Kostenloses eBook

LERNEN Angular

Free unaffiliated eBook created from
Stack Overflow contributors.

#angular

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: angular

It is an unofficial and free Angular 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.

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

Bemerkungen

Angular (allgemein als " **Angular 2+** " oder " **Angular 2** " bezeichnet) ist ein TypeScript- basiertes Open-Source-Front-End-Web-Framework, das vom Angular Team bei Google und von einer Gemeinschaft von Einzelpersonen und Unternehmen angesprochen wird Teile des Workflow des Entwicklers beim Erstellen komplexer Webanwendungen. Angular ist eine vollständige Überarbeitung vom selben Team, das AngularJS entwickelt hat . [1]

Das Framework besteht aus mehreren Bibliotheken , von denen einige Core (@ Angle / Core zum Beispiel) und einige optional (@ Angle / Animationen) sind.

Sie schreiben Angular-Anwendungen, indem Sie HTML- *Vorlagen* mit Angularized Markup erstellen, *Komponentenklassen* zum Verwalten dieser Vorlagen schreiben, Anwendungslogik in *Services* hinzufügen und Komponenten und Services in *Modulen integrieren* .

Dann starten Sie die App durch *Bootstrapping* des *Root-Moduls* . Angular übernimmt die Aufgabe, präsentiert Ihren Anwendungsinhalt in einem Browser und reagiert auf Benutzerinteraktionen gemäß den Anweisungen, die Sie gegeben haben.

Der grundlegendste Teil der Entwicklung von Angular-Anwendungen sind die **Komponenten** . Eine Komponente ist eine Kombination aus einer HTML-Vorlage und einer Komponentenklasse, die einen Teil des Bildschirms steuert. Hier ist ein Beispiel für eine Komponente, die eine einfache Zeichenfolge anzeigt:

src / app / app.component.ts

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

@Component({
    selector: 'my-app',
    template: `<h1>Hello {{name}}</h1>`
})
export class AppComponent {
    name = 'Angular';
}
```

Jede Komponente beginnt mit einer `@Component` Decorator-Funktion, die ein Metadatenobjekt übernimmt. Das Metadatenobjekt beschreibt, wie die HTML-Vorlage und die Komponentenklasse zusammenarbeiten.

Die `selector` Eigenschaft weist Angular an, die Komponente innerhalb eines benutzerdefinierten `<my-app>` -Tags in der *index.html*- Datei anzuzeigen.

index.html (im `body` Tag)

```
<my-app>Loading AppComponent content here ...</my-app>
```

Die Vorlageneigenschaft definiert eine Nachricht innerhalb eines `<h1>` -Headers. Die Nachricht beginnt mit "Hello" und endet mit `{{name}}` , einem Bindungsausdruck für die Winkelinterpolation. Zur Laufzeit ersetzt Angular `{{name}}` durch den Wert der `name` -Eigenschaft der Komponente. Die Interpolationsbindung ist eine von vielen Angular-Funktionen, die Sie in dieser Dokumentation finden. Ändern Sie im Beispiel die `name 'Angular'` der Komponentenklasse von `'Angular'` in `'World'` und sehen Sie, was passiert.

Dieses Beispiel ist in **TypeScript geschrieben** , einer Obermenge von JavaScript. Angular verwendet TypeScript, da es mit seinen Typen einfach ist, die Entwicklerproduktivität mit Werkzeugen zu unterstützen. Darüber hinaus wird **fast alles von TypeScript unterstützt. Daher** ist es **schwierig** , **einfaches JavaScript** zum Schreiben Ihrer Anwendung zu verwenden. Das Schreiben von Angular-Code in JavaScript ist jedoch möglich. Dieser Leitfaden erklärt, wie.

Weitere Informationen zur **Architektur** von Angular finden Sie **hier**

Versionen

Ausführung	Veröffentlichungsdatum
5.0.0-beta.1 (Neueste)	2017-07-27
4.3.2	2017-07-26
5.0.0-beta.0	2017-07-19
4.3.1	2017-07-19
4.3.0	2017-07-14
4.2.6	2017-07-08
4.2.5	2017-06-09
4.2.4	2017-06-21
4.2.3	2017-06-16
4.2.2	2017-06-12
4.2.1	2017-06-09
4.2.0	2017-06-08
4.2.0-rc.2	2017-06-01
4.2.0-rc.1	2017-05-26
4.2.0-rc.0	2017-05-19
4.1.3	2017-05-17

Ausführung	Veröffentlichungsdatum
4.1.2	2017-05-10
4.1.1	2017-05-04
4.1.0	2017-04-26
4.1.0-rc.0	2017-04-21
4.0.3	2017-04-21
4.0.2	2017-04-11
4.0.1	2017-03-29
4.0.0	2017-03-23
4.0.0-rc.6	2017-03-23
4.0.0-rc.5	2017-03-17
4.0.0-rc.4	2017-03-17
2.4.10	2017-03-17
4.0.0-rc.3	2017-03-10
2.4.9	2017-03-02
4.0.0-rc.2	2017-03-02
4.0.0-rc.1	2017-02-24
2.4.8	2017-02-18
2.4.7	2017-02-09
2.4.6	2017-02-03
2.4.5	2017-01-25
2.4.4	2017-01-19
2.4.3	2017-01-11
2.4.2	2017-01-06
2.4.1	2016-12-21
2.4.0	2016-12-20

Ausführung	Veröffentlichungsdatum
2.3.1	2016-12-15
2.3.0	2016-12-07
2.3.0-rc.0	2016-11-30
2.2.4	2016-11-30
2.2.3	2016-11-23
2.2.2	2016-11-22
2.2.1	2016-11-17
2.2.0	2016-11-14
2,2,0-rc,0	2016-11-02
2.1.2	2016-10-27
2.1.1	2016-10-20
2.1.0	2016-10-12
2.1.0-rc.0	2016-10-05
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
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

Installation von Angular mit Winkelkli

Dieses Beispiel ist eine schnelle Einrichtung von Angular und das Erstellen eines schnellen Beispielprojekts.

Voraussetzungen:

- Node.js 6.9.0 oder höher.
- npm v3 oder höher oder garn .
- Typisierung v1 oder höher.

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

`npm install -g typings` oder `npm install -g typings yarn global add typings`

`npm install -g @angular/cli` oder `yarn global add @angular/cli`

Der erste Befehl installiert die Typisierungsbibliothek global (und fügt die ausführbare `typings` zu PATH hinzu). Die zweite installiert @ **angle / cli** global und fügt die ausführbare Datei `ng` PATH hinzu.

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 mit Angular. Sie können jetzt zu dem in Terminal angezeigten Link navigieren und sehen, was gerade 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
# You can add --flat if you don't want to create new folder for a component
ng g component my-generated-component --flat
# You can add --spec false if you don't want a test file to be generated (my-generated-
component.spec.ts)
ng g component my-generated-component --spec false
```

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

Gerüsttyp	Verwendungszweck
Modul	`ng g module my-new-module`
Komponente	`ng g component my-new-component`
Richtlinie	`ng g directive my-new-directive`

Gerüsttyp	Verwendungszweck
Rohr	`ng g pipe my-new-pipe`
Bedienung	`ng g service my-new-service`
Klasse	`ng g class my-new-class`
Schnittstelle	`ng g interface my-new-interface`
Enum	`ng g enum my-new-enum`

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 Ihrer Angular-Web-App fertig sind und sie auf einem Webserver wie Apache Tomcat installieren 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 bietet integrierte Einheitentests, und jeder von angle-cli erstellte Artikel generiert einen grundlegenden Unit-Test, der ausgegeben 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

Winkeliges "Hello World" -Programm

Voraussetzungen:

Einrichten der Entwicklungsumgebung

Bevor wir beginnen, müssen wir unsere Umgebung einrichten.

- Installieren Sie Node.js und npm, wenn sie noch nicht auf Ihrem Computer installiert sind.

  Stellen Sie sicher, dass Sie mindestens die Knoten 6.9.x und npm 3.xx ausführen, indem Sie die Knoten -v und npm -v in einem Terminal- / Konsolenfenster ausführen. Ältere Versionen erzeugen Fehler, neuere Versionen sind jedoch in Ordnung.

- Installieren Sie die Angular-CLI global mit `npm install -g @angular/cli` .

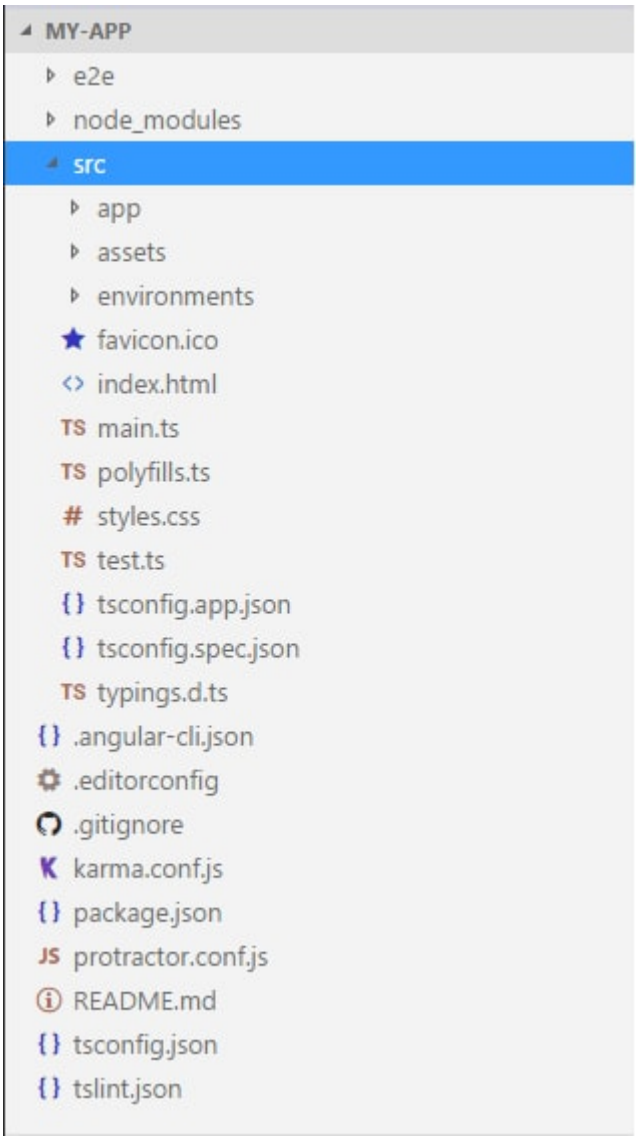
Schritt 1: Neues Projekt erstellen

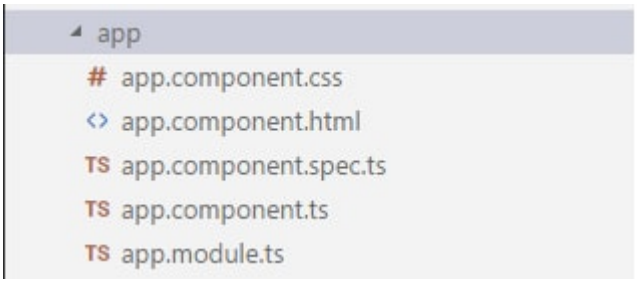
Öffnen Sie ein Terminalfenster (oder die Eingabeaufforderung Node.js in Windows).

Wir erstellen ein neues Projekt und eine neue Skeleton-Anwendung mit dem Befehl:

```
ng new my-app
```

Hier ist das `ng` für Angular. Wir bekommen so eine Dateistruktur.





Es gibt viele Dateien. Wir müssen uns jetzt nicht um alle sorgen.

Schritt 2: Bereitstellen der Anwendung

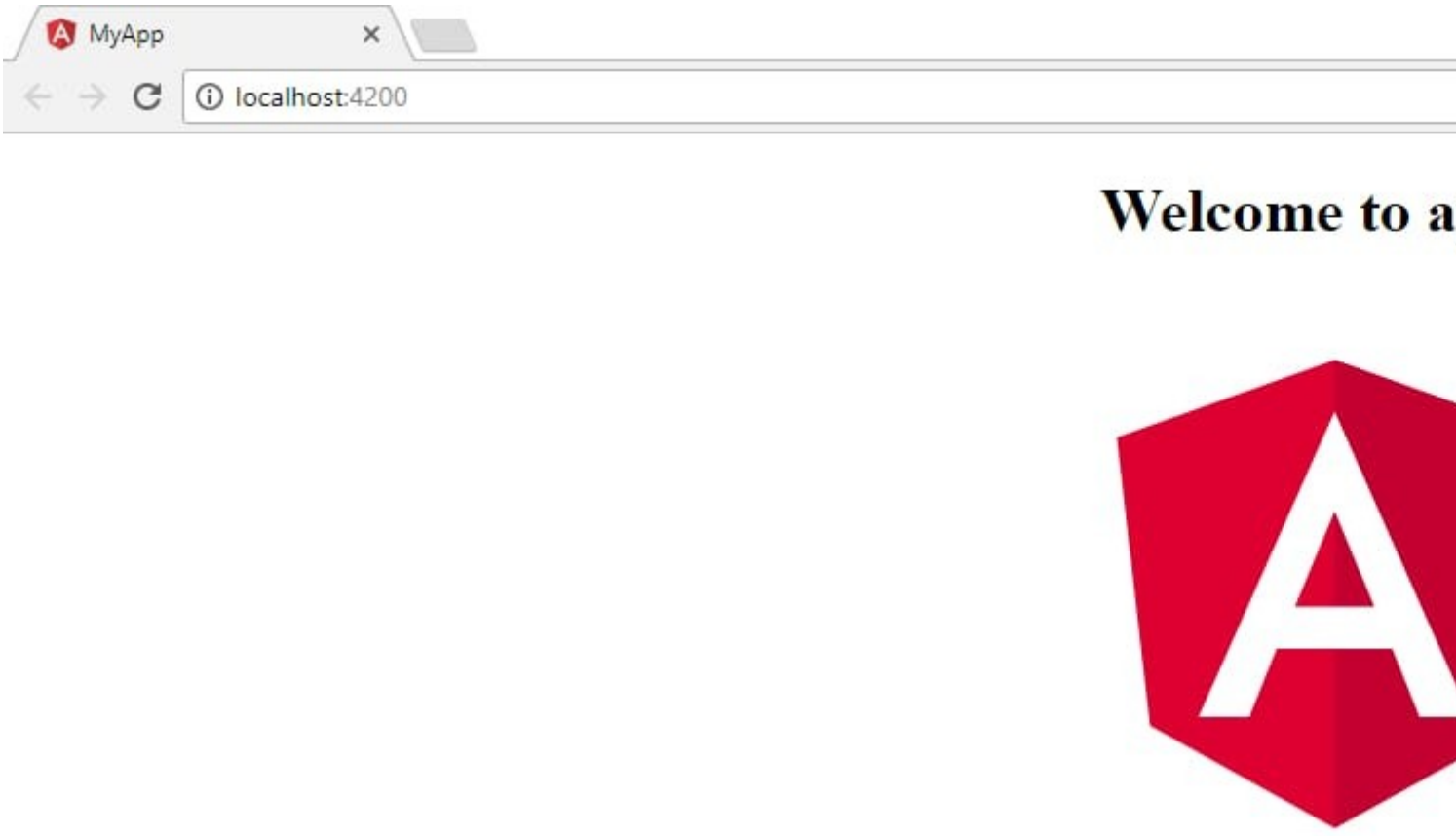
Wir starten unsere Anwendung mit folgendem Befehl:

```
ng serve
```

Wir können ein Flag `-open` (oder einfach `-o`) verwenden, das unseren Browser automatisch unter `http://localhost:4200/` öffnet.

```
ng serve --open
```

Navigieren Sie im Browser zur Adresse `http://localhost:4200/` . Es sieht ungefähr so aus:



Schritt 3: Bearbeiten unserer ersten Winkelkomponente

Die CLI hat die Standard-Angular-Komponente für uns erstellt. Dies ist die Wurzelkomponente und wird `app-root` . Man kann es in `./src/app/app.component.ts` .

Öffnen Sie die Komponentendatei und ändern Sie die Titeleigenschaft von `Welcome to app!!` zu `Hello World` . Der Browser wird automatisch mit dem überarbeiteten Titel geladen.

Originalcode: Beachten Sie den `title = 'app';`

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

Angular CLI, 20 minutes ago | 1 author (Angular CLI)
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app';
}
```

Modified Code: Wert des `title` wird geändert.

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Hello World';
}
```

Ebenso gibt es eine Änderung in `./src/app/app.component.html` .

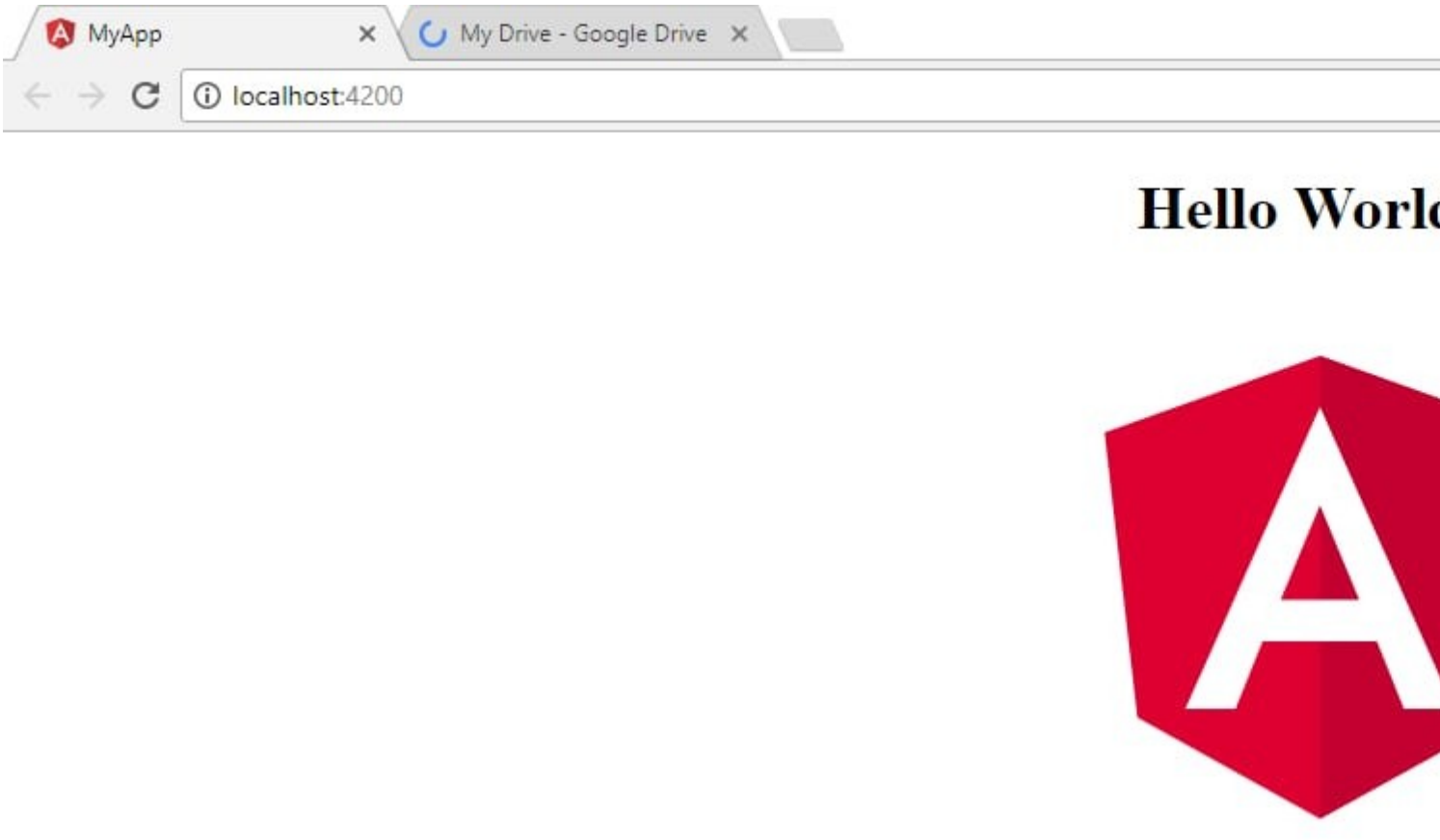
Ursprüngliches HTML

```
<div style="text-align:center">
  <h1>
    Welcome to {{title}}!!
  </h1>
  <img width="300" src="
```

Modifiziertes HTML

```
<!--The content below is only a placeholder and can be replaced.-->
<div style="text-align:center">
  <h1>
     {{title}}!!
  </h1>
  <img width="300" src="
```

Beachten Sie, dass der Wert von `title` aus der `./src/app/app.component.ts` angezeigt wird. Der Browser wird automatisch neu geladen, wenn die Änderungen abgeschlossen sind. Es sieht ungefähr so aus.



Weitere Informationen zum Thema finden Sie hier .

Erste Schritte mit Angular online lesen: https://riptutorial.com/de/angular/topic/9754/erste-schritte-mit-angular

Kapitel 2: Daten zwischen Komponenten gemeinsam nutzen

Einführung

Das Ziel dieses Themas ist die Erstellung einfacher Beispiele für verschiedene Möglichkeiten, wie Daten über die Datenbindung und den gemeinsam genutzten Dienst von Komponenten gemeinsam genutzt werden können.

Bemerkungen

Es gibt immer viele Möglichkeiten, eine Aufgabe in der Programmierung zu lösen. Gerne können Sie aktuelle Beispiele bearbeiten oder eigene hinzufügen.

Examples

Senden von Daten von einer übergeordneten Komponente an einen untergeordneten Dienst über einen gemeinsam genutzten Dienst

service.ts:

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

@Injectable()
export class AppState {

  public mylist = [];

}
```

parent.component.ts:

```
import {Component} from '@angular/core';
import { AppState } from './shared.service';

@Component({
  selector: 'parent-example',
  templateUrl: 'parent.component.html',
})
export class ParentComponent {
  mylistFromParent = [];

  constructor(private appState: AppState){
    this.appState.mylist;
  }

  add() {
    this.appState.mylist.push({"itemName":"Something"});
```

```
  }

}
```

parent.component.html:

```
<p> Parent </p>
  <button (click)="add()">Add</button>
<div>
  <child-component></child-component>
</div>
```

child.component.ts:

```
import {Component, Input } from '@angular/core';
import { AppState } from './shared.service';

@Component({
  selector: 'child-component',
  template: `
    <h3>Child powered by shared service</h3>
        {{mylist | json}}
  `,
})
export class ChildComponent {
  mylist: any;

  constructor(private appState: AppState){
    this.mylist = this.appState.mylist;
  }

}
```

Senden Sie Daten von der übergeordneten Komponente über die Datenbindung mit @Input an die untergeordnete Komponente

parent.component.ts:

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

@Component({
  selector: 'parent-example',
  templateUrl: 'parent.component.html',
})

export class ParentComponent {
  mylistFromParent = [];

  add() {
    this.mylistFromParent.push({"itemName":"Something"});
  }

}
```

parent.component.html:

```
<p> Parent </p>
  <button (click)="add()">Add</button>

<div>
  <child-component [mylistFromParent]="mylistFromParent"></child-component>
</div>
```

child.component.ts:

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

@Component({
  selector: 'child-component',
  template: `
    <h3>Child powered by parent</h3>
        {{mylistFromParent | json}}
  `,
})

export class ChildComponent {
  @Input() mylistFromParent = [];
}
```

Senden von Daten vom untergeordneten an das übergeordnete Ereignis über den Ereignisausgeber @Output

event-emitter.component.ts

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

@Component({
  selector: 'event-emitting-child-component',
  template: `<div *ngFor="let item of data">
              <div (click)="select(item)">
                {{item.id}} = {{ item.name}}
              </div>
             </div>
            `
})

export class EventEmitterChildComponent implements OnInit{

  data;

  @Output()
  selected: EventEmitter<string> = new EventEmitter<string>();

  ngOnInit(){
    this.data = [ { "id": 1, "name": "Guy Fawkes", "rate": 25 },
                  { "id": 2, "name": "Jeremy Corbyn", "rate": 20 },
                  { "id": 3, "name": "Jamie James", "rate": 12 },
                  { "id": 4, "name": "Phillip Wilson", "rate": 13 },
                  { "id": 5, "name": "Andrew Wilson", "rate": 30 },
                  { "id": 6, "name": "Adrian Bowles", "rate": 21 },
                  { "id": 7, "name": "Martha Paul", "rate": 19 },
                  { "id": 8, "name": "Lydia James", "rate": 14 },
                  { "id": 9, "name": "Amy Pond", "rate": 22 },
```

```
                  { "id": 10, "name": "Anthony Wade", "rate": 22 } ]
  }

  select(item) {
      this.selected.emit(item);

  }

}
```

Ereignisempfänger.Komponente.ts:

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

@Component({
    selector: 'event-receiver-parent-component',
    template: `<event-emitting-child-component (selected)="itemSelected($event)">
                </event-emitting-child-component>
                <p *ngIf="val">Value selected</p>
                <p style="background: skyblue">{{ val | json}}</p>`
})

export class EventReceiverParentComponent{
  val;

  itemSelected(e){
    this.val = e;
  }
}
```

Senden von Daten asynchron von übergeordneten an untergeordnete Objekte mit Observable und Subject

shared.service.ts:

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

import 'rxjs/add/operator/toPromise';

import { Observable } from 'rxjs/Observable';
import { Observable } from 'rxjs/Rx';
import {Subject} from 'rxjs/Subject';


@Injectable()
export class AppState {

  private headers = new Headers({'Content-Type': 'application/json'});
  private apiUrl = 'api/data';

  // Observable string source
  private dataStringSource = new Subject<string>();

  // Observable string stream
  dataString$ = this.dataStringSource.asObservable();
```

```
  constructor(private http: Http) { }

  public setData(value) {
    this.dataStringSource.next(value);
  }

  fetchFilterFields() {
    console.log(this.apiUrl);
    return this.http.get(this.apiUrl)
               .delay(2000)
               .toPromise()
               .then(response => response.json().data)
               .catch(this.handleError);
  }

  private handleError(error: any): Promise<any> {
    console.error('An error occurred', error); // for demo purposes only
    return Promise.reject(error.message || error);
  }

}
```

parent.component.ts:

```
import {Component, OnInit} from '@angular/core';
import 'rxjs/add/operator/toPromise';
import { AppState } from './shared.service';

@Component({
  selector: 'parent-component',
  template: `
              <h2> Parent </h2>
              <h4>{{promiseMarker}}</h4>

              <div>
                <child-component></child-component>
              </div>
            `
})
export class ParentComponent implements OnInit {

  promiseMarker = "";

  constructor(private appState: AppState){ }

  ngOnInit(){
    this.getData();
  }

  getData(): void {
    this.appState
        .fetchFilterFields()
        .then(data => {
          // console.log(data)
          this.appState.setData(data);
          this.promiseMarker = "Promise has sent Data!";
        });
  }

}
```

child.component.ts:

```
import {Component, Input } from '@angular/core';
import { AppState } from './shared.service';


@Component({
  selector: 'child-component',
  template: `
    <h3>Child powered by shared service</h3>
        {{fields | json}}
  `,
})
export class ChildComponent {
  fields: any;

  constructor(private appState: AppState){
    // this.mylist = this.appState.get('mylist');

    this.appState.dataString$.subscribe(
      data => {
        // console.log("Subs to child" + data);
        this.fields = data;
      });

  }

}
```

Daten zwischen Komponenten gemeinsam nutzen online lesen:
https://riptutorial.com/de/angular/topic/10836/daten-zwischen-komponenten-gemeinsam-nutzen

Kapitel 3: Event-Emitter

Examples

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);
    }
}
```

Event-Emitter online lesen: https://riptutorial.com/de/angular/topic/9828/event-emitter

Kapitel 4: Formen

Examples

Reaktive Formen

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 './app.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
            };
        }
    }
}
```

Vorlagengesteuerte Formulare

Vorlage - `signup.component.html`

```
<form #signUpForm="ngForm" (ngSubmit)="onSubmit()">

  <div class="title">
    Sign Up
  </div>

  <div class="input-field">
    <label for="username">username</label>
    <input
      type="text"
      pattern="\w{4,20}"
      name="username"
      required="required"
      [(ngModel)]="signUpRequest.username" />
  </div>

  <div class="input-field">
    <label for="email">email</label>
    <input
      type="email"
      pattern="^\S+@\S+$"
      name="email"
      required="required"
```

```
      [(ngModel)]="signUpRequest.email" />
  </div>

  <div class="input-field">
    <label for="password">password</label>
    <input
      type="password"
      pattern=".{6,30}"
      required="required"
      name="password"
      [(ngModel)]="signUpRequest.password" />
  </div>

  <div class="status">
    {{ status }}
  </div>

  <button [disabled]="!signUpForm.form.valid" type="submit">
    <span>Sign Up</span>
  </button>

</form>
```

Komponente - `signup.component.ts`

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

import { SignUpRequest } from './signup-request.model';

@Component({
  selector: 'app-signup',
  templateUrl: './signup.component.html',
  styleUrls: ['./signup.component.css']
})
export class SignupComponent {

  status: string;
  signUpRequest: SignUpRequest;

  constructor() {
    this.signUpRequest = new SignUpRequest();
  }

  onSubmit(value, valid) {
    this.status = `User ${this.signUpRequest.username} has successfully signed up`;
  }

}
```

Modell - `signup-request.model.ts`

```
export class SignUpRequest {

  constructor(
    public username: string="",
    public email: string="",
```

```
    public password: string=""
  ) {}

}
```

App Modul - `app.module.ts`

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

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

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

App-Komponente - `app.component.html`

```
<app-signup></app-signup>
```

Formen online lesen: https://riptutorial.com/de/angular/topic/9825/formen

Kapitel 5: Für Schleife

Examples

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.

Für Schleife online lesen: https://riptutorial.com/de/angular/topic/9826/fur-schleife

Kapitel 6: Pfeifen

Einführung

Die Rohre sind sehr ähnlich wie Filter in AngularJS, dass sie beide helfen , die Daten in einer bestimmten format.The Pipe - Zeichen zu verwandeln | dient zum Auftragen von Rohren in Angular.

Examples

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 { }
```

Mehrere benutzerdefinierte Rohre

Unterschiedliche Rohre zu haben, ist ein sehr häufiger Fall, bei dem jedes Rohr eine andere Sache tut. Das Hinzufügen jeder Pipe zu jeder Komponente kann zu einem sich wiederholenden Code werden.

Es ist möglich, alle häufig verwendeten Rohrleitungen in einem `Module` zu bündeln und das neue Modul in eine Komponente zu importieren, die die Rohrleitungen benötigt.

breaklines.ts

```
import { Pipe } from '@angular/core';
/**
 * pipe to convert the \r\n into <br />
 */
@Pipe({ name: 'br' })
export class BreakLine {
    transform(value: string): string {
        return value == undefined ? value :
             value.replace(new RegExp('\r\n', 'g'), '<br />')
              .replace(new RegExp('\n', 'g'), '<br />');
    }
}
```

Großbuchstaben.ts

```
import { Pipe } from '@angular/core';
/**
 * pipe to uppercase a string
 */
@Pipe({ name: 'upper' })
export class Uppercase{
    transform(value: string): string {
        return value == undefined ? value : value.toUpperCase( );
    }
}
```

Pipes.module.ts

```
import { NgModule } from '@angular/core';
import { BreakLine } from './breakLine';
import { Uppercase} from './uppercase';

@NgModule({
    declarations: [
        BreakLine,
        Uppercase
```

```
    ],
    imports: [

    ],
    exports: [
        BreakLine,
        Uppercase
    ]
    ,
})
export class PipesModule {}
```

meine.komponente.ts

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

@Component({
  selector: 'my-component',
  template: `{{ value | upper | br}}`
})
export class MyComponent {

    public value: string;

}
```

mein.module.ts

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

import { MyComponent } from './my.component';
import { PipesModule} from './pipes.module';

@NgModule({
  imports: [
    BrowserModule,
    PipesModule,
  ],
  declarations: [
    MyComponent,
  ],
})
```

Pfeifen online lesen: https://riptutorial.com/de/angular/topic/9824/pfeifen

Kapitel 7: Routing

Examples

Routing mit Kindern

Ich habe festgestellt, dass dies der richtige Weg ist, um untergeordnete Routen in der Datei app.routing.ts oder app.module.ts (abhängig von Ihren Vorlieben) richtig zu verschachteln. 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

Grundlegendes Routing

Der Router ermöglicht die Navigation von einer Ansicht zur anderen basierend auf den Benutzerinteraktionen mit der Anwendung.

Im Folgenden werden die Schritte zur Implementierung des grundlegenden Routings in Angular beschrieben:

HINWEIS : Stellen Sie sicher, dass Sie dieses Tag haben:

```
<base href='/'>
```

als erstes Kind unter Ihrem Head-Tag in Ihrer index.html-Datei. Dieses Element gibt an, dass Ihr App-Ordner der Anwendungsstamm ist. Angular würde dann wissen, wie Sie Ihre Links organisieren.

1. Prüfen Sie, ob Sie in package.json (unter Verwendung der neuesten Version von Angular) auf die richtigen / neuesten Routing-Abhängigkeiten verweisen und bereits eine `npm install`

   ```
   "dependencies": {
       "@angular/router": "^4.2.5"
   }
   ```

2. Definieren Sie die Route gemäß ihrer Schnittstellendefinition:

   ```
   interface Route {
     path?: string;
     pathMatch?: string;
     component?: Type<any>;
   }
   ```

3. Importieren Sie in einer Routing-Datei (`routes/app.routing.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.

```
import { Routes, RouterModule } from '@angular/router';
import { ModuleWithProviders } from '@angular/core';
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 APP_ROUTES: Routes = [
    { 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_ROUTING: ModuleWithProviders = RouterModule.forRoot(APP_ROUTES);
```

4. Platzieren Sie dies in Ihrer `app.module.ts` unter `@NgModule([])` unter `imports` :

```
// Alternatively, just import 'APP_ROUTES
import {APP_ROUTING} from '../routes/app.routing.ts';
@NgModule([
    imports: [
        APP_ROUTING
        // Or RouterModule.forRoot(APP_ROUTES)
    ]
])
```

5. Laden / Anzeigen der Routerkomponenten basierend auf dem Pfad, auf den zugegriffen wird. Die Direktive `<router-outlet>` gibt an, wohin die Komponente geladen werden soll.

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

@Component({
    selector: 'demo-app',
    template: `
        <div>
            <router-outlet></router-outlet>
        </div>
    `
})
export class AppComponent {}
```

6. Verbinden Sie die anderen Routen. Standardmäßig `RouterOutlet` die Komponente, für die in den `Routes` leerer Pfad angegeben ist. `RouterLink` Anweisung wird mit dem HTML- `RouterLink` verwendet, um die an Routen angeschlossenen Komponenten zu laden. `RouterLink` generiert das href-Attribut, das zum Generieren von Links verwendet wird. Zum Beispiel:

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

@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>
        `
})
export class AnotherComponent { }
```

Nun können wir gut auf statische Pfade routen. `RouterLink` unterstützt auch den dynamischen Pfad, indem zusätzliche Parameter zusammen mit dem Pfad übergeben werden.

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

@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>
  `
})
export class SecondComponent { }
```

`RouterLink` ein Array, in dem der erste Parameter der Pfad für das Routing und die nachfolgenden Elemente die dynamischen Routing-Parameter sind.

Routing online lesen: https://riptutorial.com/de/angular/topic/9827/routing

Kapitel 8: RXJS und Observables

Examples

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;
    }));
}
```

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

RXJS und Observables online lesen: https://riptutorial.com/de/angular/topic/9829/rxjs-und-observables

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Angular	aholtry, Anup Kumar Gupta, BogdanC, Community, daddycool, Edric, Fahad Nisar, Faisal, Hendrik Brummermann, Philipp Kief, Tom
2	Daten zwischen Komponenten gemeinsam nutzen	0mpurdy, BogdanC, JGFMK, Nehal
3	Event-Emitter	aholtry
4	Formen	aholtry, Saka7
5	Für Schleife	aholtry
6	Pfeifen	aholtry, Amr ElAdawy
7	Routing	0mpurdy, aholtry, Edric
8	RXJS und Observables	aholtry