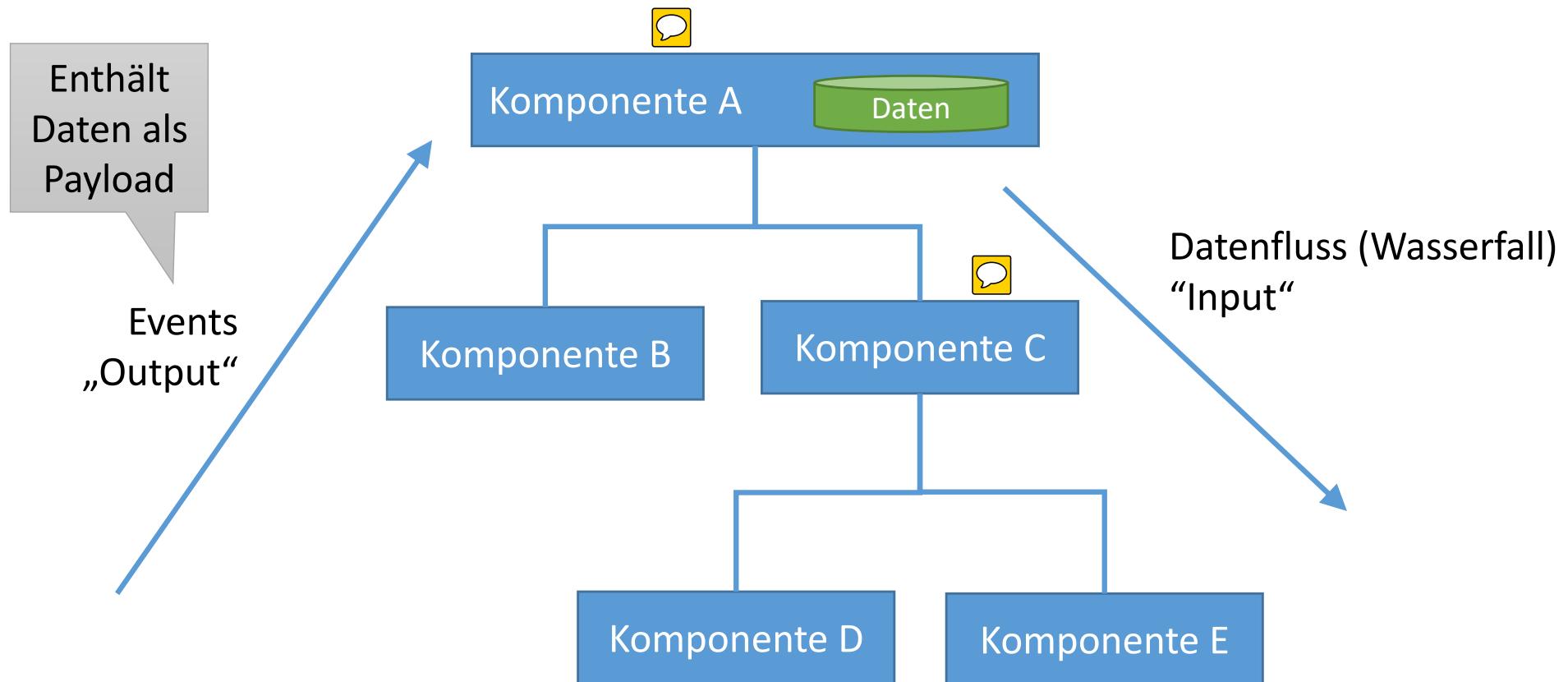


## 2.2 Angular (Teil 2)

- Jegliche Funktionalität in einer Komponente zu implementieren ist bei kleinen Projekten einfach. Bei höherem Funktionsumfang wird es jedoch schnell unübersichtlich. Im zweiten Teil der Angular Vorlesung spielt die Kapselung von Funktionalität und Kommunikation daher eine große Rolle. Dazu kommt die Verwendung von HTTP, Routing und Erweiterungen für Templates.
- Inhalt:
  1. Komponenten-Kommunikation
  2. Services
  3. HTTP
  4. Pipe
  5. Lifecycle-Hooks
  6. Routing
  7. Debugging
  8. Infrastruktur

## 2.2.1 Kommunikation zwischen Komponenten

Komponentenbaum



## 2.2.1 Kommunikation zwischen Komponenten: Input

app.component.ts



```
export class AppComponent {  
  listA: number[];  
  listB: number[];  
  
  constructor() {  
    this.listA = [  
      0, 2, 4, 6  
    ];  
  
    this.listB = [  
      1, 3, 5, 7  
    ];  
  }  
}
```

Ziel:

**Gerade Zahlen**

0  
2  
4  
6

**Ungerade Zahlen**

1  
3  
5  
7

## 2.2.1 Kommunikation zwischen Komponenten: Input

list.component.ts

```
import {Component, Input, OnInit} from  
'@angular/core';  
  
@Component({  
  selector: 'app-list',  
  templateUrl: './list.component.html',  
  styleUrls: ['./list.component.css']  
})  
export class ListComponent {  
   @Input() title: string;  
  @Input() list: number[];  
}
```

@Input() aktiviert Property Binding per HTML-Attribute

Ziel:

Gerade Zahlen

0  
2  
4  
6

Ungerade Zahlen

1  
3  
5  
7

„**title** wurde mit **@Input** dekoriert.“

## 2.2.1 Kommunikation zwischen Komponenten: Input

app.component.html

```
<app-list [list]="listA" [title]="'Gerade Zahlen'">  
</app-list>  
  
<app-list [list]="listB" [title]="'Ungerade Zahlen'">  
</app-list>
```

Property  
Binding

Dieselbe Komponente mehrmals instanziieren

Innerhalb der äußeren Anführungszeichen muss ein gültiger TypeScript Ausdruck stehen.  
Mit den einfachen Anführungszeichen wird ein Stringliteral gebildet.

Ziel:

Gerade Zahlen
0
2
4
6

Ungerade Zahlen
1
3
5
7

**list** und **title** sind TypeScript Properties, die mit **@Input()** dekoriert wurden.

Daraus folgt: **[list]** und **[title]** sind HTML Attribute.

## 2.2.1 Kommunikation zwischen Komponenten: Input

list.component.html

```
<h2>{{title}}</h2>
<p *ngFor="let item of list">
  {{item}}
</p>
```

Ziel:

**Gerade Zahlen**

0  
2  
4  
6

**Ungerade Zahlen**

1  
3  
5  
7

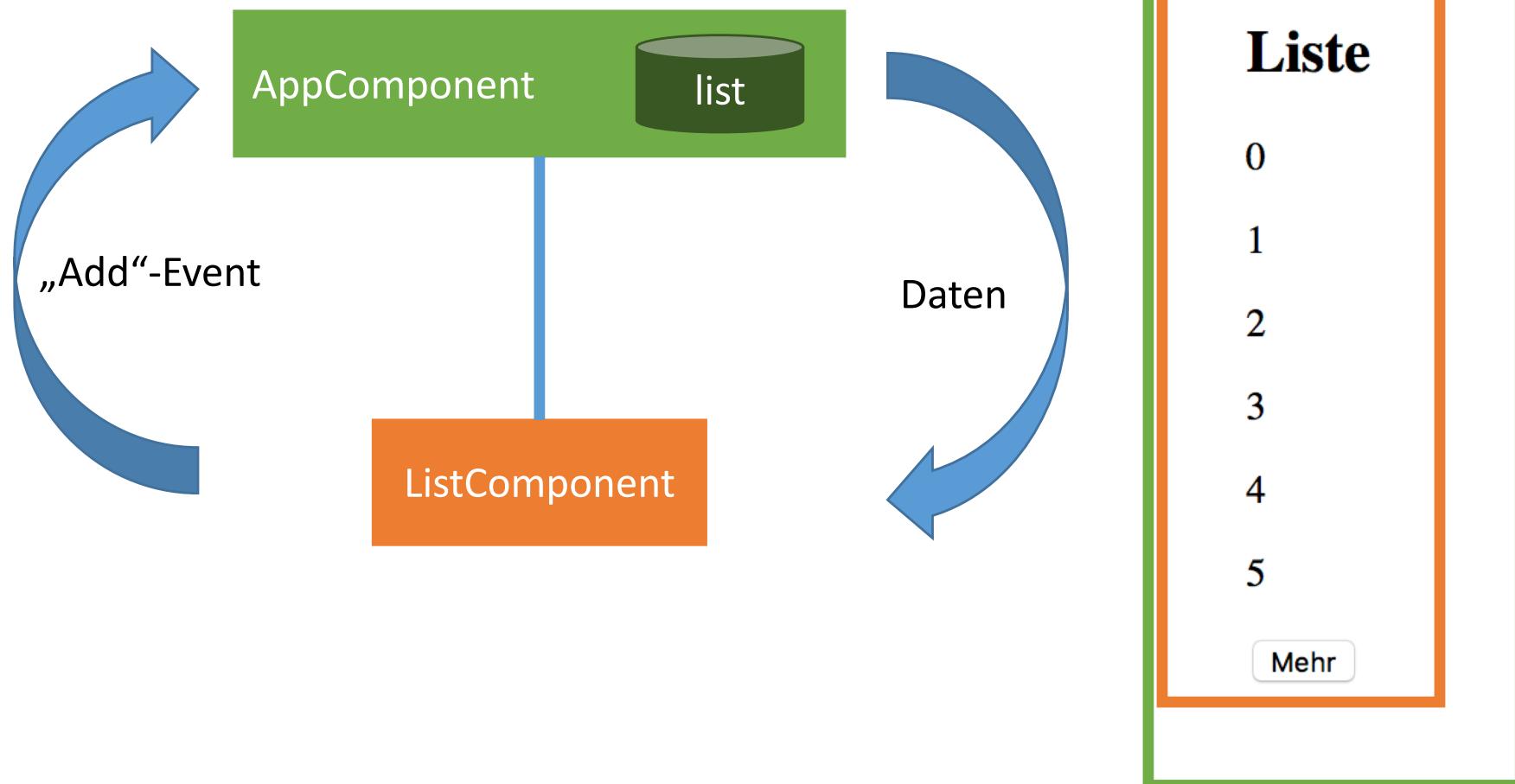
## 2.2.1 Kommunikation zwischen Komponenten: Input

- Wiederverwendbarkeit
- Kapselung von Daten und Funktionalität
- Analog zur objektorientierten Programmierung
  - Komponente => Klasse
  - @Input() => Konstruktorsignatur
  - Selector => Konstruktor
  - Instanz => Objekt

## 2.2.1 Kommunikation zwischen Komponenten: Output

AppComponent ist die Datenquelle.

Ziel:

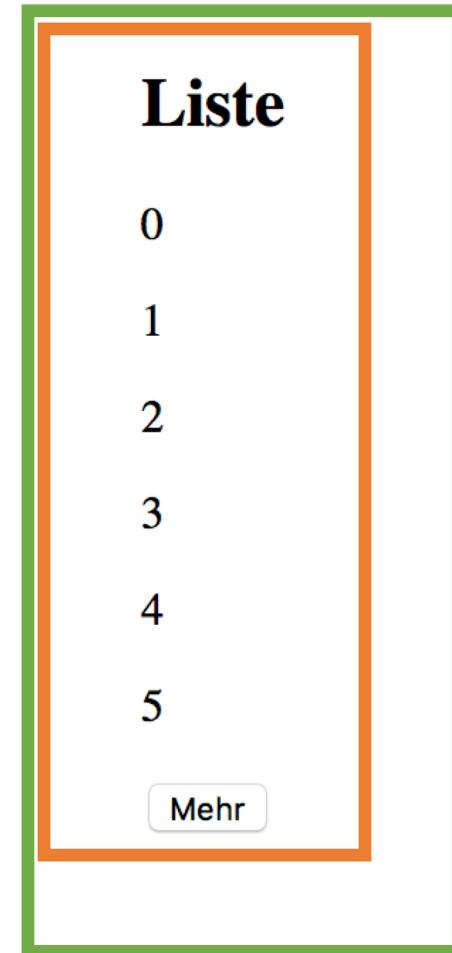


## 2.2.1 Kommunikation zwischen Komponenten: Output

Ziel:

list.component.html

```
<h2>Liste</h2>
<p *ngFor="let item of list">
  {{item}}
</p>
<button (click)="more()">
  Mehr
</button>
```



## 2.2.1 Kommunikation zwischen Komponenten: Output

list.component.ts

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

@Component({
  selector: 'app-list',
  templateUrl: './list.component.html',
  styleUrls: ['./list.component.css']
})
export class ListComponent {
  @Input() list: number[];
  @Output() addMoreEvent: EventEmitter<number>;
  constructor() {
    this.addMoreEvent = new EventEmitter<number>();
  }
  more(): void {
    let latestValue: number = this.list[this.list.length - 1];
    let newValue = latestValue + 1;
    this.addMoreEvent.emit(newValue);
  }
}
```

Initialisieren!

@Output() erzeugt ein bindbares Event

Typ des Payloads

Neue Zahl herausfinden (Anwendungsbeispiel)

Event triggern mit Payload

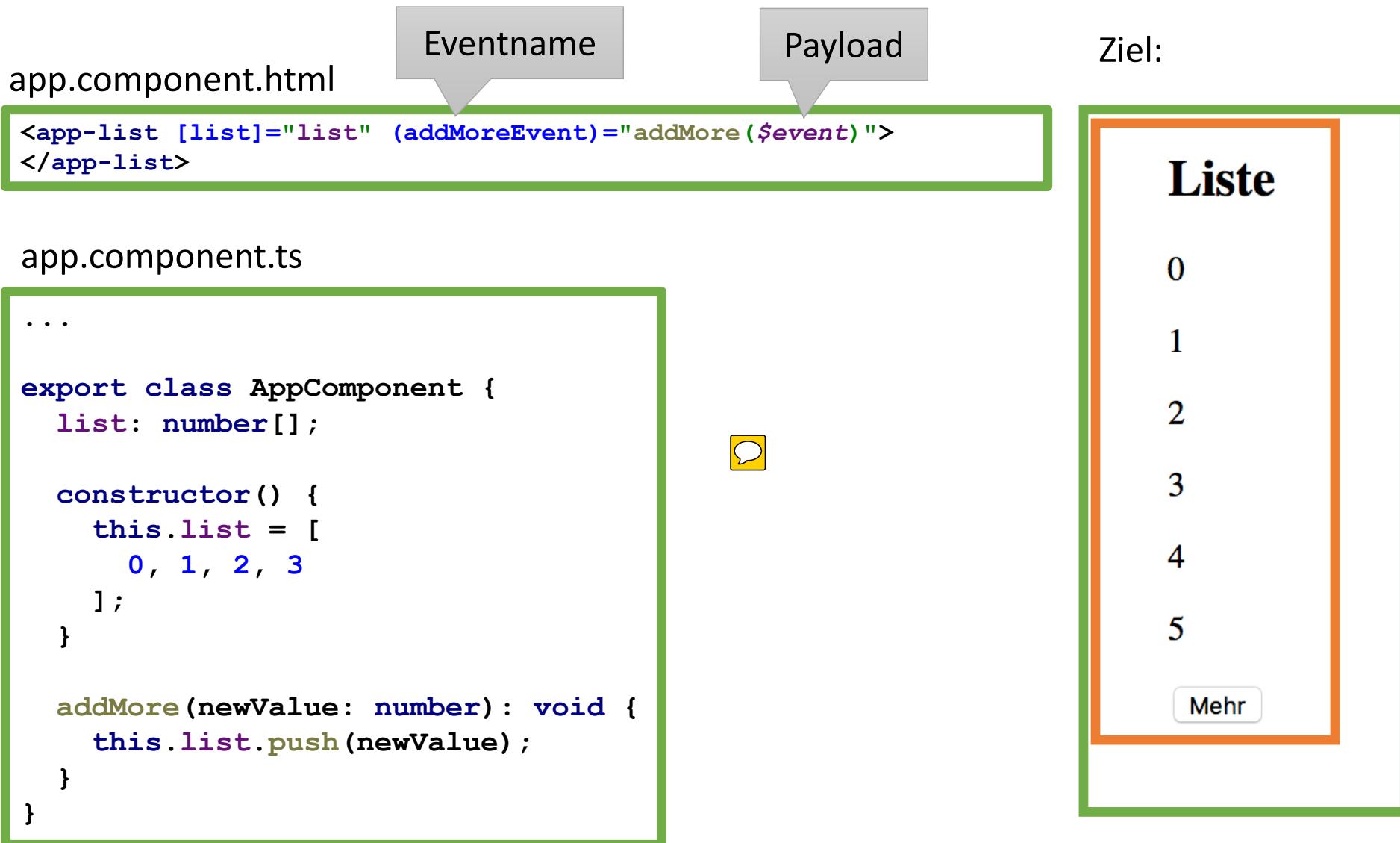
Ziel:

Liste

0  
1  
2  
3  
4  
5

Mehr

## 2.2.1 Kommunikation zwischen Komponenten: Output



## 2.2.1 Kommunikation zwischen Komponenten: Anwendungsbeispiel Userman

- Formular des Userman als eigene Komponente
  - Kein Input
  - Output
    - Add Event (Userobjekt als Payload)
- Liste des Userman als eigene Komponente
  - Input
    - Userliste
  - Output
    - Delete Event (UserID als Payload)
    - Edit Clicked Event (UserID als Payload)

## 2.2.1 Kommunikation zwischen Komponenten: Anwendungsbeispiel Userman

### Userman

## Usermanager

5 Nutzer wurden gefunden ×

#	Vorname	Nachname	Datum		
1	Peter	Kneisel	2018-4-22 19:08:47		
2	Dennis	Priefer	2018-4-22 19:08:47		
3	Wolf	Rost	2018-4-22 19:08:47		
4	Kevin	Linne	2018-4-22 19:08:47		
5	Samuel	Schepp	2018-4-22 19:08:47		

### Nutzer hinzufügen

Vorname

Nachname

**Hinzufügen**

## 2.2.1 Kommunikation zwischen Komponenten: Anwendungsbeispiel Userman

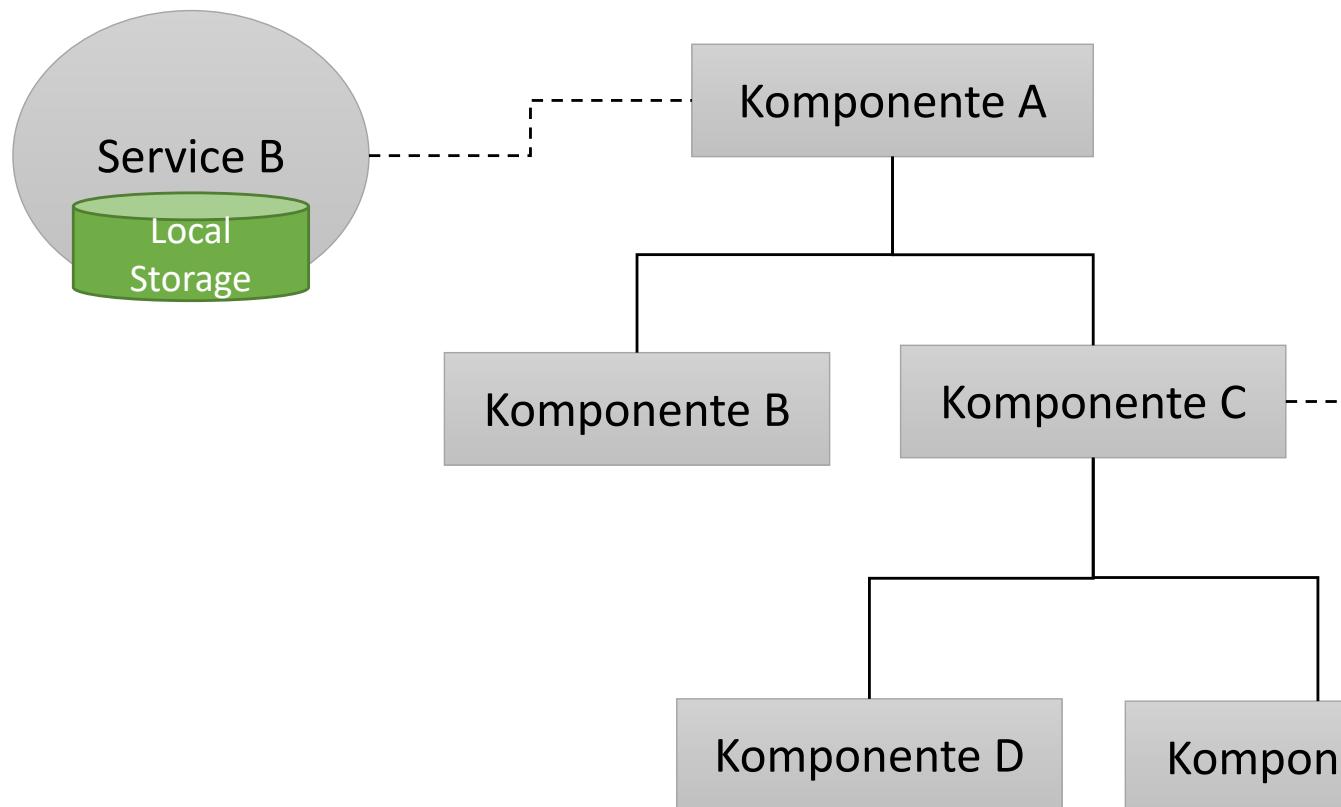
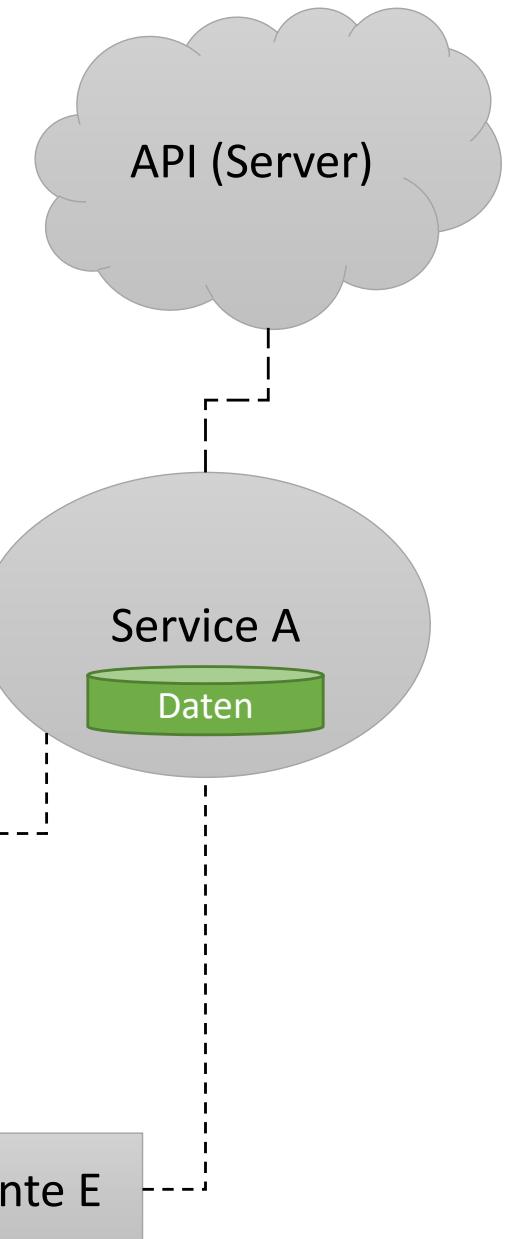
The screenshot shows a web-based application titled "Usermanager". At the top, a header bar contains the title "Userman". Below the header, the main content area has a yellow border. On the left, a light blue header bar indicates "5 Nutzer wurden gefunden". The main table area has a green border and displays a list of five users with columns for #, Vorname, Nachname, and Datum. Each row includes edit and delete icons. On the right, a purple-bordered form titled "Nutzer hinzufügen" allows for adding new users with fields for Vorname ("Max") and Nachname ("Mustermann"). A blue "Hinzufügen" button is at the bottom of the form.

#	Vorname	Nachname	Datum		
1	Peter	Kneisel	2018-4-22 19:08:47		
2	Dennis	Priefer	2018-4-22 19:08:47		
3	Wolf	Rost	2018-4-22 19:08:47		
4	Kevin	Linne	2018-4-22 19:08:47		
5	Samuel	Schepp	2018-4-22 19:08:47		

## 2.2.2 Service

Services bündeln Funktionalität und Zuständigkeiten, die nicht direkt mit der Benutzeroberfläche in Verbindung stehen.

Beispiele: Authentifizierung, Daten, Anwendungszustand, Einstellungen.



## 2.2.2 Service: Beispiel Data Service



```
ng g service data -m app
```

data.service.ts

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

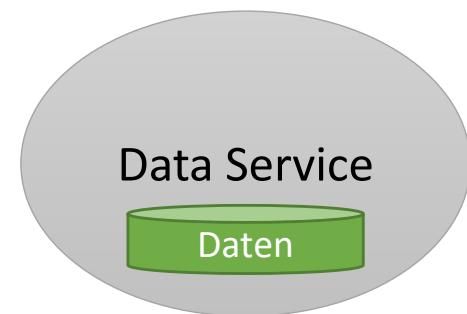
@Injectable()
export class DataService {
  numberList: number[];
  private counter: number;

  constructor() {
    this.numberList = [0, 1, 2, 3];
    this.counter = 4;
  }

  public addItem() {
    this.numberList.push(this.counter);
    this.counter += 1;
  }
}
```

Erzeugt Eintrag in  
app.module.ts

```
...
providers: [DataService],
...
```



## 2.2.2 Service: Beispiel Data Service benutzen

app.component.ts

```
import { Component } from '@angular/core';
import { DataService } from "./data.service";

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

  constructor(public dataService: DataService) {}

  addButtonClicked(): void {
    this.dataService.addItem()
  }
}
```

1) Access Modifier  
erzeugen Klassen  
Property.

2) Dependency  
Injection

## 2.2.2 Service: Beispiel Data Service benutzen

app.component.html

```
<p *ngFor="let item of dataService.numberList">  
  {{item}}  
</p>  
<button (click)="addButtonClicked()">More</button>
```

0  
1  
2  
3  
4  
5

More

## 2.2.3 HTTP Service: Benutzen

app.component.ts

```
import { Component } from '@angular/core';
import { HttpClient, HttpErrorResponse } from "@angular/common/http";
...
export class AppComponent {
  response: string;

  constructor(private httpClient: HttpClient) {
    this.response = "laden...";

    this.httpClient.post("http://localhost:8080/user", {
      vorname: "Peter",
      nachname: "Kneisel"
    }).toPromise()
      .then((res) => {
        console.log(res);
        return this.httpClient.get("http://localhost:8080/user?id=0").toPromise()
      })
      .then((data: any) => {
        this.response = JSON.stringify(data);
      })
      .catch((err: HttpErrorResponse) => {
        console.log(err.message);
      })
  }
}
```

Hinweis: HTTP eher in OnInit ausführen. Später dazu mehr.

HTTP Service als  
Dependency Injection

POST Body

Mehr HTTP Methoden: <https://angular.io/guide/http#sending-data-to-the-server>

## 2.2.3 HTTP Service: einbinden

app.module.ts

Der HTTP Service muss in „imports“ eingetragen werden.

Erst dann kann das Injection System eine Instanz des Service erzeugen.

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

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

## 2.2.4 Pipe: Beispiele

app.component.ts

```
export class AppComponent {  
  today: Date;  
  balance: number;  
  description: string;  
  
  constructor() {  
    this.today = new Date();  
    this.balance = 354.657;  
    this.description = 'angular';  
  }  
}
```

app.component.html

```
<p>{{today | date}}</p>  
<p>{{balance | currency}}</p>  
<p>{{description | uppercase}}</p>
```

- Stärkere Trennung von Wert und Darstellung
- Wiederverwendbar in der ganzen Anwendung
- Bündelt Verhalten

Apr 24, 2018

\$354.66

ANGULAR

Bereits vorhandene Pipes

Pipe Operator

Pipe: Die Ausgabe des linken Ausdrucks wird zur Eingabe des rechten.

## 2.2.4 Pipe: Eigene Funktionalität

ng g pipe dateformatter

Eine Pipe ist eine Klasse, welche eine einzige Methode zur Verfügung stellt.

dateformatter.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'dateformatter'
})
export class DateformatterPipe implements PipeTransform {

  transform(value: Date, args?: any): string {
    return value.toLocaleDateString('de-DE');
  }
}
```

Date
12.4.2018
12.4.2018
12.4.2018

List.component.html

```
...
<td>{{item.date | dateformatter}}</td>
...
```

## 2.2.4 Pipe Beispiel: Übersetzung

translate.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'translate'
})
export class TranslatePipe implements PipeTransform {
  private languageMap = {
    'en': {
      'error_1': 'Unknown error',
      'error_2': 'Server error'
    },
    'de': {
      'error_1': 'Unbekannter Fehler',
      'error_2': 'Server Fehler'
    }
  };

  private selectedLanguage = 'de';

  transform(value: string, args?: any): string {
    return this.languageMap[this.selectedLanguage][value];
  }
}
```

app.component.ts

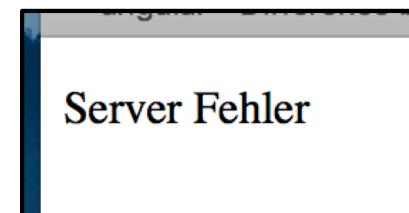
```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  errorMessage = 'error_2';
}
```

app.component.html

```
<p>{{errorMessage | translate}}</p>
```

Resultat



## 2.2.5 Lifecycle-Hooks (= Lifecycle Events)

- Komponenten (auch Services, Pipes, usw.) werden durch das Angular Framework erzeugt, verwaltet und zerstört
- Auf Events reagieren
  - ngOnChanges()
  - ngOnInit()
  - ngDoCheck()
  - ngAfterContentInit()
  - ngAfterContentChecked()
  - ngAfterViewInit()
  - ngAfterViewChecked()
  - ngOnDestroy()
- Ermöglicht organisiertes
  - Anmelden an Datenquellen
  - Starten von HTTP-Abfragen
  - Aufbauen von Sessions
  - Abbau von Sessions

<https://angular.io/guide/lifecycle-hooks#lifecycle-sequence>

## 2.2.5 Lifecycle-Hook: OnInit (vs. Konstruktor)

Properties initialisieren

Arbeit ausführen

```
import {Component, OnInit} from '@angular/core';
import {HttpClient} from '@angular/common/http';

export class AppComponent implements OnInit {
  data: string[];

  constructor(public httpClient: HttpClient) {
    this.data = [];
  }

  ngOnInit(): void {
    this.httpClient.get('http://localhost:8080/todos')
      .toPromise()
      .then((response: any[]) => {
        this.data = response as string[];
      });
  }
}
```

### implements OnInit

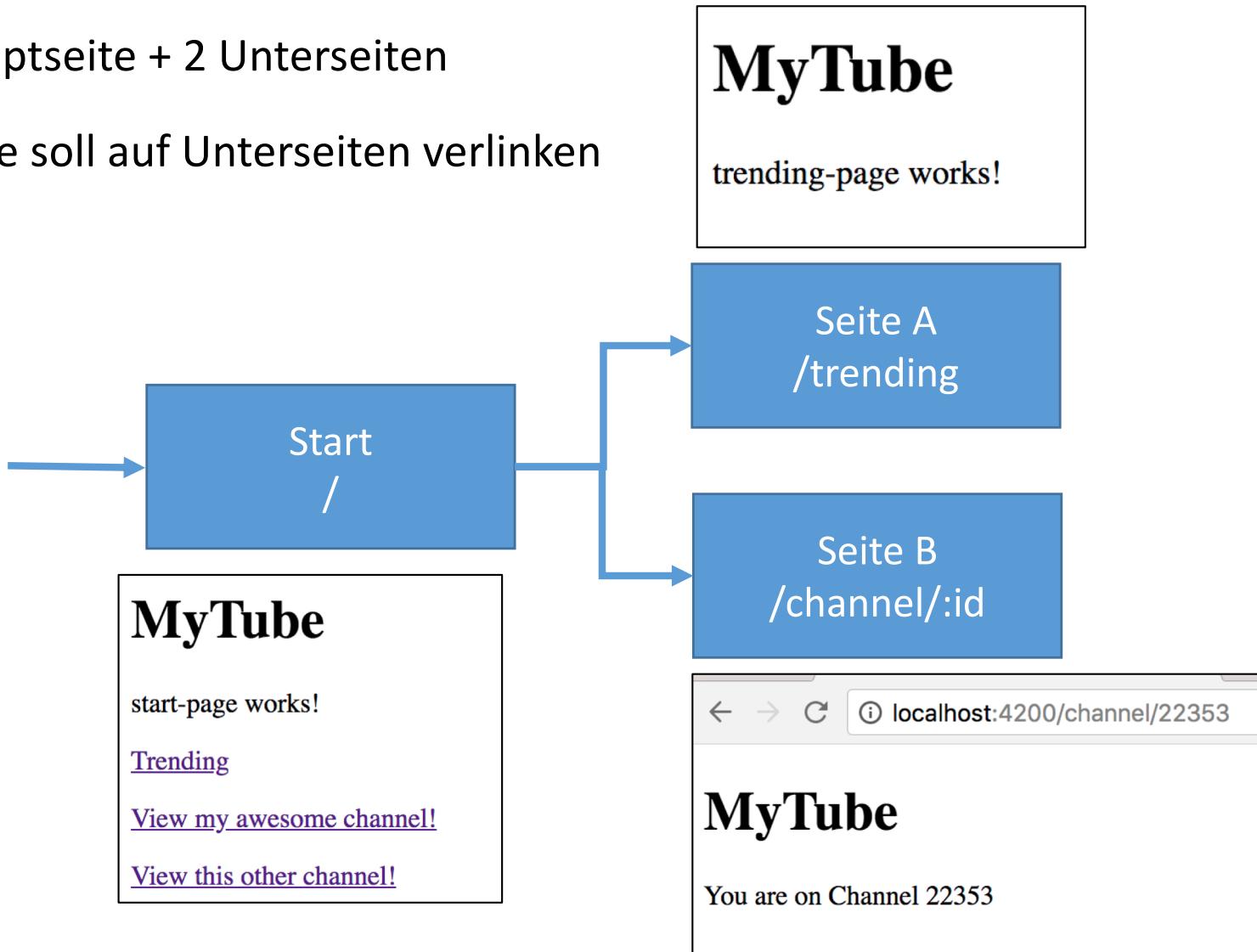
stellt sicher, dass die im Interface **OnInit** definierten Methoden durch die Klasse implementiert sind. (TypeScript Feature)

## 2.2.6 Routing

- Die in WBS, WebP1/2 entwickelten Anwendungen waren alle samt „Single Page Applikationen“.
- Typisch für Webanwendungen sind aber Links, die einen bestimmten Teil der Anwendung aufrufen.
- Beispiel Youtube:
  - Bestimmtes Video aufrufen: <https://www.youtube.com/watch?v=8ZtlnClXe1Q>
  - Trending aufrufen: <https://www.youtube.com/feed/trending>
  - Einen bestimmten Channel aufrufen: <https://www.youtube.com/channel/UC9-y-6csu5WGm29I7JiwpnA>
- Deep Linking: Ein Link triggert eine bestimmte Funktion der Anwendung (kann Single Page sein)
  - Beispiel: <http://localhost:8080/todo/56> soll den ToDo-Eintrag im Modalfenster anzeigen
  - Das heißt aber auch, dass sich beim Aufrufen des Modalfensters per Button, die angezeigte URL des Browsers ändern muss, damit der User den Link verschicken kann.

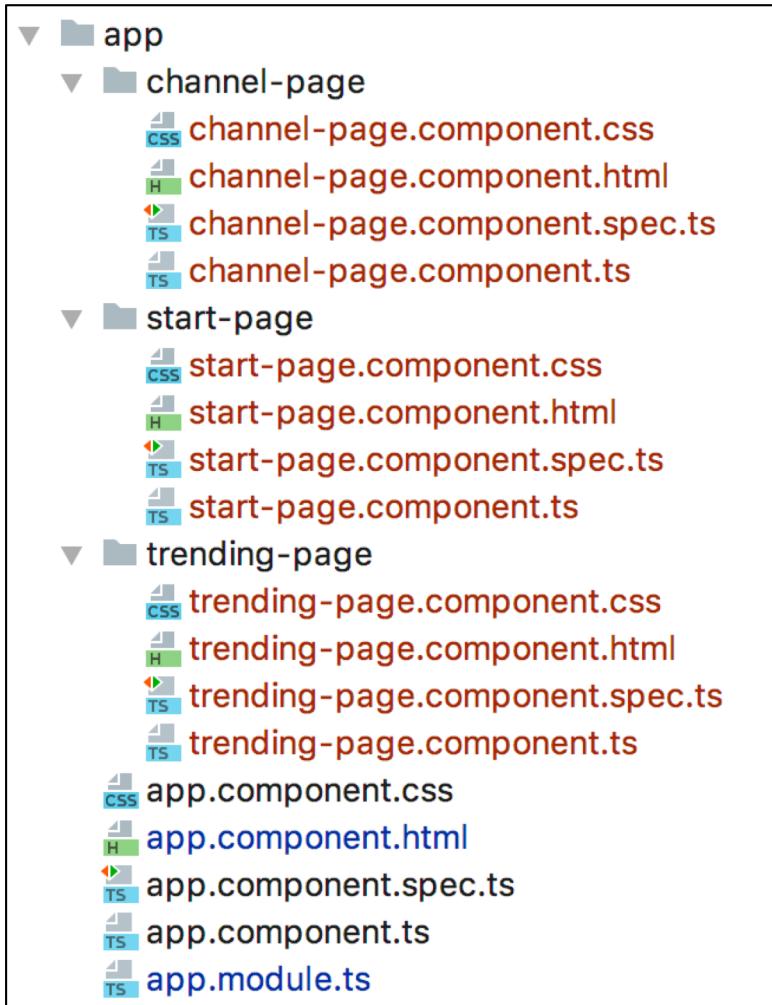
## 2.2.6 Routing: Klassisch – Aufbau eines Beispiels

- Ziel: Hauptseite + 2 Unterseiten
- Startseite soll auf Unterseiten verlinken



## 2.2.6 Routing: Klassisch – Seiten anlegen

- 1.: Für jede Seite (Seitentyp) eine Komponente anlegen



## 2.2.6 Routing: Klassisch - Konfiguration

app.module.ts

- 2.: Modul mit Routen konfigurieren

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

import { AppComponent } from './app.component';
import { RouterModule, Routes } from '@angular/router';
import { TrendingPageComponent } from './trending-page/trending-page.component';
import { ChannelPageComponent } from './channel-page/channel-page.component';
import { StartPageComponent } from './start-page/start-page.component';

const appRoutes: Routes = [
  { path: '', component: StartPageComponent },
  { path: 'trending', component: TrendingPageComponent },
  { path: 'channel/:id', component: ChannelPageComponent }
];

@NgModule({
  declarations: [
    AppComponent,
    TrendingPageComponent,
    ChannelPageComponent,
    StartPageComponent
  ],
  imports: [
    BrowserModule,
    RouterModule.forRoot(appRoutes)
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

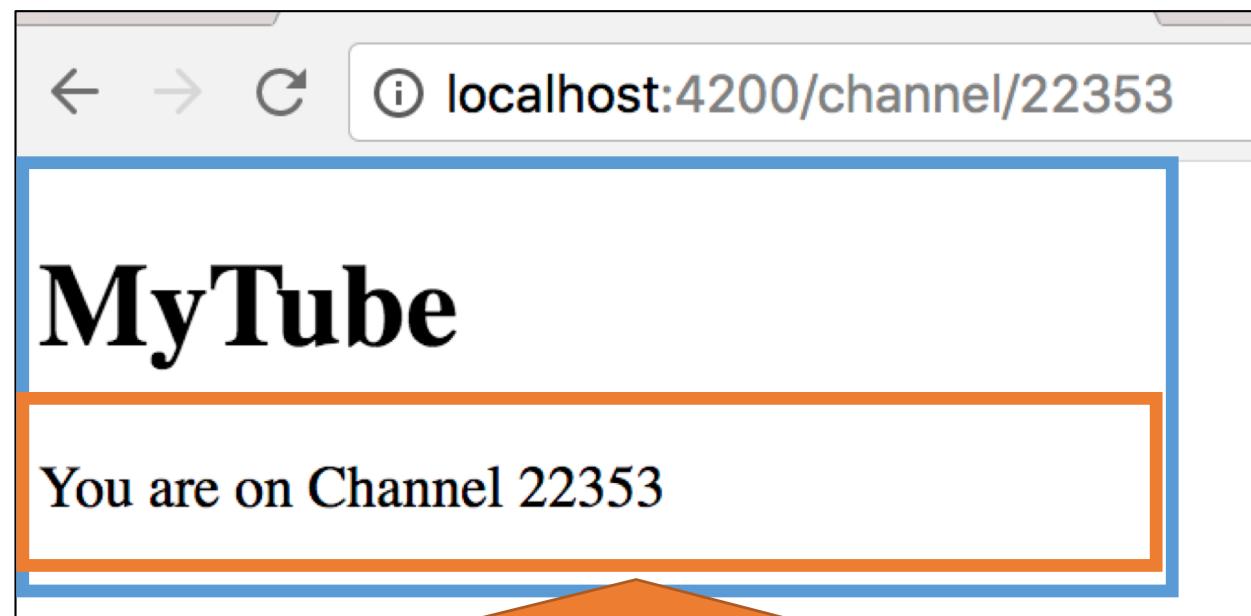
Pfade fangen  
nicht mit / an

## 2.2.6 Routing: Klassisch - Outlet

- 3.: Router Outlet

app.component.html

```
<h1>MyTube</h1>
<router-outlet></router-outlet>
```



app.component.html

Inhalt von `<router-outlet>` wird entsprechend der Browser URL ausgetauscht.  
(Hier beispielsweise mit channel-page.component)

## 2.2.6 Routing: Klassisch - Parameter

- 4.: Zugriff auf die Routing Parameter (Beispiel: Channel ID)

channel-page.component.ts

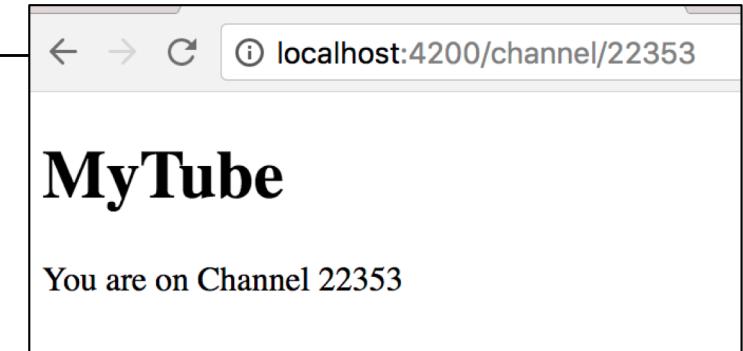
```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';

@Component({
  selector: 'app-channel-page',
  templateUrl: './channel-page.component.html',
  styleUrls: ['./channel-page.component.css']
})
export class ChannelPageComponent implements OnInit {

  channelID: string = 'unknown';

  constructor(private route: ActivatedRoute) {}

  ngOnInit() {
    this.route.params.subscribe((params: any) => {
      this.channelID = params['id'];
    });
  }
}
```



channel-page.component.html

```
<p>You are on Channel {{channelID}}</p>
```

'id' kommt aus der  
Routenkonfiguration in  
app.module.ts

## 2.2.6 Routing: Klassisch – Aufruf einer Route

- 5.: Route aufrufen

start-page.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';

@Component({
  selector: 'app-start-page',
  templateUrl: './start-page.component.html',
  styleUrls: ['./start-page.component.css']
})
export class StartPageComponent implements OnInit {

  constructor(private router: Router) { }

  buttonClicked() {
    this.router.navigate(['/trending']);
  }
}
```

buttonClicked()  
existiert nur zur Demo  
und wird nicht  
aufgerufen.

Die Seite kann programmatisch  
oder per klassischem Link  
gewechselt werden.

start-page.component.html

```
<p>
  start-page works!
</p>
<p><a [routerLink]="/trending">Trending</a></p>

<p><a [routerLink]="/channel/22353">View my awesome channel!</a></p>
<p><a [routerLink]="/channel/34235">View this other channel!</a></p>
```

## 2.2.6 Routing: Klassisch – Aufruf einer Route

Anwendungsfall Deep Linking:  
Öffnen eines Modalfensters führt zu einer Edit-Route.

start-page.component.ts

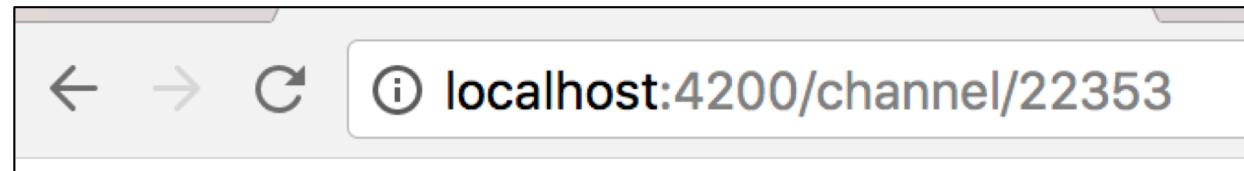
```
import { Component, OnInit } from '@angular/core';
import { Location } from '@angular/common';

@Component({
  selector: 'app-start-page',
  templateUrl: './start-page.component.html',
  styleUrls: ['./start-page.component.css']
})
export class StartPageComponent implements OnInit {

  constructor(private location: Location) { }

  buttonClicked() {
    this.location.go('/channel/22353');
  }
}
```

URL des Browserfensters ändern, ohne Seite neu zu laden.



## 2.2.6 Routing: Klassisch – Ablauf

- 1) Aufruf von <http://localhost:4200/channel/22353> durch den Browser
- 2) app.component wird dargestellt
- 3) Das Routing System liest die Browser URL
- 4) Das Routing System tauscht das Router-Outlet aus mit der entsprechenden Komponente
- 5) OnInit() der neuen Komponente wird durch das Angular System aufgerufen
- 6) Die Komponente holt sich die URL Parameter aus dem Routing Service

## 2.2.7 Angular Debugging: Breakpoints

Chrome Browser!

The screenshot shows the Chrome DevTools interface with the Sources tab selected (marked with a red box and the number 1). The left sidebar displays the project's file structure:

- top
- localhost:4200
  - (index)
  - inline.bundle.js
  - main.bundle.js
  - polyfills.bundle.js
  - styles.bundle.js
  - vendor.bundle.js
- (no domain)
- webpack://
  - (webpack-dev-server)
  - (webpack)/buildin
- webpack:///.
  - node\_modules
  - src
    - app
      - list
        - list.component.html
        - list.component.ts

The list.component.ts file is selected (marked with a blue bar and the number 3). The code editor shows the following TypeScript code:

```
import {AfterViewInit, Component, OnChanges, OnInit} from '@angular/core';
import {ToDoEntry} from '../ToDoEntry';

@Component({
  selector: 'app-list',
  templateUrl: './list.component.html',
  styleUrls: ['./list.component.css']
})
export class ListComponent implements AfterViewInit {

  public todoList: ToDoEntry[] = [
    new ToDoEntry('Milch'),
    new ToDoEntry('Butter'),
    new ToDoEntry('Brot')
  ];

  constructor() {}

  ngAfterViewInit() {
    console.log('Ich wurde instanziert!');
  }

  done(index: number): void {
    this.todoList.splice(index, 1);
  }
}

// WEBPACK FOOTER //
// ./src/app/list/list.component.ts
```

The right panel shows the developer tools' sidebar with the following sections:

- Paused on breakpoint
- Watch
- Call Stack
- Scope
- Local
  - this: ListComponent
    - todoList: Array(3)
      - 0: ToDoEntry {title: "Milch", date: Thu}
      - 1: ToDoEntry {title: "Butter", date: Thu}
      - 2: ToDoEntry {title: "Brot", date: Thu}
    - length: 3
    - \_\_proto\_\_: Array(0)
    - \_\_proto\_\_: Object
- Closure
- Closure (../src/app/list/list.component.ts)
- Global
- Breakpoints
  - list.component.ts:20  
console.log('Ich wurde instanziert!');
  - XHR/fetch Breakpoints
  - DOM Breakpoints
  - Global Listeners
  - Event Listener Breakpoints

## 2.2.7 Angular Debugging: Augury

Chrome Browser!

Installieren

<https://augury.angular.io/>



Oder Chrome Web Store

<https://chrome.google.com/webstore/detail/augury/elgalmkoelokbchhkacckoklkejnhcd>

The screenshot shows the Augury extension page on the Chrome Web Store. At the top, there's a logo of a puzzle piece with a yellow crescent moon, the word "Augury", and "angeboten von Rangle.io". Below that are ratings (4.5 stars), developer tools, and user count (182,120 Nutzer). The main interface has tabs for "ÜBERSICHT", "MEINUNGEN", "SUPPORT", and "ÄHNLICHE". The "ÜBERSICHT" tab is active, showing a component tree on the left and a properties/injector graph on the right. The component tree lists various Angular components like NgModel, Router, TodoApp, and MetadataTest. The properties/injector graph shows bindings between components, such as @Input() count: 0 and @Output() result: Huzzah!. On the right side, there's a compatibility note ("Mit Ihrem Gerät kompatibel"), a description of what Augury does (extending developer tools for debugging and profiling Angular apps), a changelog, and links to the website and reporting abuse.

## 2.2.7 Angular Debugging: Augury

Chrome Browser!

1

The screenshot shows the Chrome DevTools interface with the Augury tab selected (highlighted by a red box). A consent dialog is displayed at the top: "We would like to gather some data to help improve Augury but first we need your consent. Do you agree to let us collect your usage information? Yes No". Below the dialog, the Augury component tree shows "AppComponent" expanded, with "ListComponent" selected (highlighted by a purple bar). The "Properties" tab is active, displaying the state of "ListComponent": "Change Detection: Default", "State": "todoList: Array[3]", containing three "ToDoEntry" objects. The "Injector Graph" tab is also visible.

We would like to gather some data to help improve Augury but first we need your consent. Do you agree to let us collect your usage information?

Yes   No

AppComponent

ListComponent

Properties   Injector Graph

ListComponent [\(View Source\)](#)   (\$\$el in Console)

Change Detection: Default

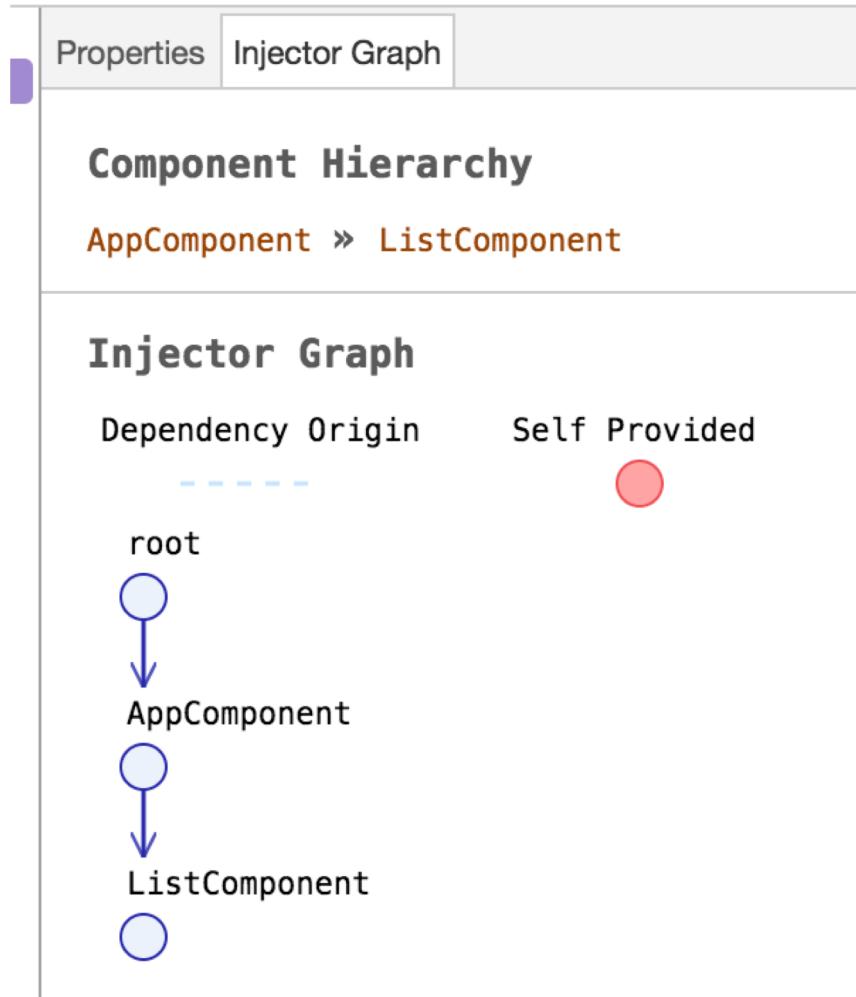
▼ State

▼ todoList: Array[3]

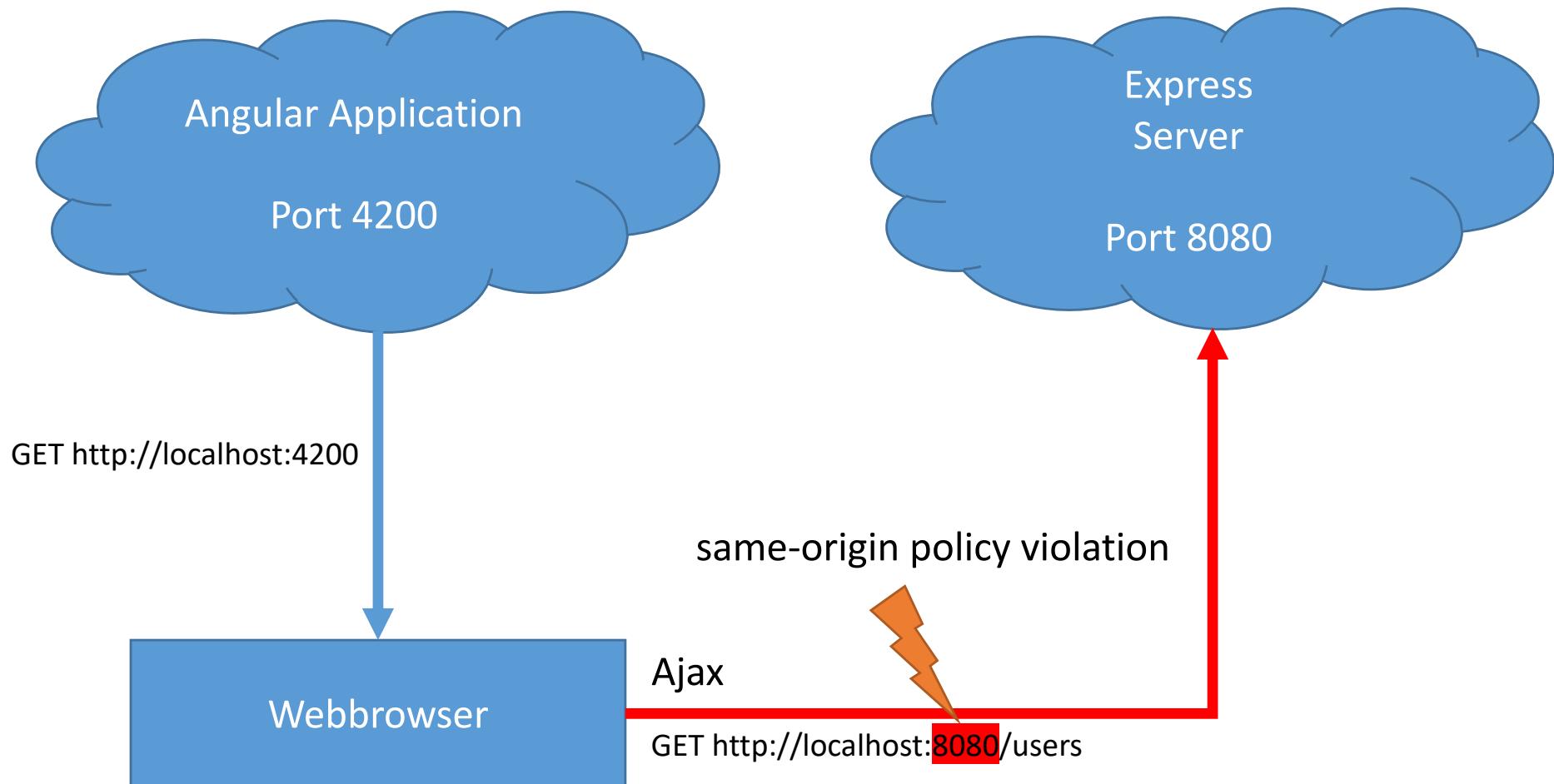
- ▶ 0: ToDoEntry
- ▶ 1: ToDoEntry
- ▶ 2: ToDoEntry

## 2.2.7 Angular Debugging: Augury

Chrome Browser!

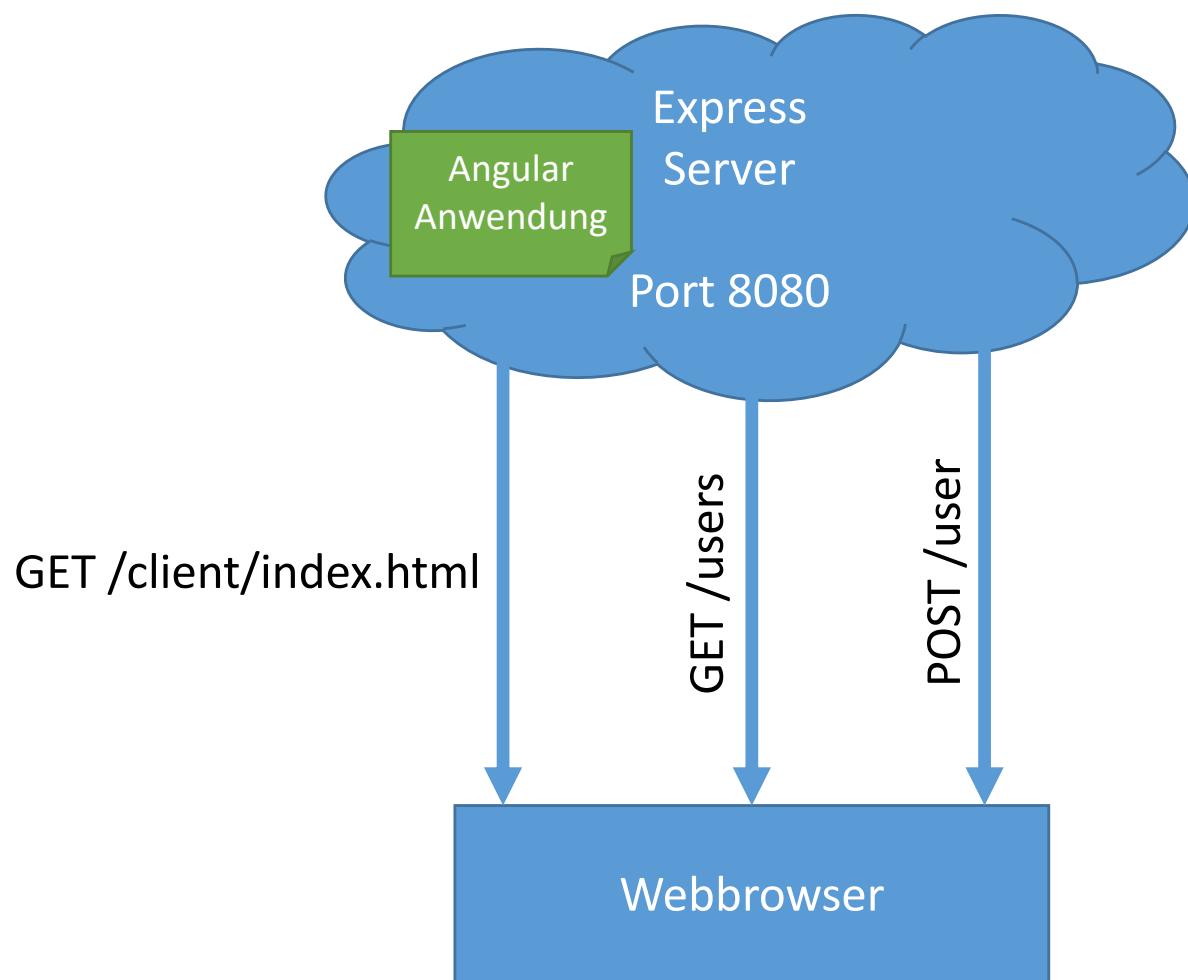


## 2.2.8 Client + Server: Infrastruktur



Nur Ajax ist explizit verboten. Okay sind: Bilder, Stylesheets, Scripte, iframes und Videos.  
Schutz vor Cross-Site-Scripting: <https://de.wikipedia.org/wiki/Cross-Site-Scripting>

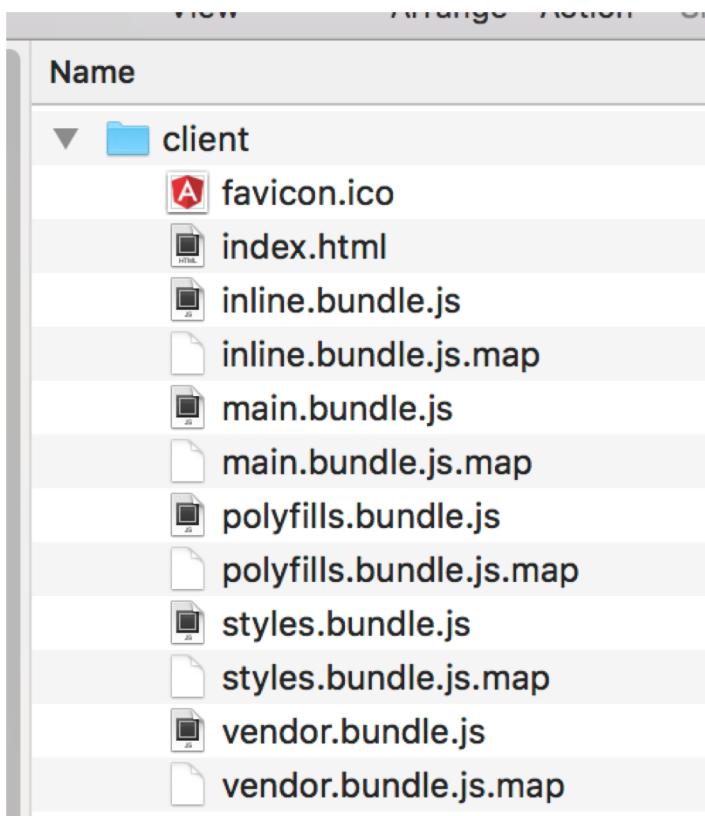
## 2.2.8 Client + Server: Infrastruktur



Lösung der same-origin policy violation:  
Die Angular Anwendung wird per static-Route vom Server ausgeliefert.

## 2.2.8 Client + Server: Angular Build

```
ng build --op ./client
```



... erzeugt eine Standalone-Webseite, welche durch einen Server ausgeliefert werden kann (z.B. Apache oder ein eigener Express Server mit static-Route).

ng build: „Als klassische HTML+JavaScript Seite exportieren.“

--op

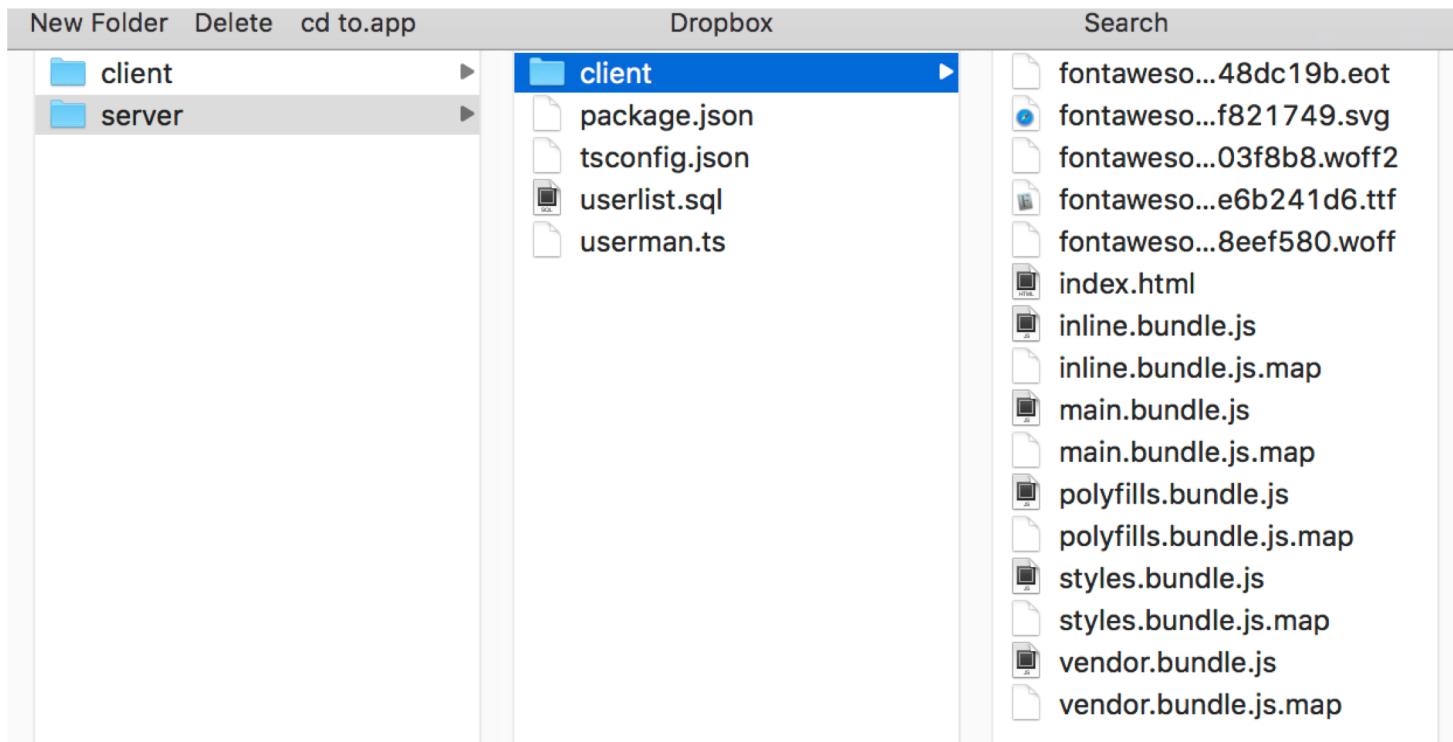
Output Path, wie immer bezieht sich der Punkt im Pfad auf das Verzeichnis, aus dem der Befehl aufgerufen wurde.  
(Webstorm Terminal => Projektverzeichnis)

## 2.2.8 Build-Instruktion zur (Haus)übung

In den Server „hinein-builden“

Im Webstorm Terminal des Clients:

```
ng build --op ../server/client --watch
```



Build Befehl bleibt  
aktiv und baut bei  
Änderungen  
automatisch neu.

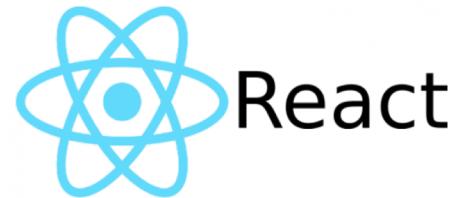
```
router.use("/", express.static(__dirname + "/client"));
```

## 2.2.9 Ausblick

Objektorientiert



Funktional / Virtual DOM



Convention over Configuration



Progressive Extensions

# Übung: Komponenten, Service, Wiederverwendbarkeit

Bauen Sie eine Angular-Anwendung, passend zum vorgegebenen Server:

- Strukturieren Sie Ihre ToDo-Anwendung in folgende Komponenten:
  - NavBar (obere Leiste)
  - „New element“-Leiste (Enthält den Button, der das Modalfenster öffnet)
  - Listen-Komponente (Soll benutzt werden für sowohl die ToDo-Liste, als auch die Done-Liste)
  - Optional: Zeilen-Komponente (Stellt die Zeilen der Tabellen dar)
- Lagern Sie die Daten in einen eigenen Service aus, der mit HTTP mit dem Server kommuniziert
- Die ToDo-Liste und die Done-Liste sollen jeweils Instanzen der Komponente list.component sein.
  - Das heißt: Die Komponente muss generisch programmiert werden und sich je nach Verwendung (ToDo oder Done) anders verhalten.
    - Tipp: Übergeben Sie der Instanz nur die Informationen, die für sie relevant sind.
- Optional: Implementieren Sie Routing zu den einzelnen Komponenten

## 2.2 Angular: Kompetenz (Teil 2)

- Komponentenarchitektur
  - Sie wissen, wie sich eine Anwendung in Komponenten aufbauen lässt.
  - Sie kennen die Möglichkeiten, direkt Daten zwischen Komponenten auszutauschen.
    - @Input(), @Output()
  - Sie verstehen den wasserfallartigen Datenfluss und können diesen informell beschreiben.
- Services
  - Sie erkennen den Grund für Services und können diese sauber umsetzen.
- HTTP
  - Sie können HTTP Requests mit unterschiedlichen HTTP-Methoden und entsprechenden Parametern absetzen.
  - Sie können die angegebene offizielle Dokumentation konsultieren und dort Informationen zu anderen HTTP-Methoden nutzen.
- Pipes
  - Sie kennen wichtige, bereits vorhandene Pipes und können diese verwenden.
  - Sie können eigene Pipes implementieren.
- Lifecycle-Hooks
  - Sie kennen die wichtigsten Events (Bsp. ngOnInit()) und können auf diese reagieren.
- Routing
  - Sie können eine Angularanwendung in Seiten aufteilen und diese miteinander verlinken.
  - Sie können auf URL Parameter reagieren.
- Debugging
  - Sie können eine Angularanwendung debuggen und den Wert von Variablen einsehen.
  - Sie können sich die Komponentenstruktur anzeigen lassen.
- Build
  - Sie können eine Angularanwendung bauen lassen und durch einen Server ausliefern lassen.
  - Sie können an einer Angularanwendung entwickeln, während diese durch einen eigenen Server ausgeliefert wird.

**Ende VL 4**