bars', bar.id]">
          {{bar.name}}
        </a>
      </li>
    </ul>
    <div>
      <router-outlet></router-outlet>
    </div>
  `,
  // Add our router directives we will be using
  directives: [ROUTER_DIRECTIVES]
})
export class AppComponent { }

```

RouterLink verwendet ein Array, in dem das erste Element der Pfad für das Routing und die nachfolgenden Elemente die dynamischen Routing-Parameter sind.

Untergeordnete Routen

Manchmal ist es sinnvoll, Ansichten oder Routen ineinander zu verschachteln. Auf dem Dashboard möchten Sie beispielsweise mehrere Unteransichten, ähnlich wie Registerkarten, die jedoch über das Routingsystem implementiert werden, um die Projekte, Kontakte und Nachrichten der Benutzer anzuzeigen. Um solche Szenarien zu unterstützen, können Sie mit dem Router untergeordnete Routen definieren.

Zuerst passen wir unsere RouterConfig von oben an und fügen die untergeordneten Routen hinzu:

```

import { ProjectsComponent } from '../components/projects.component';
import { MessagesComponent } from '../components/messages.component';

```

```

export const appRoutes: RouterConfig = [
  { path: '', pathMatch: 'full', redirectTo: 'login' },
  { path: 'dashboard', component: DashboardComponent,
    children: [
      { path: '', redirectTo: 'projects', pathMatch: 'full' },
      { path: 'projects', component: 'ProjectsComponent' },
      { path: 'messages', component: 'MessagesComponent' }
    ] },
  { path: 'bars/:id', component: BarDetailComponent },
  { path: 'login', component: LoginComponent },
  { path: 'signup', component: SignupComponent }
];

```

Nachdem wir nun unsere untergeordneten Routen definiert haben, müssen wir sicherstellen, dass diese untergeordneten Routen in unserer `DashboardComponent` angezeigt werden können, da wir die untergeordneten Routen hinzugefügt haben. Bisher haben wir erfahren, dass die Komponenten in einem `<router-outlet></router-outlet>`-Tag angezeigt werden. Ähnlich deklarieren wir ein weiteres `RouterOutlet` in der `DashboardComponent`:

```

import { Component } from '@angular/core';

@Component({
  selector: 'dashboard',
  template: `
    <a [routerLink]="['projects']">Projects</a>
    <a [routerLink]="['messages']">Messages</a>
    <div>
      <router-outlet></router-outlet>
    </div>
  `
})
export class DashboardComponent { }

```

Wie Sie sehen, haben wir ein weiteres `RouterOutlet` hinzugefügt, in dem die `RouterOutlet` Routen angezeigt werden. Normalerweise wird die Route mit einem leeren Pfad angezeigt. Wir richten jedoch eine Weiterleitung zur `projects`, da diese sofort angezeigt werden soll, wenn die `dashboard` Route geladen wird. Davon abgesehen brauchen wir eine leere Route, ansonsten erhalten Sie eine Fehlermeldung wie diese:

```
Cannot match any routes: 'dashboard'
```

Durch Hinzufügen der *leeren* Route, dh einer Route mit einem leeren Pfad, haben wir einen Einstiegspunkt für den Router definiert.

ResolveData

In diesem Beispiel wird gezeigt, wie Sie aus einem Dienst abgerufene Daten auflösen können, bevor Sie die Ansicht Ihrer Anwendung darstellen.

Verwendet zum Zeitpunkt des Schreibens Winkel / Router 3.0.0-beta.2

Benutzer.Service.ts


```

...
import { Http, Response } from '@angular/http';
import { Observable } from 'rxjs/Rx';
import { User } from './user.ts';

@Injectable()
export class UsersService {

  constructor(public http:Http) {}

  /**
   * Returns all users
   * @returns {Observable<User[]>}
   */
  index():Observable<User[]> {

    return this.http.get('http://mywebsite.com/api/v1/users')
      .map((res:Response) => res.json());
  }

  /**
   * Returns a user by ID
   * @param id
   * @returns {Observable<User>}
   */
  get(id:number|string):Observable<User> {

    return this.http.get('http://mywebsite.com/api/v1/users/' + id)
      .map((res:Response) => res.json());
  }
}

```

users.resolver.ts

```

...
import { UsersService } from './users.service.ts';
import { Observable } from 'rxjs/Rx';
import {
  Resolve,
  ActivatedRouteSnapshot,
  RouterStateSnapshot
} from "@angular/router";

@Injectable()
export class UsersResolver implements Resolve<User[] | User> {

  // Inject UsersService into the resolver
  constructor(private service:UsersService) {}

  resolve(route:ActivatedRouteSnapshot, state:RouterStateSnapshot):Observable<User[] | User>
  {
    // If userId param exists in current URL, return a single user, else return all users
    // Uses brackets notation to access `id` to suppress editor warning, may use dot
    notation if you create an interface extending ActivatedRoute with an optional id? attribute
    if (route.params['id']) return this.service.get(route.params['id']);
  }
}

```

```

        return this.service.index();
    }
}

```

Benutzer.Komponente.ts

Dies ist eine Seitenkomponente mit einer Liste aller Benutzer. Es funktioniert auf ähnliche Weise für die Komponente "Benutzerdetailseite". Ersetzen Sie `data.users` durch `data.user` oder den in `app.routes.ts` definierten *Schlüssel* (siehe unten).

```

...
import { ActivatedRoute } from "@angular/router";

@Component(...)
export class UsersComponent {

    users:User[];

    constructor(route: ActivatedRoute) {
        route.data.subscribe(data => {
            // data['Match key defined in RouterConfig, see below']
            this.users = data.users;
        });
    }

    /**
     * It is not required to unsubscribe from the resolver as Angular's HTTP
     * automatically completes the subscription when data is received from the server
     */
}

```

app.routes.ts

```

...
import { UsersResolver } from '../resolvers/users.resolver';

export const routes:RouterConfig = <RouterConfig>[
    ...
    {
        path: 'user/:id',
        component: UserComponent,
        resolve: {
            // hence data.user in UserComponent
            user: UsersResolver
        }
    },
    {
        path: 'users',
        component: UsersComponent,
        resolve: {
            // hence data.users in UsersComponent, note the pluralisation
            users: UsersResolver
        }
    }
]

```

```
    },  
    ...  
  ]  
  ...
```

app.resolver.ts

Bündeln Sie optional mehrere Resolver zusammen.

WICHTIG: Dienste, die in Resolver verwendet werden, müssen zuerst importiert werden. Andernfalls wird der Fehler "Kein Anbieter für .. Resolver" angezeigt. Beachten Sie, dass diese Dienste weltweit verfügbar sind und Sie nicht mehr bei den *providers* einer Komponente *providers* werden müssen. Stellen Sie sicher, dass Sie sich von keinem Abonnement abmelden, um Speicherverluste zu vermeiden

```
...  
import { UserService } from './users.service';  
import { UsersResolver } from './users.resolver';  
  
export const ROUTE_RESOLVERS = [  
  ...,  
  UserService,  
  UsersResolver  
]
```

main.browser.ts

Resolver müssen beim Bootstrapping injiziert werden.

```
...  
import {bootstrap} from '@angular/platform-browser-dynamic';  
import { ROUTE_RESOLVERS } from './app.resolver';  
  
bootstrap(<Type>App, [  
  ...  
  ...ROUTE_RESOLVERS  
)  
.catch(err => console.error(err));
```

Routing mit Kindern

Im Gegensatz zur Originaldokumentation habe ich festgestellt, dass dies der richtige Weg ist, um untergeordnete Routen in der Datei *app.routing.ts* oder *app.module.ts* richtig zu verschachteln (je nach Präferenz). Dieser Ansatz funktioniert bei Verwendung von WebPack oder SystemJS.

Das folgende Beispiel zeigt Routen für Home, Home / Counter und Home / Counter / Fetch-Daten. Die erste und letzte Route sind Beispiele für Weiterleitungen. Am Ende des Beispiels können Sie die zu importierende Route in einer separaten Datei exportieren. Für ex. *app.module.ts*

Angular erfordert, dass Sie über eine Pfadlose Route im untergeordneten Array verfügen, die die übergeordnete Komponente enthält, um die übergeordnete Route darzustellen. Dies ist ein wenig verwirrend, aber wenn Sie über eine leere URL für eine untergeordnete Route nachdenken,

entspricht sie im Wesentlichen der gleichen URL wie die übergeordnete Route.

```
import { NgModule } from "@angular/core";
import { RouterModule, Routes } from "@angular/router";

import { HomeComponent } from "../components/home/home.component";
import { FetchDataComponent } from "../components/fetchdata/fetchdata.component";
import { CounterComponent } from "../components/counter/counter.component";

const appRoutes: Routes = [
  {
    path: "",
    redirectTo: "home",
    pathMatch: "full"
  },
  {
    path: "home",
    children: [
      {
        path: "",
        component: HomeComponent
      },
      {
        path: "counter",
        children: [
          {
            path: "",
            component: CounterComponent
          },
          {
            path: "fetch-data",
            component: FetchDataComponent
          }
        ]
      }
    ]
  },
  {
    path: "**",
    redirectTo: "home"
  }
];

@NgModule({
  imports: [
    RouterModule.forRoot(appRoutes)
  ],
  exports: [
    RouterModule
  ]
})
export class AppRoutingModule { }
```

[Tolles Beispiel und Beschreibung über Siraj](#)

[Routing online lesen: https://riptutorial.com/de/angular2/topic/2334/routing](https://riptutorial.com/de/angular2/topic/2334/routing)

Kapitel 57: Routing (3.0.0+)

Bemerkungen

Es gibt ein paar weitere Tricks, die wir mit dem Router machen können (z. B. Zugriffsbeschränkungen), aber diese können in einem separaten Tutorial behandelt werden.

Wenn Sie eine neue Route benötigen, ändern `app.routes.ts` einfach `app.routes.ts` und führen Sie die folgenden Schritte aus:

1. Importieren Sie die Komponente
2. Fügen Sie das `routes` Array hinzu. Stellen Sie sicher, dass Sie einen neuen `path` und eine neue `component` .

Examples

Bootstrapping

Jetzt, da die Routen definiert sind, müssen wir unsere Anwendung über die Routen informieren. Bootstrap des Anbieters, den wir im vorherigen Beispiel exportiert haben.

Finden Sie Ihre Bootstrap-Konfiguration (sollte sich in `main.ts` , **Ihre Laufleistung kann jedoch variieren**).

```
//main.ts

import {bootstrap} from '@angular/platform-browser-dynamic';

//Import the App component (root component)
import { App } from './app/app';

//Also import the app routes
import { APP_ROUTES_PROVIDER } from './app/app.routes';

bootstrap(App, [
  APP_ROUTES_PROVIDER,
])
.catch(err => console.error(err));
```

Router-Steckdose konfigurieren

Nun, da der Router konfiguriert ist und unsere App mit den Routen umgehen kann, müssen die tatsächlichen Komponenten angezeigt werden, die wir konfiguriert haben.

Konfigurieren Sie dazu Ihre HTML-Vorlage / Datei für Ihre Komponente der **obersten Ebene (App- Komponente)** wie folgt:

```
//app.ts
```

```

import {Component} from '@angular/core';
import {Router, ROUTER_DIRECTIVES} from '@angular/router';

@Component({
  selector: 'app',
  templateUrl: 'app.html',
  styleUrls: ['app.css'],
  directives: [
    ROUTER_DIRECTIVES,
  ]
})
export class App {
  constructor() {
  }
}

<!-- app.html -->

<!-- All of your 'views' will go here -->
<router-outlet></router-outlet>

```

Das Element `<router-outlet></router-outlet>` wechselt den Inhalt anhand der Route. Ein weiterer guter Aspekt dieses Elements ist, dass es *nicht* das einzige Element in Ihrem HTML-Code sein muss.

Beispiel: Angenommen, Sie wollten auf jeder Seite eine Symbolleiste, die zwischen den Routen konstant bleibt, ähnlich wie der Stapelüberlauf aussieht. Sie können den `<router-outlet>` unter Elementen verschachteln, sodass nur bestimmte Teile der Seite geändert werden.

Routen ändern (mithilfe von Vorlagen und Anweisungen)

Nun, da die Routen eingerichtet sind, brauchen wir einen Weg, um die Routen tatsächlich zu ändern.

In diesem Beispiel wird gezeigt, wie Routen mithilfe der Vorlage geändert werden. Es ist jedoch möglich, Routen in TypeScript zu ändern.

Hier ist ein Beispiel (ohne Bindung):

```
<a routerLink="/home">Home</a>
```

Wenn der Benutzer auf diesen Link klickt, wird er nach `/home` . Der Router weiß, wie mit `/home` umzugehen ist, und zeigt die `Home` Komponente an.

Hier ein Beispiel mit Datenbindung:

```
<a *ngFor="let link of links" [routerLink]="link">{{link}}</a>
```

Für das `app.ts` eines Arrays, das als `links` , müssen Sie dies zu `app.ts` hinzufügen:

```
public links[] = [
```

```
'home',  
'login'  
]
```

Dadurch wird das Array durchlaufen und ein `<a>`-Element mit der `routerLink` Direktive = dem Wert des aktuellen Elements im Array `routerLink` Dies erzeugt `routerLink` :

```
<a routerLink="home">home</a>  
<a routerLink="login">login</a>
```

Dies ist besonders hilfreich, wenn Sie viele Links haben oder die Links ständig geändert werden müssen. Wir lassen Angular die anstrengende Arbeit beim Hinzufügen von Links erledigen, indem wir einfach die benötigten Informationen eingeben.

Im Moment ist `links[]` statisch, es ist jedoch möglich, Daten aus einer anderen Quelle einzugeben.

Routen einstellen

HINWEIS: Dieses Beispiel basiert auf der Version 3.0.0.-beta.2 des @ angle / router. Zum Zeitpunkt des Schreibens ist dies die neueste Version des Routers.

Um den Router zu verwenden, definieren Sie die Routen in einer neuen TypeScript-Datei wie dieser

```
//app.routes.ts  
  
import {provideRouter} from '@angular/router';  
  
import {Home} from './routes/home/home';  
import {Profile} from './routes/profile/profile';  
  
export const routes = [  
  {path: '', redirectTo: 'home'},  
  {path: 'home', component: Home},  
  {path: 'login', component: Login},  
];  
  
export const APP_ROUTES_PROVIDER = provideRouter(routes);
```

In der ersten Zeile importieren wir " `provideRouter` damit unsere Anwendung während der Bootstrap-Phase über die Routen `provideRouter` werden kann.

`Home` und `Profile` sind nur zwei Komponenten. Sie müssen jede `Component` Sie benötigen, als Route importieren.

Exportieren Sie dann das Array von Routen.

`path` : Der Pfad zur Komponente. **SIE MÜSSEN NICHT '/'!** VERWENDEN . Angular führt dies automatisch aus

`component` : Die Komponente, die geladen werden soll, wenn auf die Route zugegriffen wird

`redirectTo` : *Optional* . Wenn Sie einen Benutzer beim Zugriff auf eine bestimmte Route automatisch umleiten müssen, geben Sie dies an.

Zum Schluss exportieren wir den konfigurierten Router. `provideRouter` eines Providers von `providerRouter`, damit wir unsere Route verbessern können.

Zugriff auf oder von einer Route steuern

Der Standard-Angular-Router ermöglicht die uneingeschränkte Navigation von und zu jeder Route. Dies ist nicht immer das gewünschte Verhalten.

In einem Szenario, in dem ein Benutzer unter Umständen dazu berechtigt ist, zu einer Route zu navigieren oder von dieser abzureisen, kann ein **Route Guard** verwendet werden, um dieses Verhalten einzuschränken.

Wenn Ihr Szenario eine der folgenden Bedingungen erfüllt, ziehen Sie die Verwendung eines Route Guard in Betracht.

- Der Benutzer muss authentifiziert sein, um zur Zielkomponente zu navigieren.
- Der Benutzer muss berechtigt sein, zur Zielkomponente zu navigieren.
- Komponente erfordert vor der Initialisierung eine asynchrone Anforderung.
- Die Komponente erfordert eine Benutzereingabe, bevor von navigiert wird.

Funktionsweise von Route Guards

Route Guards funktionieren, indem sie einen booleschen Wert zurückgeben, um das Verhalten der Routernavigation zu steuern. Wenn *true* zurückgegeben wird, fährt der Router mit der Navigation zur Zielkomponente fort. Wenn *false* zurückgegeben wird, verweigert der Router die Navigation zur Zielkomponente.

Route Guard-Schnittstellen

Der Router unterstützt mehrere Guard-Schnittstellen:

- *CanActivate* : tritt zwischen *Routennavigation* auf.
- *CanActivateChild* : tritt zwischen der *Routennavigation* zu einer *untergeordneten* Route auf.
- *CanDeactivate* : tritt auf, wenn Sie von der aktuellen Route weg navigieren.
- *CanLoad* : tritt zwischen der *Routennavigation* zu einem asynchron geladenen Funktionsmodul auf.
- *Auflösen* : Dient zum Abrufen von Daten vor der Aktivierung der Route.

Diese Schnittstellen können in Ihrem Guard implementiert werden, um Zugriff auf bestimmte Prozesse der Navigation zu gewähren oder zu entfernen.

Synchronous vs. Asynchronous Route Guards

Routenschutz ermöglicht synchrone und asynchrone Vorgänge zur bedingten Steuerung der Navigation.

Synchroner Routenschutz

Ein synchroner Routenwächter gibt einen Booleschen Wert zurück, z. B. durch Berechnen eines unmittelbaren Ergebnisses, um die Navigation bedingt zu steuern.

```
import { Injectable }    from '@angular/core';
import { CanActivate }  from '@angular/router';

@Injectable()
export class SynchronousGuard implements CanActivate {
  canActivate() {
    console.log('SynchronousGuard#canActivate called');
    return true;
  }
}
```

Asynchroner Routenschutz

Für komplexeres Verhalten kann ein Route Guard die Navigation asynchron blockieren. Ein asynchroner Route Guard kann ein Observable oder Promise zurückgeben.

Dies ist nützlich, wenn beispielsweise auf die Beantwortung einer Frage auf Benutzereingaben gewartet wird, auf das erfolgreiche Speichern der Änderungen auf dem Server gewartet wird oder auf den Empfang von Daten, die von einem Remote-Server abgerufen werden.

```
import { Injectable }    from '@angular/core';
import { CanActivate, Router, ActivatedRouteSnapshot, RouterStateSnapshot } from
 '@angular/router';
import { Observable }    from 'rxjs/Rx';
import { MockAuthenticationService } from './authentication/authentication.service';

@Injectable()
export class AsynchronousGuard implements CanActivate {
  constructor(private router: Router, private auth: MockAuthenticationService) {}

  canActivate(route:ActivatedRouteSnapshot,
state:RouterStateSnapshot):Observable<boolean>|boolean {
    this.auth.subscribe((authenticated) => {
      if (authenticated) {
        return true;
      }
      this.router.navigateByUrl('/login');
    });
  }
}
```

```
        return false;
    });
}
}
```

Wächter zur Routenkonfiguration hinzufügen

Datei *app.routes*

Geschützte Routen haben `canActivate` `binded` Guard

```
import { provideRouter, Router, RouterConfig, CanActivate } from '@angular/router';

//components
import { LoginComponent } from '../login/login.component';
import { DashboardComponent } from '../dashboard/dashboard.component';

export const routes: RouterConfig = [
  { path: 'login', component: LoginComponent },
  { path: 'dashboard', component: DashboardComponent, canActivate: [AuthGuard] }
]
```

Exportieren Sie das **APP_ROUTER_PROVIDERS** , das in App-Bootstrap verwendet werden soll

```
export const APP_ROUTER_PROVIDERS = [
  AuthGuard,
  provideRouter(routes)
];
```

Verwenden Sie Guard im App-Bootstrap

Datei *main.ts* (oder *boot.ts*)

Betrachten Sie die obigen Beispiele:

1. **Erstellen Sie die Wache** (wo die Wache erstellt wird) und
2. **Guard zur Routenkonfiguration hinzufügen** (wenn der Guard für die Route **konfiguriert** ist, wird **APP_ROUTER_PROVIDERS** exportiert)

Wir können den Bootstrap wie folgt an Guard koppeln

```
import { bootstrap } from '@angular/platform-browser-dynamic';
import { provide } from '@angular/core';

import { APP_ROUTER_PROVIDERS } from './app.routes';
import { AppComponent } from './app.component';

bootstrap(AppComponent, [
  APP_ROUTER_PROVIDERS
])
.then(success => console.log(`Bootstrap success`))
.catch(error => console.log(error));
```

Resolver und Guards verwenden

Wir verwenden einen Toplevel-Guard in unserer Routenkonfiguration, um den aktuellen Benutzer beim Laden der ersten Seite `currentUser`, und einen Resolver, um den Wert des aktuellen Benutzers, des authentifizierten Benutzers aus dem Backend, zu speichern.

Eine vereinfachte Version unserer Implementierung sieht folgendermaßen aus:

Hier ist unsere Top-Level-Route:

```
export const routes = [
  {
    path: 'Dash',
    pathMatch: 'prefix',
    component: DashCmp,
    canActivate: [AuthGuard],
    resolve: {
      currentUser: CurrentUserResolver
    },
    children: [...[
      {
        path: '',
        component: ProfileCmp,
        resolve: {
          currentUser: currentUser
        }
      }
    ]]
  }
];
```

Hier ist unser `AuthService`

```
import { Injectable } from '@angular/core';
import { Http, Headers, RequestOptions } from '@angular/http';
import { Observable } from 'rxjs/Rx';
import 'rxjs/add/operator/do';

@Injectable()
export class AuthService {
  constructor(http: Http) {
    this.http = http;

    let headers = new Headers({ 'Content-Type': 'application/json' });
    this.options = new RequestOptions({ headers: headers });
  }
  fetchCurrentUser() {
    return this.http.get('/api/users/me')
      .map(res => res.json())
      .do(val => this.currentUser = val);
  }
}
```

Hier ist unser `AuthGuard`:

```
import { Injectable } from '@angular/core';
import { CanActivate } from "@angular/router";
import { Observable } from 'rxjs/Rx';
```

```

import { AuthService } from '../services/AuthService';

@Injectable()
export class AuthGuard implements CanActivate {
  constructor(auth: AuthService) {
    this.auth = auth;
  }
  canActivate(route, state) {
    return Observable
      .merge(this.auth.fetchCurrentUser(), Observable.of(true))
      .filter(x => x == true);
  }
}

```

Hier ist unser `CurrentUserResolver` :

```

import { Injectable } from '@angular/core';
import { Resolve } from "@angular/router";
import { Observable } from 'rxjs/Rx';

import { AuthService } from '../services/AuthService';

@Injectable()
export class CurrentUserResolver implements Resolve {
  constructor(auth: AuthService) {
    this.auth = auth;
  }
  resolve(route, state) {
    return this.auth.currentUser;
  }
}

```

Routing (3.0.0+) online lesen: <https://riptutorial.com/de/angular2/topic/1208/routing--3-0-0plus->

Kapitel 58: Seitentitel

Einführung

Wie können Sie den Titel der Seite ändern?

Syntax

- setTitle(newTitle: string): void;
- getTitle(): string;

Examples

den Seitentitel ändern

1. Zuerst müssen wir den Titelservice anbieten.
2. setTitle verwenden

```
import {Title} from "@angular/platform-browser";
@Component({
  selector: 'app',
  templateUrl: './app.component.html',
  providers : [Title]
})

export class AppComponent implements {
  constructor( private title: Title) {
    this.title.setTitle('page title changed');
  }
}
```

Seitentitel online lesen: <https://riptutorial.com/de/angular2/topic/8954/seitentitel>

Kapitel 59: Servicemitarbeiter

Einführung

Wir werden sehen, wie Sie einen Service einrichten, der mit Winkle arbeitet, damit unsere Web-App Offline-Funktionen haben kann.

Ein Service Worker ist ein spezielles Skript, das im Hintergrund im Browser ausgeführt wird und Netzwerkanforderungen an einen bestimmten Ursprung verwaltet. Es wurde ursprünglich von einer App installiert und verbleibt auf dem Computer / Gerät des Benutzers. Sie wird vom Browser aktiviert, wenn eine Seite aus ihrem Ursprung geladen wird, und hat die Möglichkeit, während des Ladens der Seite auf HTTP-Anforderungen zu antworten

Examples

Füge Service Worker unserer App hinzu

Erst wenn Sie mobile.angular.io konsultieren, funktioniert die Flagge `--mobile` nicht mehr.

Um zu beginnen, können wir ein normales Projekt mit Winkelcli erstellen.

```
ng new serviceWorking-example
cd serviceWorking-example
```

Nun das Wichtigste, um zu sagen, dass wir Service-Mitarbeiter einsetzen wollen, müssen wir Folgendes tun:

```
ng set apps.0.serviceWorker = true
```

Wenn Sie aus irgendeinem Grund nicht `@ eckig / Service-Worker` installiert haben, wird eine Meldung angezeigt:

```
Ihr Projekt ist mit serviceWorker = true konfiguriert, aber @ angle / Service-Worker ist
nicht installiert. Führen Sie npm install --save-dev @angular/service-worker und
versuchen Sie es erneut, oder führen Sie ng set apps.0.serviceWorker=false in Ihrer
.angular-cli.json aus.
```

Überprüfen Sie die `.angular-cli.json` und Sie sollten jetzt Folgendes sehen: `"serviceWorker": true`

Wenn dieses Flag aktiviert ist, werden Produktions-Builds mit einem Service-Mitarbeiter eingerichtet.

Es wird eine Datei `ngsw-manifest.json` generiert (oder erweitert, falls im Stammverzeichnis des Projekts eine Datei `ngsw-manifest.json` erstellt wird. Normalerweise wird dies zur Festlegung des Routings verwendet. In Zukunft wird dies wahrscheinlich automatisch erfolgen.) in `dist / root`, und das Service-Worker-Skript wird dort kopiert. Zu `index.html` wird ein kurzes Skript hinzugefügt, um

den Service Worker zu registrieren.

Wenn wir die App jetzt im Produktionsmodus erstellen, erstellen Sie - prod

Und überprüfen Sie dist / Ordner.

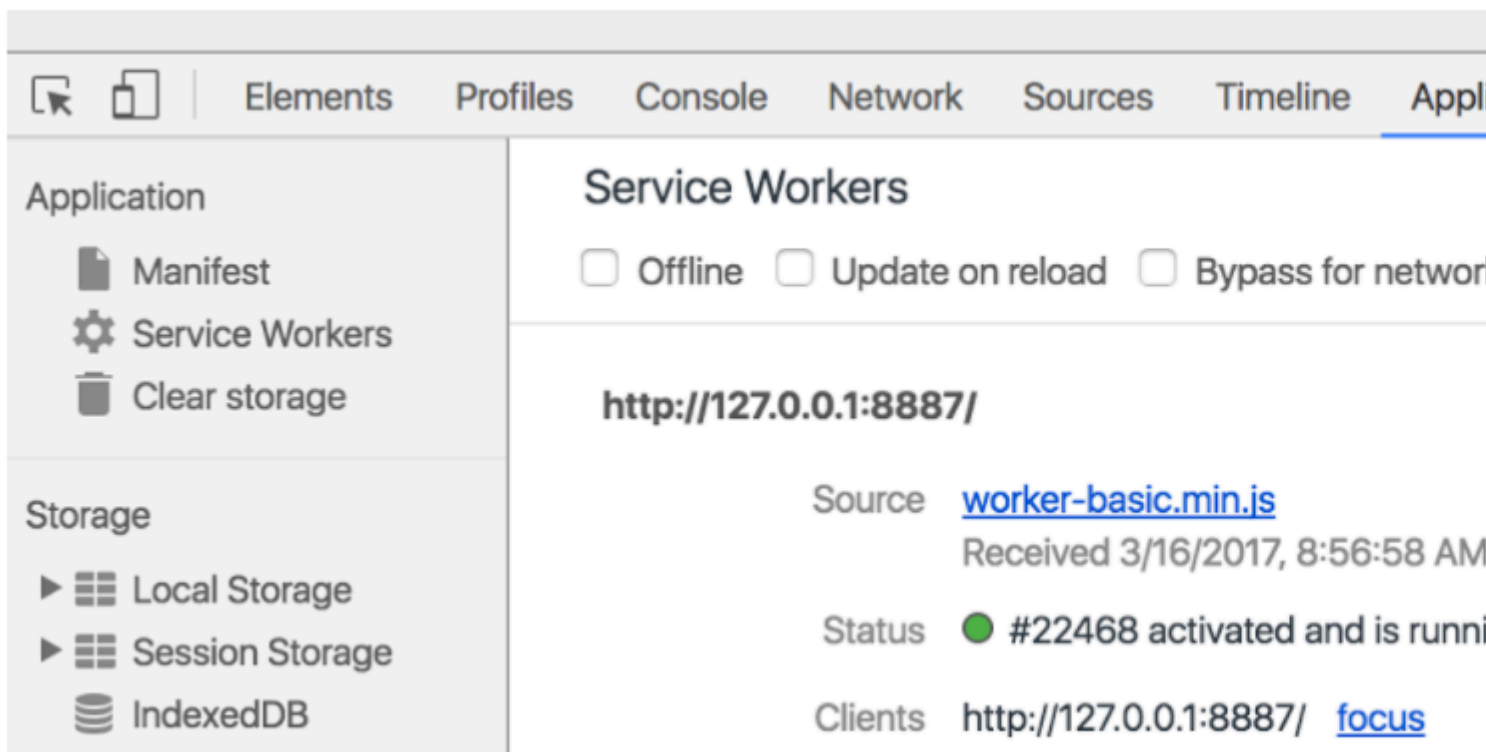
Dort sehen Sie drei neue Dateien:

- worker-basic.min.js
- sw-register.HASH.bundle.js
- ngsw-manifest.json

Außerdem enthält index.html jetzt dieses sw-register-Skript, das einen Angular Service Worker (ASW) für uns registriert.

Aktualisieren Sie die Seite in Ihrem Browser (wird vom Webserver für Chrome bereitgestellt).

Öffnen Sie die Entwicklertools. Gehen Sie zur Anwendung -> Servicemitarbeiter



Gut, jetzt ist der Service Worker in Betrieb!

Jetzt sollte unsere Anwendung schneller geladen werden und wir sollten die App offline verwenden können.

Wenn Sie nun den Offline-Modus in der Chrome-Konsole aktivieren, sollten Sie feststellen, dass unsere App in <http://localhost:4200/index.html> ohne Internetverbindung funktioniert.

In <http://localhost:4200/haben> wir jedoch ein Problem und werden nicht geladen. Dies liegt daran, dass der statische Inhaltscache nur die im Manifest aufgelisteten Dateien bedient.

Wenn das Manifest beispielsweise eine URL von /index.html deklariert, werden Anforderungen an /index.html vom Cache beantwortet, eine Anforderung an / oder / some / route jedoch an das Netzwerk.

Hier kommt das Plug-In für die Routenumleitung zum Einsatz. Es liest eine Routing-Konfiguration aus dem Manifest und leitet konfigurierte Routen an eine angegebene Indexroute weiter.

Derzeit muss dieser Konfigurationsabschnitt von Hand geschrieben werden (19-7-2017). Eventuell wird es aus der in der Anwendungsquelle vorhandenen Routenkonfiguration generiert.

Wenn wir nun also ngsw-manifest.json im Stammverzeichnis des Projekts erstellen

```
{
  "routing": {
    "routes": {
      "/": {
        "prefix": false
      }
    },
    "index": "/index.html"
  }
}
```

Und wir bauen unsere App erneut. Wenn wir nun zu <http://localhost:4200/> gehen , sollten wir zu <http://localhost:4200/index.html> umgeleitet werden.

Weitere Informationen zum Routing finden Sie in der [offiziellen Dokumentation](#)

Hier finden Sie weitere Unterlagen zu Servicemitarbeitern:

<https://developers.google.com/web/fundamentals/getting-started/primers/service-workers>

https://docs.google.com/document/d/19S5ozevWighny788nI99worpcIMDnwWVmaJDGf_RoDY/edit#

Und hier sehen Sie einen alternativen Weg, um den Dienst zu implementieren, der mit der SW-Precache-Bibliothek arbeitet:

<https://coryryan.com/blog/fast-offline-angular-apps-mit-service-workers>

Servicemitarbeiter online lesen: <https://riptutorial.com/de/angular2/topic/10809/servicemitarbeiter>

Kapitel 60: Spott @ ngrx / Store

Einführung

@ ngrx / Store wird in Angular 2-Projekten immer häufiger eingesetzt. Daher muss der Store in den Konstruktor einer Komponente oder eines Dienstes eingefügt werden, die ihn verwenden möchte. Unit Testing Store ist nicht so einfach wie das Testen eines einfachen Services. Wie bei vielen Problemen gibt es unzählige Möglichkeiten, Lösungen zu implementieren. Das Grundrezept besteht jedoch darin, eine Mock-Klasse für die Observer-Schnittstelle und eine Mock-Klasse für Store zu schreiben. Dann können Sie Store als Anbieter in Ihr TestBed einspritzen.

Parameter

Name	Beschreibung
Wert	nächster zu beobachtender Wert
Error	Beschreibung
Irr	Fehler geworfen werden
Super	Beschreibung
Aktion \$	Scheinbeobachter, der nichts tut, sofern er nicht in der Scheinklasse definiert ist
actionReducer \$	Scheinbeobachter, der nichts tut, sofern er nicht in der Scheinklasse definiert ist
obs \$	Schein Beobachtbar

Bemerkungen

Observer ist generisch, muss jedoch vom Typ `any`, um die Komplexität von `any` zu vermeiden. Der Grund für diese Komplexität ist, dass der Konstruktor von Store Observer-Argumente mit unterschiedlichen generischen Typen erwartet. Die Verwendung von `any` vermeidet diese Komplikation.

Es ist möglich, Nullwerte an den Superkonstruktor von StoreMock zu übergeben. Dies beschränkt jedoch die Anzahl der Assertions, die zum Testen der Klasse verwendet werden können.

Die Komponente, die in diesem Beispiel verwendet wird, wird nur als Kontext dafür verwendet, wie Store als Testobjekt in das Test-Setup eingefügt werden soll.

Examples

Beobachter Mock

```
class ObserverMock implements Observer<any> {
  closed?: boolean = false; // inherited from Observer
  nextVal: any = ''; // variable I made up

  constructor() {}

  next = (value: any): void => { this.nextVal = value; };
  error = (err: any): void => { console.error(err); };
  complete = (): void => { this.closed = true; }
}

let actionReducer$: ObserverMock = new ObserverMock();
let action$: ObserverMock = new ObserverMock();
let obs$: Observable<any> = new Observable<any>();

class StoreMock extends Store<any> {
  constructor() {
    super(action$, actionReducer$, obs$);
  }
}

describe('Component:Typeahead', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [...],
      declarations: [Typeahead],
      providers: [
        {provide: Store, useClass: StoreMock} // NOTICE useClass instead of useValue
      ]
    }).compileComponents();
  });
});
```

Komponententest für Komponente mit Scheinspeicher

Dies ist eine Testeinheit einer Komponente , die *Store* als Abhängigkeit aufweist. Hier erstellen wir eine neue Klasse namens *MockStore* , die in unsere Komponente anstelle des üblichen Speichers *eingefügt* wird.

```
import { Injectable } from '@angular/core';
import { TestBed, async } from '@angular/core/testing';
import { AppComponent } from './app.component';
import { DumbComponentComponent } from './dumb-component/dumb-component.component';
import { SmartComponentComponent } from './smart-component/smart-component.component';
import { mainReducer } from './state-management/reducers/main-reducer';
import { StoreModule } from '@ngrx/store';
import { Store } from '@ngrx/store';
import { Observable } from 'rxjs';

class MockStore {
  public dispatch(obj) {
```

```

    console.log('dispatching from the mock store!')
  }

  public select(obj) {
    console.log('selecting from the mock store!');

    return Observable.of({})
  }
}

describe('AppComponent', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [
        AppComponent,
        SmartComponentComponent,
        DumbComponentComponent,
      ],
      imports: [
        StoreModule.provideStore({mainReducer})
      ],
      providers: [
        {provide: Store, useClass: MockStore}
      ]
    });
  });

  it('should create the app', async(() => {

    let fixture = TestBed.createComponent(AppComponent);
    let app = fixture.debugElement.componentInstance;
    expect(app).toBeTruthy();
  }));
});

```

Gerätetest für das Ausspähen von Komponenten im Laden

Dies ist eine Testeinheit einer Komponente, die *Store* als Abhängigkeit aufweist. Hier können wir einen Store mit dem Standard "Anfangsstatus" verwenden, während er verhindert, dass tatsächlich Aktionen ausgeführt werden, wenn *store.dispatch()* aufgerufen wird.

```

import {TestBed, async} from '@angular/core/testing';
import {AppComponent} from './app.component';
import {DumbComponentComponent} from './dumb-component/dumb-component.component';
import {SmartComponentComponent} from './smart-component/smart-component.component';
import {mainReducer} from './state-management/reducers/main-reducer';
import {StoreModule} from '@ngrx/store';
import {Store} from '@ngrx/store';
import {Observable} from 'rxjs';

describe('AppComponent', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [
        AppComponent,
        SmartComponentComponent,
        DumbComponentComponent,
      ],

```

```

    imports: [
      StoreModule.provideStore({mainReducer})
    ]
  });

});

it('should create the app', async(() => {
  let fixture = TestBed.createComponent(AppComponent);
  let app = fixture.debugElement.componentInstance;

  var mockStore = fixture.debugElement.injector.get(Store);
  var storeSpy = spyOn(mockStore, 'dispatch').and.callFake(function () {
    console.log('dispatching from the spy!');
  });

}));

});

```

Winkel 2 - Scheinbeobachtung (Service + Komponente)

Bedienung

- Ich habe einen Postdienst mit der Methode postRequest erstellt.

```

import {Injectable} from '@angular/core';
import {Http, Headers, Response} from "@angular/http";
import {PostModel} from "../PostModel";
import 'rxjs/add/operator/map';
import {Observable} from "rxjs";

@Injectable()
export class PostService {

  constructor(private _http: Http) {
  }

  postRequest(postModel: PostModel) : Observable<Response> {
    let headers = new Headers();
    headers.append('Content-Type', 'application/json');
    return this._http.post("/postUrl", postModel, {headers})
      .map(res => res.json());
  }
}

```

Komponente

- Ich habe eine Komponente mit result-Parametern und einer postExample-Funktion erstellt, die postService aufrufen.
- Wenn die Nach-Anfrage als Ergebnisparameter erfolgreich war, sollte 'Erfolg' oder 'Nicht bestanden' angegeben werden.

```

import {Component} from '@angular/core';
import {PostService} from "../PostService";
import {PostModel} from "../PostModel";

@Component({
  selector: 'app-post',
  templateUrl: './post.component.html',
  styleUrls: ['./post.component.scss'],
  providers : [PostService]
})
export class PostComponent{

  constructor(private _postService : PostService) {

    let postModel = new PostModel();
    result : string = null;
    postExample(){
      this._postService.postRequest(this.postModel)
        .subscribe(
          () => {
            this.result = 'Success';
          },
          err => this.result = 'Fail'
        )
    }
  }
}

```

Testdienst

- Wenn Sie den Dienst mit http testen möchten, sollten Sie mockBackend verwenden. und injiziere es dazu.
- Sie müssen auch postService injizieren.

```

describe('Test PostService', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [HttpModule],
      providers: [
        PostService,
        MockBackend,
        BaseRequestOptions,
        {
          provide: Http,
          deps: [MockBackend, BaseRequestOptions],
          useFactory: (backend: XHRBackend, defaultOptions: BaseRequestOptions) => {
            return new Http(backend, defaultOptions);
          }
        }
      ]
    });
  });
});

it('sendPostRequest function return Observable', inject([PostService, MockBackend],
(service: PostService, mockBackend: MockBackend) => {
  let mockPostModel = PostModel();

```

```

mockBackend.connections.subscribe((connection: MockConnection) => {
  expect(connection.request.method).toEqual(RequestMethod.Post);
  expect(connection.request.url.indexOf('postUrl')).not.toEqual(-1);
  expect(connection.request.headers.get('Content-Type')).toEqual('application/json');
});

service
  .postRequest(PostModel)
  .subscribe((response) => {
    expect(response).toBeDefined();
  });
}));
});

```

Testkomponente

```

describe('testing post component', () => {
  let component: PostComponent;
  let fixture: ComponentFixture<PostComponent>;

  let mockRouter = {
    navigate: jasmine.createSpy('navigate')
  };

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [PostComponent],
      imports: [RouterTestingModule.withRoutes([], ModalModule.forRoot())],
      providers: [PostService, MockBackend, BaseRequestOptions,
        {provide: Http, deps: [MockBackend, BaseRequestOptions],
          useFactory: (backend: XHRBackend, defaultOptions: BaseRequestOptions) => {
            return new Http(backend, defaultOptions);
          }
        }
      ],
      {provide: Router, useValue: mockRouter}
    ],
    schemas: [CUSTOM_ELEMENTS_SCHEMA]
  }).compileComponents();
}));

beforeEach(() => {
  fixture = TestBed.createComponent(PostComponent);
  component = fixture.componentInstance;
  fixture.detectChanges();
});

it('test postRequest success', inject([PostService, MockBackend], (service: PostService,
mockBackend: MockBackend) => {
  fixturePostComponent = TestBed.createComponent(PostComponent);
  componentPostComponent = fixturePostComponent.componentInstance;
  fixturePostComponent.detectChanges();

  component.postExample();
  let postModel = new PostModel();
  let response = {

```

```

    'message' : 'message',
    'ok'      : true
  };
  mockBackend.connections.subscribe((connection: MockConnection) => {
    postComponent.result = 'Success'
    connection.mockRespond(new Response(
      new ResponseOptions({
        body: response
      })
    ))
  });
  service.postRequest(postModel)
    .subscribe((data) => {
      expect(component.result).toBeDefined();
      expect(PostComponent.result).toEqual('Success');
      expect(data).toEqual(response);
    });
  });
});
});

```

Einfacher Laden

einfach.aktion.ts

```

import { Action } from '@ngrx/store';

export enum simpleActionTpye {
  add = "simpleAction_Add",
  add_Success = "simpleAction_Add_Success"
}

export class simpleAction {
  type: simpleActionTpye
  constructor(public payload: number) { }
}

```

simple.effects.ts

```

import { Effect, Actions } from '@ngrx/effects';
import { Injectable } from '@angular/core';
import { Action } from '@ngrx/store';
import { Observable } from 'rxjs';

import { simpleAction, simpleActionTpye } from './simple.action';

@Injectable()
export class simpleEffects {

  @Effect()
  addAction$: Observable<simpleAction> = this.actions$
    .ofType(simpleActionTpye.add)
    .switchMap((action: simpleAction) => {
      console.log(action);

      return Observable.of({ type: simpleActionTpye.add_Success, payload: action.payload
    })

    // if you have an api use this code

```

```

        // return this.http.post(url).catch().map(res=>{ type: simpleAction.add_Success,
payload:res})
    });
    constructor(private actions$: Actions) { }
}

```

einfach.reducer.ts

```

import { Action, ActionReducer } from '@ngrx/store';

import { simpleAction, simpleActionType } from './simple.action';

export const simpleReducer: ActionReducer<number> = (state: number = 0, action: simpleAction):
number => {
    switch (action.type) {
        case simpleActionType.add_Success:
            console.log(action);
            return state + action.payload;
        default:
            return state;
    }
}

```

store / index.ts

```

import { combineReducers, ActionReducer, Action, StoreModule } from '@ngrx/store';
import { EffectsModule } from '@ngrx/effects';
import { ModuleWithProviders } from '@angular/core';
import { compose } from '@ngrx/core';

import { simpleReducer } from "../simple/simple.reducer";
import { simpleEffects } from "../simple/simple.effects";

export interface IAppState {
    sum: number;
}

// all new reducers should be define here
const reducers = {
    sum: simpleReducer
};

export const store: ModuleWithProviders = StoreModule.forRoot(reducers);
export const effects: ModuleWithProviders[] = [
    EffectsModule.forRoot([simpleEffects])
];

```

app.module.ts

```

import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core';

import { effects, store } from "../Store/index";
import { AppComponent } from './app.component';

@NgModule({

```



```

declarations: [
  AppComponent
],
imports: [
  BrowserModule,
  // store
  store,
  effects
],
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }

```

app.component.ts

```

import { Component } from '@angular/core';

import { Store } from '@ngrx/store';
import { Observable } from 'rxjs';

import { IAppState } from '../Store/index';
import { simpleActionTpye } from '../Store/simple/simple.action';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app';

  constructor(private store: Store<IAppState>) {
    store.select(s => s.sum).subscribe((res) => {
      console.log(res);
    })
    this.store.dispatch({
      type: simpleActionTpye.add,
      payload: 1
    })
    this.store.dispatch({
      type: simpleActionTpye.add,
      payload: 2
    })
    this.store.dispatch({
      type: simpleActionTpye.add,
      payload: 3
    })
  }
}

```

Ergebnis 0 1 3 6

Spott @ ngrx / Store online lesen: <https://riptutorial.com/de/angular2/topic/8038/spott---ngrx---store>

Kapitel 61: Testen einer Angular 2 App

Examples

Installation des Jasmine-Testframeworks

Die gebräuchlichste Methode zum Testen von Angular 2-Apps ist das Jasmine-Test-Framework. Mit Jasmine können Sie Ihren Code im Browser testen.

Installieren

Alles, was Sie brauchen, ist das `jasmine-core` Paket (nicht `jasmine`).

```
npm install jasmine-core --save-dev --save-exact
```

Überprüfen

Um zu überprüfen, ob Jasmine ordnungsgemäß eingerichtet ist, erstellen Sie die Datei `./src/unit-tests.html` mit folgendem Inhalt und öffnen Sie sie im Browser.

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8">
  <title>Ng App Unit Tests</title>
  <link rel="stylesheet" href="../node_modules/jasmine-core/lib/jasmine-core/jasmine.css">
  <script src="../node_modules/jasmine-core/lib/jasmine-core/jasmine.js"></script>
  <script src="../node_modules/jasmine-core/lib/jasmine-core/jasmine-html.js"></script>
  <script src="../node_modules/jasmine-core/lib/jasmine-core/boot.js"></script>
</head>
<body>
  <!-- Unit Testing Chapter #1: Proof of life. -->
  <script>
    it('true is true', function () {
      expect(true).toEqual(true);
    });
  </script>
</body>
</html>
```

Einrichten von Tests mit Gulp, Webpack, Karma und Jasmine

Als Erstes müssen wir Karma anweisen, unsere Tests mithilfe von Webpack zu lesen. Dies ist eine Konfiguration, die wir für die Webpack-Engine festgelegt haben. Hier verwende ich Babel, weil ich meinen Code in ES6 schreibe. Sie können dies für andere Flavours, wie beispielsweise TypeScript, ändern. Oder ich verwende Pug-Vorlagen (früher Jade-Vorlagen).

Trotzdem bleibt die Strategie gleich.

Dies ist also eine Webpack-Konfiguration:

```
const webpack = require("webpack");
let packConfig = {
  entry: {},
  output: {},
  plugins:[
    new webpack.DefinePlugin({
      ENVIRONMENT: JSON.stringify('test')
    })
  ],
  module: {
    loaders: [
      {
        test: /\.js$/,
        exclude:/(node_modules|bower_components)/,
        loader: "babel",
        query:{
          presets:["es2015", "angular2"]
        }
      },
      {
        test: /\.woff2?$|\.ttf$|\.eot$|\.svg$/,
        loader: "file"
      },
      {
        test: /\.scss$/,
        loaders: ["style", "css", "sass"]
      },
      {
        test: /\.pug$/,
        loader: 'pug-html-loader'
      },
    ]
  },
  devtool : 'inline-cheap-source-map'
};
module.exports = packConfig;
```

Und dann brauchen wir eine karma.config.js-Datei, um diese Webpack-Konfiguration zu verwenden:

```
const packConfig = require("../webpack.config.js");
module.exports = function (config) {
  config.set({
    basePath: '',
    frameworks: ['jasmine'],
    exclude:[],
    files: [
      {pattern: './karma.shim.js', watched: false}
    ],

    preprocessors: {
      './karma.shim.js':["webpack"]
    },
    webpack: packConfig,
  });
};
```

```

    webpackServer: {noInfo: true},

    port: 9876,

    colors: true,

    logLevel: config.LOG_INFO,

    browsers: ['PhantomJS'],

    concurrency: Infinity,

    autoWatch: false,
    singleRun: true
  });
};

```

Bis jetzt haben wir Karma angewiesen, Webpack zu verwenden, und wir haben gesagt, dass es mit einer Datei namens **karma.shim.js** beginnen soll . Diese Datei hat die Aufgabe, als Ausgangspunkt für das Webpack zu fungieren. webpack wird diese Datei lesen und den **Import** verwenden und **benötigen** Aussagen alle unsere Abhängigkeiten zu sammeln und unsere Tests durchführen.

Schauen wir uns nun die Datei karma.shim.js an:

```

// Start of ES6 Specific stuff
import "es6-shim";
import "es6-promise";
import "reflect-metadata";
// End of ES6 Specific stuff

import "zone.js/dist/zone";
import "zone.js/dist/long-stack-trace-zone";
import "zone.js/dist/jasmine-patch";
import "zone.js/dist/async-test";
import "zone.js/dist/fake-async-test";
import "zone.js/dist/sync-test";
import "zone.js/dist/proxy-zone";

import 'rxjs/add/operator/map';
import 'rxjs/add/observable/of';

Error.stackTraceLimit = Infinity;

import {TestBed} from "@angular/core/testing";
import { BrowserDynamicTestingModule, platformBrowserDynamicTesting} from "@angular/platform-browser-dynamic/testing";

TestBed.initTestEnvironment (
  BrowserDynamicTestingModule,
  platformBrowserDynamicTesting());

let testContext = require.context('../src/app', true, /\.spec\.js/);
testContext.keys().forEach(testContext);

```

Im Wesentlichen importieren wir **TestBed** aus **Winkelkerntests** und initiieren die Umgebung, da sie für alle unsere Tests nur einmal initiiert werden muss. Dann gehen wir rekursiv durch das

Verzeichnis **src / app** und lesen jede Datei, die mit **.spec.js** endet, und geben sie an `testContext` weiter, damit sie ausgeführt werden.

Normalerweise versuche ich, meine Prüfungen am selben Ort wie die Klasse abzulegen. Der persönliche Geschmack macht es mir einfacher, Abhängigkeiten und Refactor-Tests mit Klassen zu importieren. Wenn Sie Ihre Tests jedoch an einem anderen Ort **ablegen** möchten, beispielsweise im Verzeichnis **src / test**, haben Sie die Chance. Ändern Sie die Zeile vor dem letzten in der Datei `karma.shim.js`.

Perfekt. was ist übrig? ah, die gulp-Aufgabe, die die oben erstellte Datei `karma.config.js` verwendet:

```
gulp.task("karmaTests",function(done){
  var Server = require("karma").Server;
  new Server({
    configFile : "./karma.config.js",
    singleRun: true,
    autoWatch: false
  }, function(result){
    return result ? done(new Error(`Karma failed with error code ${result}`)):done();
  }).start();
});
```

Ich starte den Server jetzt mit der von uns erstellten Konfigurationsdatei und sage, dass er einmal ausgeführt werden soll und nicht auf Änderungen achten muss. Ich finde, dass dies für mich besser geeignet ist, da die Tests nur ausgeführt werden, wenn ich bereit bin, sie auszuführen. Wenn Sie jedoch einen anderen Computer benötigen, wissen Sie, wo Sie Änderungen vornehmen müssen.

Und als letztes Codebeispiel finden Sie hier eine Reihe von Tests für das Angular 2-Tutorial "Tour of Heroes".

```
import {
  TestBed,
  ComponentFixture,
  async
} from "@angular/core/testing";

import {AppComponent} from "../app.component";
import {AppModule} from "../app.module";
import Hero from "../hero/hero";

describe("App Component", function () {

  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [AppModule]
    });

    this.fixture = TestBed.createComponent(AppComponent);
    this.fixture.detectChanges();
  });

  it("Should have a title", async(() => {
    this.fixture.whenStable().then(() => {
```

```

        expect(this.fixture.componentInstance.title).toEqual("Tour of Heros");
    });
    });

it("Should have a hero", async(()=> {
    this.fixture.whenStable().then(()=> {
        expect(this.fixture.componentInstance.selectedHero).toBeNull();
    });
}));

it("Should have an array of heros", async(()=>
    this.fixture.whenStable().then(()=> {
        const cmp = this.fixture.componentInstance;
        expect(cmp.heroes).toBeDefined("component should have a list of heroes");
        expect(cmp.heroes.length).toEqual(10, "heroes list should have 10 members");
        cmp.heroes.map((h, i)=> {
            expect(h instanceof Hero).toBeTruthy(`member ${i} is not a Hero instance.
${h}`)
        });
    }));

it("Should have one list item per hero", async(()=>
    this.fixture.whenStable().then(()=> {
        const ul = this.fixture.nativeElement.querySelector("ul.heroes");
        const li = Array.prototype.slice.call(
            this.fixture.nativeElement.querySelectorAll("ul.heroes>li"));
        const cmp = this.fixture.componentInstance;
        expect(ul).toBeTruthy("There should be an unnumbered list for heroes");
        expect(li.length).toEqual(cmp.heroes.length, "there should be one li for each
hero");
        li.forEach((li, i)=> {
            expect(li.querySelector("span.badge"))
                .toBeTruthy(`hero ${i} has to have a span for id`);
            expect(li.querySelector("span.badge").textContent.trim())
                .toEqual(cmp.heroes[i].id.toString(), `hero ${i} had wrong id displayed`);
            expect(li.textContent)
                .toMatch(cmp.heroes[i].name, `hero ${i} has wrong name displayed`);
        });
    }));

it("should have correct styling of hero items", async(()=>
    this.fixture.whenStable().then(()=> {
        const hero = this.fixture.nativeElement.querySelector("ul.heroes>li");
        const win = hero.ownerDocument.defaultView || hero.ownerDocument.parentWindow;
        const styles = win.getComputedStyle(hero);
        expect(styles["cursor"]).toEqual("pointer", "cursor should be pointer on hero");
        expect(styles["borderRadius"]).toEqual("4px", "borderRadius should be 4px");
    }));

it("should have a click handler for hero items", async(()=>
    this.fixture.whenStable().then(()=>{
        const cmp = this.fixture.componentInstance;
        expect(cmp.onSelect)
            .toBeDefined("should have a click handler for heros");
        expect(this.fixture.nativeElement.querySelector("input.heroName"))
            .toBeNull("should not show the hero details when no hero has been selected");
        expect(this.fixture.nativeElement.querySelector("ul.heroes li.selected"))
            .toBeNull("Should not have any selected heroes at start");

        spyOn(cmp, "onSelect").and.callThrough();
        this.fixture.nativeElement.querySelectorAll("ul.heroes li")[5].click();
    }));

```

```

    expect(cmp.onSelect)
      .toHaveBeenCalledWith(cmp.heroes[5]);
    expect(cmp.selectedHero)
      .toEqual(cmp.heroes[5], "click on hero should change hero");
  })
});
});

```

Bemerkenswert dabei ist, wie wir **vorEach ()** ein Testmodul konfigurieren und die Komponente in test erstellen, und wie wir **detectChanges ()** nennen, so dass **Angular** tatsächlich die Doppelbindung durchläuft.

Beachten Sie, dass jeder Test ein Aufruf von **async ()** ist und immer wartet, bis das **timeStable-**Versprechen aufgelöst wird, bevor das Gerät untersucht wird. Es hat dann Zugriff auf die Komponente über **ComponentInstance** und auf das Element über **nativeElement** .

Es gibt einen Test, bei dem das korrekte Styling geprüft wird. Das Angular-Team demonstriert die Verwendung von Stilen in Komponenten. In unserem Test verwenden wir **getComputedStyle ()** , um zu prüfen, ob die Stile von dem von uns angegebenen stammen. Wir benötigen jedoch das Window-Objekt, und wir erhalten es vom Element, wie Sie im Test sehen können.

HTTP-Dienst testen

Normalerweise rufen Dienste Remote-API an, um Daten abzurufen / zu senden. Bei Unit-Tests sollten jedoch keine Netzwerkanrufe durchgeführt werden. Angular verwendet intern die `XHRBackend` Klasse, um HTTP-Anforderungen `XHRBackend` . Benutzer können dies überschreiben, um das Verhalten zu ändern. Das Angular- `MockBackend` stellt `MockBackend` und `MockConnection` Klassen `MockConnection` , mit denen HTTP-Anforderungen `MockBackend` und `MockConnection` werden können.

`posts.service.ts` Dieser Dienst trifft einen Endpunkt der API, um eine Liste der Beiträge abzurufen.

```

import { Http } from '@angular/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs/rx';

import 'rxjs/add/operator/map';

export interface IPost {
  userId: number;
  id: number;
  title: string;
  body: string;
}

@Injectable()
export class PostsService {
  posts: IPost[];

  private postsUri = 'http://jsonplaceholder.typicode.com/posts';

  constructor(private http: Http) {
  }
}

```

```

get(): Observable<IPost[]> {
  return this.http.get(this.postsUri)
    .map((response) => response.json());
}
}

```

posts.service.spec.ts **Hier werden wir den obigen Dienst testen, indem wir http api-Aufrufe**

posts.service.spec.ts .

```

import { TestBed, inject, fakeAsync } from '@angular/core/testing';
import {
  HttpClientModule,
  XMLHttpRequest,
  ResponseOptions,
  Response,
  RequestMethod
} from '@angular/http';
import {
  MockBackend,
  MockConnection
} from '@angular/http/testing';

import { PostsService } from './posts.service';

describe('PostsService', () => {
  // Mock http response
  const mockResponse = [
    {
      'userId': 1,
      'id': 1,
      'title': 'sunt aut facere repellat provident occaecati excepturi optio reprehenderit',
      'body': 'quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto'
    },
    {
      'userId': 1,
      'id': 2,
      'title': 'qui est esse',
      'body': 'est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui aperiam non debitis possimus qui neque nisi nulla'
    },
    {
      'userId': 1,
      'id': 3,
      'title': 'ea molestias quasi exercitationem repellat qui ipsa sit aut',
      'body': 'et iusto sed quo iure\nvoluptatem occaecati omnis eligendi aut ad\nvoluptatem doloribus vel accusantium quis pariatur\nmolestiae porro eius odio et labore et velit aut'
    },
    {
      'userId': 1,
      'id': 4,
      'title': 'eum et est occaecati',
      'body': 'ullam et saepe reiciendis voluptatem adipisci\nsit amet autem assumenda provident rerum culpa\nquis hic commodi nesciunt rem tenetur doloremque ipsam iure\nquis sunt voluptatem rerum illo velit'
    }
  ]
}

```



```

];

beforeEach(() => {
  TestBed.configureTestingModule({
    imports: [HttpModule],
    providers: [
      {
        provide: XHRBackend,
        // This provides mocked XHR backend
        useClass: MockBackend
      },
      PostsService
    ]
  });
});

it('should return posts retrieved from Api', fakeAsync(
  inject([XHRBackend, PostsService],
    (mockBackend, postsService) => {
      mockBackend.connections.subscribe(
        (connection: MockConnection) => {
          // Assert that service has requested correct url with expected method
          expect(connection.request.method).toBe(RequestMethod.Get);

expect(connection.request.url).toBe('http://jsonplaceholder.typicode.com/posts');
          // Send mock response
          connection.mockRespond(new Response(new ResponseOptions({
            body: mockResponse
          })));
        });

      postsService.get()
        .subscribe((posts) => {
          expect(posts).toBe(mockResponse);
        });

    }));
});

```

Winkelkomponenten testen - Basic

Der Komponentencode ist wie folgt angegeben.

```

import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: '<h1>{{title}}</h1>'
})
export class MyAppComponent {
  title = 'welcome';
}

```

Für das Testen von Winkeln bietet angle seine Test-Dienstprogramme zusammen mit dem Test-Framework, das beim Schreiben des guten Testfalls in Winkel hilft. Angular-Dienstprogramme können aus `@angular/core/testing` importiert `@angular/core/testing`

```

import { ComponentFixture, TestBed } from '@angular/core/testing';
import { MyAppComponent } from './banner-inline.component';

describe('Tests for MyAppComponent', () => {

  let fixture: ComponentFixture<MyAppComponent>;
  let comp: MyAppComponent;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [
        MyAppComponent
      ]
    });
  });

  beforeEach(() => {

    fixture = TestBed.createComponent(MyAppComponent);
    comp = fixture.componentInstance;

  });

  it('should create the MyAppComponent', () => {

    expect(comp).toBeTruthy();

  });

});

```

Im obigen Beispiel gibt es nur einen Testfall, der den Testfall für das Vorhandensein von Komponenten erklärt. Im obigen Beispiel werden Hilfsprogramme für die Winkelprüfung wie `TestBed` und `ComponentFixture` verwendet.

`TestBed` wird zum Erstellen des Winkelprüfmoduls verwendet. Dieses Modul wird mit der `configureTestingModule` Methode `configureTestingModule`, um die Modul Umgebung für die Klasse zu erstellen, die wir testen möchten. Testmodul, das vor der Ausführung jedes Testfalls konfiguriert werden muss. Deshalb konfigurieren wir das `beforeEach` in der Funktion `beforeEach`.

`createComponent` Methode von `TestBed` wird zum Erstellen der Instanz der zu `createComponent` Komponente verwendet. `createComponent` das `ComponentFixture`. Das Fixture bietet Zugriff auf die Komponenteninstanz selbst.

Testen einer Angular 2 App online lesen: <https://riptutorial.com/de/angular2/topic/2329/testen-einer-angular-2-app>

Kapitel 62: Umgehen Desinfektion für vertrauenswürdige Werte

Parameter

Params	Einzelheiten
Wähler	Tag-Name, auf den Sie Ihre Komponente verweisen, in der HTML-Datei
Vorlage (templateUrl)	Eine Zeichenfolge, die html darstellt und an der Stelle des Tags <code><selector></code> eingefügt wird. <code>templateUrl</code> ist ein Pfad zu einer HTML-Datei mit demselben Verhalten
Rohre	ein Array von Pipes, die von dieser Komponente verwendet werden.

Bemerkungen

SUPER WICHTIG!

DIE UNTERSUCHUNG DER SANITIERUNG VERLETZT SIE AUF RISIKO VON XSS (Cross-Site Scripting) und anderen Angriffs-Vektoren. BITTE STELLEN SIE SICHER, DASS SIE VERTRAUEN, WAS SIE 100% ERHALTEN

Wenn Sie Pipes verwenden, müssen Sie nur Attributwerte wie folgt ändern:

```
<tag [attribute]="expression or variable reference | pipeName">
```

Sie sind nicht in der Lage, Rohre auf diese Weise zu verwenden:

```
<tag attribute="expression or variable reference | pipeName">
```

oder so

```
<tag attribute={{expression or variable reference | pipeName}}>
```

Examples

Bypass-Desinfektion mit Rohren (zur Wiederverwendung von Code)

Das Projekt wird im Anschluss an die Struktur von der Angular2 Kurzanleitung [hier](#) .

```
RootOfProject
|
+-- app
|   |-- app.component.ts
|   |-- main.ts
|   |-- pipeUser.component.ts
|   \-- sanitize.pipe.ts
|
|-- index.html
|-- main.html
|-- pipe.html
```

main.ts

```
import { bootstrap } from '@angular/platform-browser-dynamic';
import { AppComponent } from './app.component';

bootstrap(AppComponent);
```

Dies findet die Datei index.html im Stammverzeichnis des Projekts und baut darauf auf.

app.component.ts

```
import { Component } from '@angular/core';
import { PipeUserComponent } from './pipeUser.component';

@Component({
  selector: 'main-app',
  templateUrl: 'main.html',
  directives: [PipeUserComponent]
})

export class AppComponent { }
```

Dies ist die Komponente der obersten Ebene, die andere verwendete Komponenten gruppiert.

pipeUser.component.ts

```
import { Component } from '@angular/core';
import { IgnoreSanitize } from "./sanitize.pipe";

@Component({
  selector: 'pipe-example',
  templateUrl: "pipe.html",
  pipes: [IgnoreSanitize]
})

export class PipeUserComponent{
  constructor () { }
  unsafeValue: string = "unsafe/picUrl?id=";
  docNum: string;

  getUrl(input: string): any {
    if(input !== undefined) {
```

```

        return this.unsafeValue.concat(input);
        // returns : "unsafe/picUrl?id=input"
    } else {
        return "fallback/to/something";
    }
}
}
}

```

Diese Komponente bietet die Ansicht, mit der die Pipe arbeiten kann.

sanitize.pipe.ts

```

import { Pipe, PipeTransform } from '@angular/core';
import { DomSanitizationService } from '@angular/platform-browser';

@Pipe({
  name: 'sanitaryPipe'
})
export class IgnoreSanitize implements PipeTransform {

  constructor(private sanitizer: DomSanitizationService){}

  transform(input: string) : any {
    return this.sanitizer.bypassSecurityTrustUrl(input);
  }
}

```

Dies ist die Logik, die die Pipe-Formate beschreibt.

index.html

```

<head>
  Stuff goes here...
</head>
<body>
  <main-app>
    main.html will load inside here.
  </main-app>
</body>

```

main.html

```

<othertags>
</othertags>

<pipe-example>
  pipe.html will load inside here.
</pipe-example>

<moretags>
</moretags>

```

pipe.html

```
<img [src]="getUrl('1234') | sanitaryPipe">
<embed [src]="getUrl() | sanitaryPipe">
```

Wenn Sie die HTML-Datei während der Ausführung der App inspizieren würden, würden Sie Folgendes sehen:

```
<head>
  Stuff goes here...
</head>

<body>

  <othertags>
  </othertags>

  <img [src]="getUrl('1234') | sanitaryPipe">
  <embed [src]="getUrl() | sanitaryPipe">

  <moretags>
  </moretags>

</body>
```

Umgehen Desinfektion für vertrauenswürdige Werte online lesen:

<https://riptutorial.com/de/angular2/topic/5942/umgehen-desinfektion-fur-vertrauenswurdige-werte>

Kapitel 63: Verwenden Sie in Angular 2 native Webkomponenten

Bemerkungen

Wenn Sie eine Webkomponente in Ihrer Angular 2-Vorlage verwenden, versucht Angular, eine Komponente mit einem Selektor zu finden, der mit dem benutzerdefinierten Tag der Webkomponente übereinstimmt - was natürlich keinen Fehler verursachen kann und wird.

Die Lösung besteht darin, ein "benutzerdefiniertes Elementeschema" in das Modul zu importieren, das die Komponente enthält. Dies bewirkt, dass angle alle benutzerdefinierten Tags akzeptiert, die nicht mit den Auswahlelementen für ng-Komponenten übereinstimmen.

Examples

Fügen Sie dem Modul benutzerdefinierte Elemente hinzu

```
import { NgModule, CUSTOM_ELEMENTS_SCHEMA } from '@angular/core';
import { CommonModule } from '@angular/common';
import { AboutComponent } from './about.component';

@NgModule({
  imports: [ CommonModule ],
  declarations: [ AboutComponent ],
  exports: [ AboutComponent ],
  schemas: [ CUSTOM_ELEMENTS_SCHEMA ]
})

export class AboutModule { }
```

Verwenden Sie Ihre Webkomponente in einer Vorlage

```
import { Component } from '@angular/core';

@Component({
  selector: 'myapp-about',
  template: `<my-webcomponent></my-webcomponent>`
})
export class AboutComponent { }
```

Verwenden Sie in Angular 2 native Webkomponenten online lesen:

<https://riptutorial.com/de/angular2/topic/7414/verwenden-sie-in-angular-2-native-webkomponenten>

Kapitel 64: Verwenden von Drittanbieter-Bibliotheken wie jQuery in Angular 2

Einführung

Beim Erstellen von Anwendungen mit Angular 2.x müssen unter Umständen Bibliotheken von Drittanbietern wie jQuery, Google Analytics, Chat-Integration-JavaScript-APIs usw. verwendet werden.

Examples

Konfiguration mit Winkelkli

NPM

Wenn eine externe Bibliothek wie `jQuery` mit NPM installiert wird

```
npm install --save jquery
```

Fügen Sie einen `angular-cli.json` in die `angular-cli.json`

```
"scripts": [  
  "../node_modules/jquery/dist/jquery.js"  
]
```

Assets-Ordner

Sie können die Bibliotheksdatei auch in Ihrem `assets/js` Verzeichnis speichern und in die Datei

```
assets/js angular-cli.json
```

```
"scripts": [  
  "assets/js/jquery.js"  
]
```

Hinweis

Speichern Sie Ihre Hauptbibliothek- `jQuery` und ihre Abhängigkeiten wie das `jQuery-cycle-plugin` im Assets-Verzeichnis, und fügen Sie beide in die `angular-cli.json`.

JQuery in Angular 2.x-Komponenten verwenden

Um `jQuery` in Ihren Angular 2.x-Komponenten zu verwenden, deklarieren Sie oben eine globale Variable

Wenn Sie `$` für jQuery verwenden

```
declare var $: any;
```

Wenn Sie `jQuery` für jQuery verwenden

```
declare var jQuery: any
```

Dies ermöglicht die Verwendung von `$` oder `jQuery` in Ihrer Angular 2.x-Komponente.

Verwenden von Drittanbieter-Bibliotheken wie jQuery in Angular 2 online lesen:

<https://riptutorial.com/de/angular2/topic/9285/verwenden-von-drittanbieter-bibliotheken-wie-jquery-in-angular-2>

Kapitel 65: Verwendung von ngfor

Einführung

Die `ngFor` Direktive wird von Angular2 verwendet, um eine Vorlage einmal für jedes Element in einem iterierbaren Objekt zu instantiieren. Diese Anweisung bindet das Iterable an das DOM. Wenn sich der Inhalt der Iteration ändert, wird auch der Inhalt des DOM geändert.

Examples

Beispiel für ungeordnete Listen

```
<ul>
  <li *ngFor="let item of items">{{item.name}}</li>
</ul>
```

Komplexeres Template-Beispiel

```
<div *ngFor="let item of items">
  <p>{{item.name}}</p>
  <p>{{item.price}}</p>
  <p>{{item.description}}</p>
</div>
```

Beispiel für aktuelles Zusammenspiel verfolgen

```
<div *ngFor="let item of items; let i = index">
  <p>Item number: {{i}}</p>
</div>
```

In diesem Fall nehme ich den Wert von `index`, der die aktuelle Schleifeniteration darstellt.

Angular2-Alias exportierte Werte

Angular2 bietet mehrere exportierte Werte, die mit lokalen Variablen als Alias bezeichnet werden können. Diese sind:

- `Index`
- `zuerst`
- `zuletzt`
- `sogar`
- `ungerade`

Mit Ausnahme des `index` die anderen einen `Boolean` Wert an. Wie im vorherigen Beispiel mit dem `Index` kann jeder dieser exportierten Werte verwendet werden:

```
<div *ngFor="let item of items; let firstItem = first; let lastItem = last">
  <p *ngIf="firstItem">I am the first item and I am gonna be showed</p>
  <p *ngIf="firstItem">I am not the first item and I will not show up :(</p>
  <p *ngIf="lastItem">But I'm gonna be showed as I am the last item :)</p>
</div>
```

* ngFür mit Rohr

```
import { Pipe, PipeTransform } from '@angular/core';
@Pipe({
  name: 'even'
})

export class EvenPipe implements PipeTransform {
  transform(value: string): string {
    if(value && value %2 === 0){
      return value;
    }
  }
}

@Component({
  selector: 'example-component',
  template: '<div>
    <div *ngFor="let number of numbers | even">
      {{number}}
    </div>
  </div>'
})

export class exampleComponent {
  let numbers : List<number> = Array.apply(null, {length: 10}).map(Number.call, Number)
}
```

Verwendung von ngfor online lesen: <https://riptutorial.com/de/angular2/topic/8051/verwendung-von-ngfor>

Kapitel 66: Vorlagen

Einführung

Vorlagen sind den Vorlagen in Angular 1 sehr ähnlich, obwohl es viele kleine syntaktische Änderungen gibt, die den Vorgang klarer machen.

Examples

Angular 2 Vorlagen

Eine einfache Vorlage

Beginnen wir mit einer sehr einfachen Vorlage, die unseren Namen und unsere Lieblingssache zeigt:

```
<div>
  Hello my name is {{name}} and I like {{thing}} quite a lot.
</div>
```

{}: RENDERING

Um einen Wert zu rendern, können wir die standardmäßige doppelt geschweifte Syntax verwenden:

```
My name is {{name}}
```

Pipes, zuvor als "Filter" bekannt, wandeln einen Wert in einen neuen Wert um, z. B. das Lokalisieren einer Zeichenfolge oder das Konvertieren eines Gleitkommawerts in eine Währungsrepräsentation:

[]: VERBINDENDE EIGENSCHAFTEN

Um eine Variable aufzulösen und an eine Komponente zu binden, verwenden Sie die Syntax []. Wenn sich `this.currentVolume` in unserer Komponente befindet, geben wir dies an unsere Komponente weiter und die Werte bleiben synchron:

```
<video-control [volume]="currentVolume"></video-control>
(): HANDLING EVENTS
```

() : BEARBEITEN VON EREIGNISSEN Um auf ein Ereignis auf einer Komponente zu hören, verwenden wir die () -Syntax

```
<my-component (click)="onClick($event)"></my-component>
```

[()]: ZWEI-WEGE-DATENBINDUNG

Verwenden Sie die Syntax `[(())]`, um die Benutzereingaben und andere Ereignisse auf dem neuesten Stand zu halten. Betrachten Sie es als eine Kombination aus der Verarbeitung eines Ereignisses und der Bindung einer Eigenschaft:

`<input [(ngModel)] = "myName">` Der `this.myName`-Wert Ihrer Komponente bleibt mit dem Eingabewert synchron.

*** : DIE ASTERISK**

Gibt an, dass diese Direktive diese Komponente als Vorlage behandelt und nicht so wie sie ist gezeichnet wird. Zum Beispiel: `ngFor` nimmt our und stempelt es für jedes Element in Items ab. Es wird jedoch nie unser ursprüngliches Element dargestellt, da es sich um eine Vorlage handelt:

```
<my-component *ngFor="#item of items">  
</my-component>
```

Andere ähnliche Anweisungen, die mit Vorlagen arbeiten, als gerenderte Komponenten sind `* ngIf` und `* ngSwitch`.

Vorlagen online lesen: <https://riptutorial.com/de/angular2/topic/9471/vorlagen>

Kapitel 67: Wie benutze ich ngIf?

Einführung

* **NgIf** : **Entfernt** oder erstellt einen Teil des DOM-Baums je nach Ausdrucksauswertung. Es handelt sich um eine strukturelle Richtlinie, und strukturelle Richtlinien ändern das Layout des DOM durch Hinzufügen, Ersetzen und Entfernen seiner Elemente.

Syntax

- `<div *ngIf = "false"> test </div> <! - wird zu false -> ausgewertet`
- `<div *ngIf = "undefined"> test </div> <! - wird zu false -> ausgewertet`
- `<div *ngIf = "null"> test </div> <! - wird zu false -> ausgewertet`
- `<div *ngIf = "0"> test </div> <! - wird zu false -> ausgewertet`
- `<div *ngIf = "NaN"> test </div> <! - wird zu false -> ausgewertet`
- `<div *ngIf = ""> test </div> <! - wird zu false -> ausgewertet`
- Alle anderen Werte werden als wahr ausgewertet.

Examples

Zeigt eine Lademeldung an

Wenn unsere Komponente nicht bereit ist und auf Daten vom Server wartet, können Sie den Loader mit *ngIf hinzufügen. **Schritte:**

Zuerst deklarieren Sie einen Boolean:

```
loading: boolean = false;
```

Als nächstes fügen Sie in Ihrer Komponente einen Lebenszyklus-Hook namens `ngOnInit`

```
ngOnInit() {  
  this.loading = true;  
}
```

und nachdem Sie vollständige Daten vom Server erhalten haben, laden Sie boolean auf false.

```
this.loading=false;
```

In Ihrem HTML - Template verwenden *ngIf mit der `loading` Eigenschaft:

```
<div *ngIf="loading" class="progress">  
  <div class="progress-bar info" style="width: 125%;"></div>  
</div>
```

Warnmeldung zu einer Bedingung anzeigen

```
<p class="alert alert-success" *ngIf="names.length > 2">Currently there are more than 2 names!</p>
```

Eine Funktion am Anfang oder Ende der * ngFor-Schleife ausführen Mit * ngIf

NgFor stellt einige Werte bereit, die für lokale Variablen einen Alias haben können

- **index** - (variable) Position des aktuellen Elements im iterierbaren Element, beginnend bei 0
- **first** - (boolean) true, wenn das aktuelle Element das erste Element im iterierbaren Element ist
- **last** - (boolean) true, wenn das aktuelle Element das letzte Element im iterierbaren Element ist
- **even** - (boolean) true, wenn der aktuelle Index eine gerade Zahl ist
- **odd** - (boolean) true, wenn der aktuelle Index eine ungerade Zahl ist

```
<div *ngFor="let note of csvdata; let i=index; let lastcall=last">
  <h3>{{i}}</h3> <!-- to show index position
  <h3>{{note}}</h3>
  <span *ngIf="lastcall">{{anyfunction()}} </span><!-- this lastcall boolean value will be
true only if this is last in loop
  // anyfunction() will run at the end of loop same way we can do at start
</div>
```

Verwenden Sie * ngIf mit * ngFor

Während Sie *ngIf und *ngFor im selben div verwenden dürfen (es wird ein Fehler in der Laufzeit *ngIf), können Sie die *ngIf in der *ngFor , um das gewünschte Verhalten zu erhalten.

Beispiel 1: Allgemeine Syntax

```
<div *ngFor="let item of items; let i = index">
  <div *ngIf="<your condition here>"

  <!-- Execute code here if statement true -->

</div>
</div>
```

Beispiel 2: Anzeigeelemente mit geradem Index

```
<div *ngFor="let item of items; let i = index">
  <div *ngIf="i % 2 == 0">
    {{ item }}
  </div>
</div>
```

Der Nachteil ist, dass ein zusätzliches äußeres div Element hinzugefügt werden muss.

Beachten Sie **diesen Anwendungsfall**, bei dem ein `div` Element iteriert werden muss (mithilfe von `* ngFor`). Außerdem wird geprüft, ob das Element entfernt werden muss oder nicht (mit `* ngIf`). Das Hinzufügen eines zusätzlichen `div` wird jedoch nicht bevorzugt. In diesem Fall können Sie das `template` Tag für die `* ngFor` verwenden:

```
<template ngFor let-item [ngForOf]="items">
  <div *ngIf="item.price > 100">
    </div>
</template>
```

Auf diese Weise ist das Hinzufügen eines zusätzlichen äußeren `div` nicht erforderlich. Außerdem wird das Element `<template>` nicht zum DOM hinzugefügt. Die einzigen Elemente, die in dem DOM aus dem obigen Beispiel hinzugefügt werden, sind die iterierten `div` Elemente.

Hinweis: In Angular v4 ist `<template>` zugunsten von `<ng-template>` veraltet und wird in v5 entfernt. In Angular v2.x-Versionen ist `<template>` noch gültig.

Wie benutze ich `ngif`? online lesen: <https://riptutorial.com/de/angular2/topic/8346/wie-benutze-ich-ngif->

Kapitel 68: Winkel 2 - Winkelmesser

Examples

Testen der Navbar-Routen mit dem Winkelmesser

Zuerst können Sie grundlegende Navbar.html mit 3 Optionen erstellen. (Startseite, Liste, Erstellen)

```
<nav class="navbar navbar-default" role="navigation">
<ul class="nav navbar-nav">
  <li>
    <a id="home-navbar" routerLink="/home">Home</a>
  </li>
  <li>
    <a id="list-navbar" routerLink="/create" >List</a>
  </li>
  <li>
    <a id="create-navbar" routerLink="/create">Create</a>
  </li>
</ul>
```

Zweitens können Sie navbar.e2e-spec.ts erstellen

```
describe('Navbar', () => {

  beforeEach(() => {
    browser.get('home'); // before each test navigate to home page.
  });

  it('testing Navbar', () => {
    browser.sleep(2000).then(function() {
      checkNavbarTexts();
      navigateToListPage();
    });
  });

  function checkNavbarTexts() {
    element(by.id('home-navbar')).getText().then(function(text) { // Promise
      expect(text).toEqual('Home');
    });

    element(by.id('list-navbar')).getText().then(function(text) { // Promise
      expect(text).toEqual('List');
    });

    element(by.id('create-navbar')).getText().then(function(text) { // Promise
      expect(text).toEqual('Create');
    });
  }

  function navigateToListPage() {
    element(by.id('list-home')).click().then(function() { // first find list-home a tag and
    than click
      browser.sleep(2000).then(function() {
        browser.getCurrentUrl().then(function(actualUrl) { // promise
```

```

        expect(actualUrl.indexOf('list') !== -1).toBeTruthy(); // check the current url is
list
        });
    });

});
}
});

```

Angular2-Winkelmesser - Installation

Führen Sie die folgenden Befehle bei cmd aus

- `npm install -g protractor`
- `webdriver-manager update`
- `webdriver-manager start`

**** Erstellen Sie die Datei `protractor.conf.js` im Haupt-App-Stammverzeichnis.**

Sehr wichtig, `useAllAngular2AppRoots: true` zu deklarieren

```

const config = {
  baseUrl: 'http://localhost:3000/',

  specs: [
    './dev/**/*.e2e-spec.js'
  ],

  exclude: [],
  framework: 'jasmine',

  jasmineNodeOpts: {
    showColors: true,
    isVerbose: false,
    includeStackTrace: false
  },

  directConnect: true,

  capabilities: {
    browserName: 'chrome',
    shardTestFiles: false,
    chromeOptions: {
      'args': ['--disable-web-security ', '--no-sandbox', 'disable-extensions', 'start-maximized', 'enable-crash-reporter-for-testing']
    }
  },

  onPrepare: function() {
    const SpecReporter = require('jasmine-spec-reporter');
    // add jasmine spec reporter
    jasmine.getEnv().addReporter(new SpecReporter({ displayStackTrace: true }));

    browser.ignoreSynchronization = false;
  },
  useAllAngular2AppRoots: true
};

```

```
if (process.env.TRAVIS) {
  config.capabilities = {
    browserName: 'firefox'
  };
}

exports.config = config;
```

Erstellen Sie einen grundlegenden Test im Verzeichnis dev.

```
describe('basic test', () => {

  beforeEach(() => {
    browser.get('http://google.com');
  });

  it('testing basic test', () => {
    browser.sleep(2000).then(function() {
      browser.getCurrentUrl().then(function(actualUrl) {
        expect(actualUrl.indexOf('google') !== -1).toBeTruthy();
      });
    });
  });
});
```

in cmd ausführen

```
protractor conf.js
```

Winkel 2 - Winkelmesser online lesen: <https://riptutorial.com/de/angular2/topic/8900/winkel-2---winkelmesser>

Kapitel 69: Winkel-Cli-Testabdeckung

Einführung

Testabdeckung ist eine Technik, die bestimmt, ob unsere Testfälle tatsächlich den Anwendungscode abdecken und wie viel Code beim Ausführen dieser Testfälle ausgeübt wird.

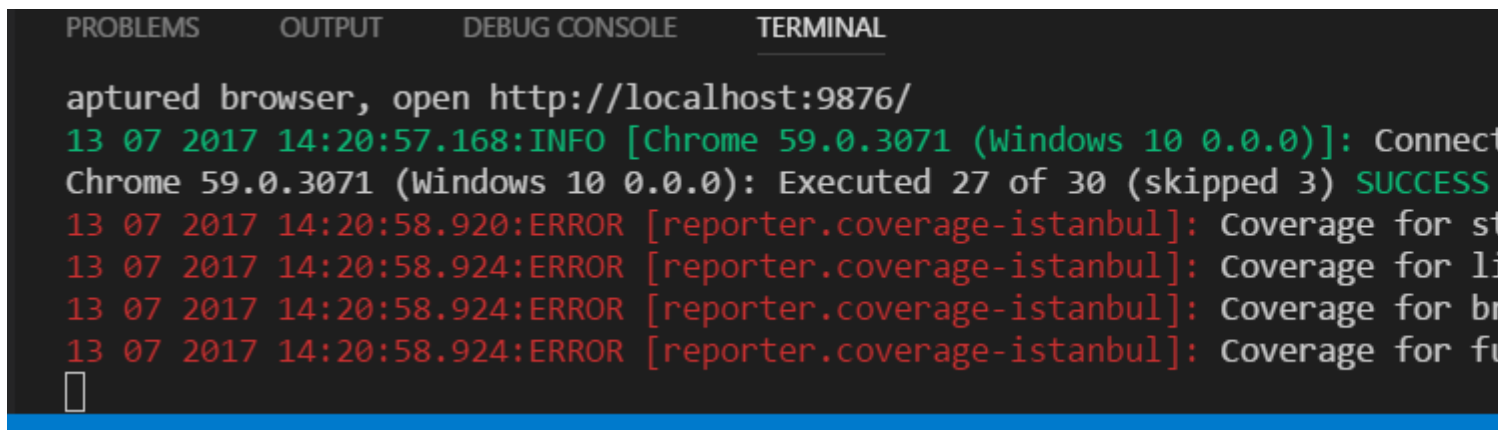
Angular CLI verfügt über eine integrierte Code-Coverage-Funktion mit einem einfachen Befehl `ng test --cc`

Examples

Eine einfache Winkelüberprüfung für den Befehlstest

Wenn Sie allgemeine Testabdeckungsstatistiken als in Angular CLI sehen möchten, geben Sie einfach den Befehl unten ein und sehen Sie unten im Eingabeaufforderungsfenster die Ergebnisse.

```
ng test --cc // or --code-coverage
```



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
aptured browser, open http://localhost:9876/
13 07 2017 14:20:57.168:INFO [Chrome 59.0.3071 (Windows 10 0.0.0)]: Connect
Chrome 59.0.3071 (Windows 10 0.0.0): Executed 27 of 30 (skipped 3) SUCCESS
13 07 2017 14:20:58.920:ERROR [reporter.coverage-istanbul]: Coverage for st
13 07 2017 14:20:58.924:ERROR [reporter.coverage-istanbul]: Coverage for li
13 07 2017 14:20:58.924:ERROR [reporter.coverage-istanbul]: Coverage for br
13 07 2017 14:20:58.924:ERROR [reporter.coverage-istanbul]: Coverage for fu
```

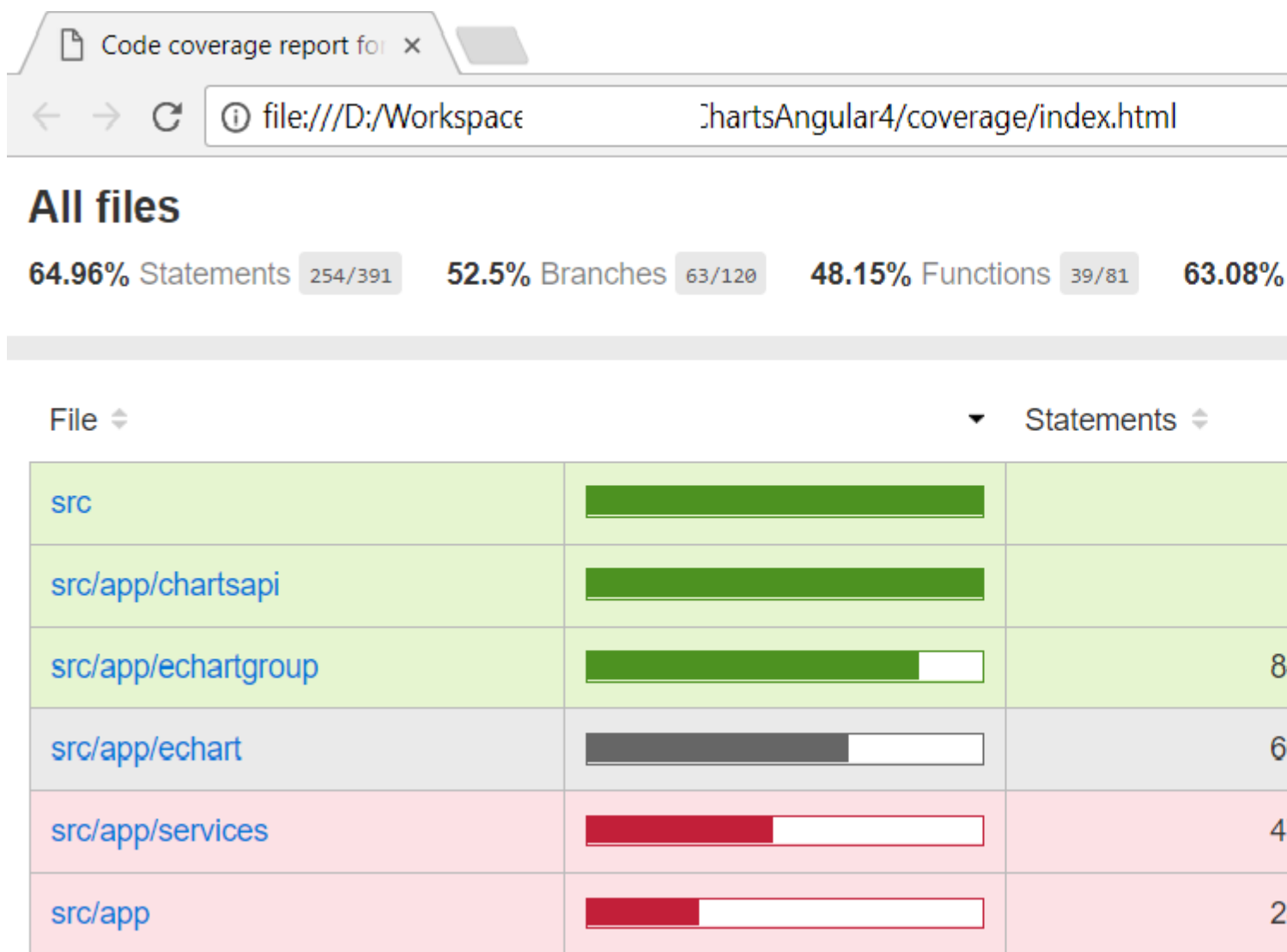
Detaillierte grafische Berichterstattung zur Testabdeckung für einzelne Komponenten

Wenn Sie die individuelle Testabdeckung der Komponente sehen möchten, befolgen Sie diese Schritte.

1. `npm install --save-dev karma-teamcity-reporter`
2. Add `require('karma-teamcity-reporter')` to list of plugins in `karma.conf.js`
3. `ng test --code-coverage --reporters=teamcity,coverage-istanbul`

Beachten Sie, dass die Liste der Reporter durch Kommas getrennt ist, da wir einen neuen Reporter, Teamcity, hinzugefügt haben.

nachdem Sie diesen Befehl ausführen können Sie den Ordner sehen `coverage` in Ihrem Verzeichnis und öffnen `index.html` für eine grafische Darstellung der Testabdeckung.



Sie können den Abdeckungsschwellenwert, den Sie erreichen möchten, auch in `karma.conf.js` wie `karma.conf.js`.

```
coverageIstanbulReporter: {
  reports: ['html', 'lcovonly'],
  fixWebpackSourcePaths: true,
  thresholds: {
    statements: 90,
    lines: 90,
    branches: 90,
    functions: 90
  }
},
```

Winkel-Cli-Testabdeckung online lesen: <https://riptutorial.com/de/angular2/topic/10764/winkel-cli-testabdeckung>

Kapitel 70: Zone.js

Examples

Verweis auf NgZone bekommen

`NgZone` Referenz kann über die Dependency Injection (DI) `NgZone` werden.

meine.komponente.ts

```
import { Component, NgOnInit, NgZone } from '@angular/core';

@Component({...})
export class Mycomponent implements NgOnInit {
  constructor(private _ngZone: NgZone) { }
  ngOnInit() {
    this._ngZone.runOutsideAngular(() => {
      // Do something outside Angular so it won't get noticed
    });
  }
}
```

Verwenden Sie NgZone, um mehrere HTTP-Anforderungen auszuführen, bevor Sie die Daten anzeigen

`runOutsideAngular` kann verwendet werden, um Code außerhalb von Angular 2 auszuführen, sodass die Änderungserkennung nicht unnötig ausgelöst wird. Dies kann verwendet werden, um beispielsweise mehrere HTTP-Anforderungen auszuführen, um alle Daten vor dem Rendern abzurufen. Um Code in Angular 2 erneut auszuführen, kann die `run` Methode von `NgZone` verwendet werden.

meine.komponente.ts

```
import { Component, OnInit, NgZone } from '@angular/core';
import { Http } from '@angular/http';

@Component({...})
export class Mycomponent implements OnInit {
  private data: any[];
  constructor(private http: Http, private _ngZone: NgZone) { }
  ngOnInit() {
    this._ngZone.runOutsideAngular(() => {
      this.http.get('resource1').subscribe((data1:any) => {
        // First response came back, so its data can be used in consecutive request
        this.http.get(`resource2?id=${data1['id']}`).subscribe((data2:any) => {
          this.http.get(`resource3?id1=${data1['id']}&id2=${data2}`).subscribe((data3:any) =>
          {
            this._ngZone.run(() => {
              this.data = [data1, data2, data3];
            });
          });
        });
      });
    });
  }
}
```

```
    });  
  });  
}  
}
```

Zone.js online lesen: <https://riptutorial.com/de/angular2/topic/4184/zone-js>

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Angular 2	acdcjunior , Alexander Ciesielski , beagleknight , Bean0341 , Bhoomi Bhalani , BogdanC , briantylor , cDecker32 , Christopher Moore , Community , daniellmb , drbishop , echonax , Ekin Yücel , elliott-j , etayluz , ettanany , Everettss , H. Pauwelyn , Harry , He11ion , Janco Boscan , Jim , Kaspars Bergs , Logan H , Madhu Ranjan , michaelbahr , Michal Pietraszko , Mihai , nick , Nicolas Irisarri , Peter , QoP , rickysullivan , Shahzad , spike , theblindprophet , user6939352
2	Aktualisierungen	kEpEx
3	Angular - ForLoop	aholtry , Anil Singh , Berseker59 , gerl , Johan Van de Merwe , ob1 , Pujan Srivastava , Stephen Leppik , Yoav Schniederman
4	Angular 2 Änderungserkennung und manuelle Auslösung	Yoav Schniederman
5	Angular 2 datengesteuerte Formulare	MatWaligora , ThunderRoid
6	Angular 2 Forms Update	Amit kumar , Anil Singh , Christopher Taylor , Highmastdon , Johan Van de Merwe , K3v1n , Manmeet Gill , mayur , Norsk , Sachin S , victoroniibukun , vijaykumar , Yoav Schniederman
7	Angular RXJS Subjects und Observables mit API-Anforderungen	daniellmb , Maciej Treder , Ronald Zarīts , Sam Storie , Sébastien Temprado , willydee
8	Angular2 Animationen	Yoav Schniederman
9	Angular2 Benutzerdefinierte Validierungen	Arnold Wiersma , Norsk , Yoav Schniederman
10	Angular2 CanActivate	Companjo , Yoav Schniederman
11	Angular2 Datenbindung	Yoav Schniederman
12	Angular2 Eingang () Ausgang ()	Kaloyan , Yoav Schniederman

13	Angular2 In Memory-Web-API	Jaime Still
14	Angular2 mit Webpack	luukgruijs
15	Angular2 stellt App vor dem Bootstrap externe Daten bereit	Ajey
16	Angular-cli	BogdanC , Yoav Schniederman
17	Angulares Materialdesign	Ketan Akbari , Shailesh Ladumor
18	Animation	Gaurav Mukherjee , Nate May
19	AOT-Compilierung mit Angular 2	Anil Singh , Eric Jimenez , Harry , Robin Dijkhof
20	Attributanweisungen, um den Wert von Eigenschaften auf dem Hostknoten mithilfe des @HostBinding-Dekors zu beeinflussen	Max Karpovets
21	Beispiel für Routen wie / route / subroute für statische URLs	Yoav Schniederman
22	Beispiele für erweiterte Komponenten	borislemke , smnbbvr
23	benutzerdefinierte ngx-bootstrap datepicker + input	Yoav Schniederman
24	Bootstrap Leeres Modul in Winkel 2	AryanJ-NYC , autoboxer , Berseker59 , Eric Jimenez , Krishan , Sanket , snorkpete
25	Brute Force Upgrade	Jim , Treveshan Naidoo
26	CRUD in Angular2 mit Restful-API	bleakgadfly , Sefa
27	Debuggen der Angular2-Typoskriptanwendung mit Visual Studio Code	PSabuwala
28	Dienste und Abhängigkeitsinjektion	BrunoLM , Eduardo Carísio , Kaspars Bergs , Matrim , Roope Hakulinen , Syam Pradeep , theblindprophet
29	Dropzone in Angular2	Ketan Akbari
30	Dynamisches Hinzufügen von Komponenten mithilfe von ViewContainerRef.createComponent	amansoni211 , daniellmb , Günter Zöchbauer , jupiter24 , Khaled
31	eckiger Redux	Yoav Schniederman

32	Ermitteln von Größenänderungsereignissen	Eric Jimenez
33	Erstellen einer Angular-npm-Bibliothek	Maciej Treder
34	Erstellen Sie ein Angular 2+ NPM-Paket	BogdanC , Janco Boscan , vinagreti
35	EventEmitter-Dienst	Abrar Jahin
36	Fass	TechJhola
37	Faules Laden eines Moduls	Batajus , M4R1KU , Shannon Young , Syam Pradeep
38	Funktionsmodule	AryanJ-NYC , gsc
39	Geräteprüfung	Yoav Schniederman
40	Häufig eingebaute Richtlinien und Dienste	Jim , Sanket
41	HTTP Interceptor	Everettss , Mihai , Mike Kovetsky , Nilz11 , Paul Marshall , peeskillt , theblindprophet
42	Installieren von Drittanbieter-Plugins mit angle-cli@1.0.0-beta.10	Alex Morales , Daredzik , filoxo , Kaspars Bergs , pd farhad
43	Komponenten	BrunoLM
44	Komponenteninteraktionen	H. Pauwelyn , Janco Boscan , LLL , Sefa
45	Konfigurieren der ASP.net Core-Anwendung für die Arbeit mit Angular 2 und TypeScript	Oleksii Aza , Sam
46	Lebenszyklus-Haken	Alexandre Junges , daniellmb , Deen John , muetzerich , Sbats , theblindprophet
47	Module	BrunoLM
48	NgModel testen	jesussegado
49	ngrx	Maxime
50	Optimieren des Renderns mit ChangeDetectionStrategy	daniellmb , Eric Jimenez , Everettss
51	OrderBy Pipe	Yoav Schniederman

52	Pfeifen	adcdjunior , Boris , borislemke , BrunoLM , Christopher Taylor , Chybie , daniellmb , Daredzik , elliott-j , Everettss , Fredrik Lundin , Jarod Moser , Jeff Cross , Jim , Kaspars Bergs , Leon Adler , Lexi , LordTribual , michaelbahr , Philipp Kief , theblindprophet
53	Richtlinien	adcdjunior , Andrei Zhytkevich , borislemke , BrunoLM , daniellmb , Everettss , lexith , Stian Standahl , theblindprophet
54	Richtlinien und Komponenten: @Input @Output	adcdjunior , dafyddPrys , Everettss , Joel Almeida , lexith , muetzerich , theblindprophet , ThomasP1988
55	Routing	aholtry , Apmis , AryanJ-NYC , borislemke , camwhite , Kaspars Bergs , LordTribual , Sachin S , theblindprophet
56	Routing (3.0.0+)	Ai_boy , Alexis Le Gal , Everettss , Gerard Simpson , Kaspars Bergs , mast3rd3mon , meorfi , rivanov , SlashTag , smnbbv , theblindprophet , ThomasP1988 , Trent
57	Seitentitel	Yoav Schniederman
58	Servicemitarbeiter	Roberto Fernandez
59	Spott @ ngx / Store	BrianRT , Hatem , Jim , Lucas , Yoav Schniederman
60	Testen einer Angular 2 App	Arun Redhu , michaelbahr , nick , Reza , Rumit Parakhiya
61	Umgehen Desinfektion für vertrauenswürdige Werte	Scrambo
62	Verwenden Sie in Angular 2 native Webkomponenten	ugreen
63	Verwenden von Drittanbieter-Bibliotheken wie jQuery in Angular 2	Ashok Vishwakarma
64	Verwendung von ngfor	Jorge , Yoav Schniederman
65	Vorlagen	Max Karpovets
66	Wie benutze ich ngif?	Amit kumar , ob1 , ppovoski , samAlvin
67	Winkel 2 - Winkelmesser	Yoav Schniederman

68	Winkel-Cli-Testabdeckung	ahmadalibaloch
69	Zone.js	Roope Hakulinen