Angular Grundlagen

von: **Saban Ünlü**

für: **PTA**

Zwei Worte zu mir …

Saban Ünlü

- Software Architekt und Programmierer
- Berater und Dozent seit 2000
- Autor
- Influencer
- Gründer von netTrek

Einleitung

Einleitung

- Was ist Angular?
- Angular: Die Highlights
- Was ist neu?
- Unterschiede zu AngularJS 1x
- Technologien
- Polyfills und Vendors
- Angular-Module

Was ist Angular?

- Framework für Single Page Application
- Komponentenbasierte Anwendungen
  – inspiriert von Web-Components
- Modulare Arbeitsweise
- Trennung von Logik und View

Angular: Die Highlights

- Vorlagen
- Bindungen
- Services
- Dependency Injection
- Routing
- Formulare

Was ist neu?

- Angular ist kein klassisches Update
- Komplett neu geschrieben
- Fokus
  - Performance (3-5 mal schneller als Angular 1)
  - Komponenten
  - Modulare Arbeitsweise

Unterschiede zu AngularJS 1x

- Komponenten statt Kontroller
  Wie Element-Direktiven in Angular 1

- Verzicht auf Scope
  Bindung erfolgt auf Komponenten-Instanz

Unterschiede zu AngularJS 1x

- Entwicklung mit ES5, Dart oder TypeScript
- Angular wurde mit TypeScript entwickelt
    - ES2015: Klassen, Interfaces und Vererbung
    - ES2015: Templates
    - Typisiert
    - Annotations

Unterschiede zu AngularJS 1x

- Angular ist mit TypeScript ES2015 konform
- TypeScript wird für aktuelle Browser in ES5 ausgegeben
- ES2015 Polyfills für “weniger moderne” Browser
- System zum Laden und Verwalten von Modulen z.B.:
    - System.js
    - webpack

Technologien - Überblick

core js Webpack RxJS reflect-metadata

Technologie

- JavaScript-Laufzeitumgebung für diverse Betriebssysteme
- Versionierungssystem für Software (GitHub – Filehoster)
- Auf ES2015 basierende Programmiersprache

Polyfills

- core-js
  - ES2015/ES6 Polyfills
- web-animations-js
  - Firefox Animationen
- intl (ng4 for ng5+ use locales)
  - I18n für Internationalisierung

Vendors

- RxJS
  - Bibliothek, um Ereignisse und asynchrone Prozesse zu überwachen. Wird für HTTP-Aufrufe von Angular genutzt.
- zone.js
  - Ähnlich Domains in Node: Ausführungskontext ermöglicht, die Ausführung zu überwachen und zu steuern.

Vendors

- reflect-metadata
  - Metadaten in konsistenter Weise zu einer Klasse hinzufügen
- ~~systemjs~~
  - ~~Modul Loader für ES2015/ES6 Module~~

Angular Module

- @angular/core
  - Notwendig für jede Anwendung – Kern für Komponenten, Direktiven, Dependency Injection und Komponentenlebenszyklus
- @angular/common
  - Häufig verwendete Direktiven, Pipes und Services

Angular Module

- @angular/compiler
  - Kombiniert Logik mit Vorlagen – Compiler wird automatisch über platform-browser-dynamic angestoßen
- @angular/platform-browser
  - Browser und DOM-relevante Bestandteile, vor allem zum Rendern neuer Elemente

Angular Module

- @angular/platform-browser-dynamic
  - Verfügt über die Bootstrapping-Methode
- @angular/http
  - Modul für HTTP-Aufrufe
- @angular/router
  - Module für den Komponenten-Router

Angular Module

- @angular/animate
  - Animationen im Angular-Kontext

TypeScript excursus

TypeScript excursus

- var, let, const
  - types
    - native, class, interface, own
- arrow function. ()=>{}
  - scope

TypeScript excursus

- parameter
  - default (param: boolean = true)
  - optional (param?: boolean)
  - rest (…rest)

TypeScript excursus

- class ES5 vs TypeScript
  - extends
  - interfaces
  - abstract class

TypeScript excursus

- Syntax magic (ES6/TS)
  - private, public definition in constructor
  - Concat Array
  - Object Assign
  - Destructing

Projektsetup

Erste Schritte

- Mac
  - XCODE installieren
  - node.js installieren (>= 6.9.x)
- Win
  - node.js installieren (>= 6.9.x)
  - Git installieren (inkl. Bash)

Setup Manuell

- Node initialisieren
- Abhängigkeiten installieren
- TypeScript konfigurieren
- Webpack konfigurieren

Seed Setup

- git clone https://github.com/angular/quickstart.git myProject
- npm install

angular-cli

- ng new pta --prefix= pta --style=scss --routing=true
- ng serve
- ng g m commonUi
- cd common-ui/
- ng g c user

Architektur

Architektur

- Module
- Komponenten
- Bootstrap
- Direktiven
- Pipes
- Datenbindung
- Dependency Injection (DI)
- Services
- Router

Module

- Nicht vergleichbar mit JavaScript-Modulen
- Funktionen und Features in einer Black-Box bündeln
- Anwendung und eigene Module mit externen Modulen erweitern
- Compiler mitteilen, nach welchen Elementen auszuschauen ist

Module

Angular-eigene Module

- BrowserModule (Ereignisse, DOM)
- CommonModule (Direktiven, Pipes)
- HttpModule (XHR)
- FormsModule (Formulare)
- RouterModule (Komponenten-Router)

Module

Module erzeugen

- Modul-Klasse anlegen

Module

```
class AppModule {}
```

Module

```
@NgModule({
  imports:      [ BrowserModule ]
})
export class AppModule {}
```

Module

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

Pipes

Angular Module

- ng g m commonUi in src/app
- @NgModule
  - imports
    - definiert Module die in diesem Modul benötigt werden
  - declarations
    - benötigte Komponenten, Direktiven, Pipes

Angular Module

- @NgModule
  - providers
    - Bestimmt welche Service der Injector dieses Moduls für die DI bereitstellt.
  - exports
    - Exportiert Komponenten, Direktiven, Pipes dieses Moduls damit importierende Module das nutzen

Angular Module

- @NgModule
  - bootstrap
    - Komponenten, die beim Bootstrap dieses Moduls in den ComponentFactoryResolver abgelegt werden. Analog - entryComponents

Angular Module

- @NgModule
  - entryComponents
    - Kompiliert Komponenten bei der Definition des Moduls. Anschließend ist die Nutzung ohne Komponente-Kontext möglich, weil es als ComponentFactory und die componentFactoryResolver abgelegt wird.

Komponenten

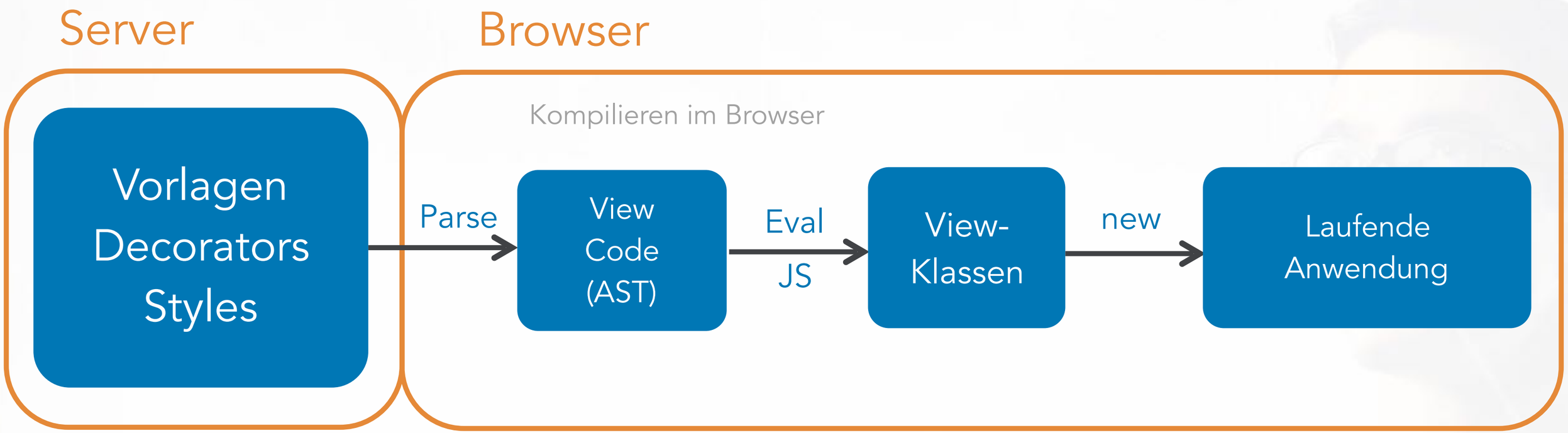
Komponenten

- Decorator und Metadaten
- Angular Module
- Bootstrap Root-Component
- Bootstrap eine Modules
- Selector
- Vorlagen
- Styling
- Komponenten verschachteln (Shared-Modules)
- ng-content
- ViewChilds
- Lifecycle hook

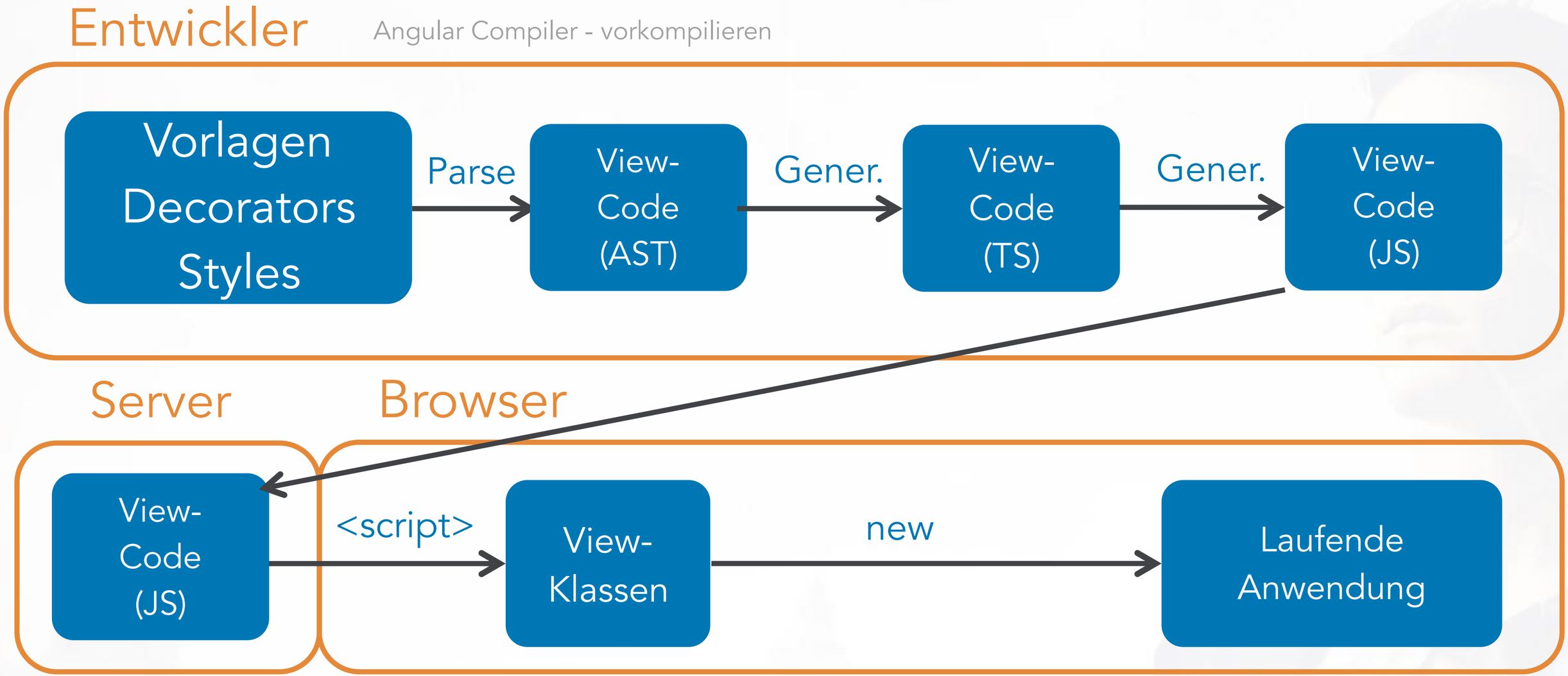
Komponenten

- Eigene HTML Knoten
- Bestandteile
  - Vorlage
  - Style
  - Logik

JIT


Server
Browser
Vorlagen
Decorators
Styles
Kompilieren im Browser
Parse
View
Code
(AST)
Eval
JS
View-
Klassen
new
Laufende
Anwendung

AOT


Entwickler
Angular Compiler - vorkompilieren
Vorlagen
Decorators
Styles
Parse
View-
Code
(AST)
Gener.
View-
Code
(TS)
Gener.
View-
Code
(JS)
Server
Browser
View-
Code
(JS)
<script>
View-
Klassen
new
Laufende
Anwendung

Komponenten

```
class AppComponent {

    constructor () {
        console.log ( "App Component" );
    }

}
```

Komponenten

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

Komponenten

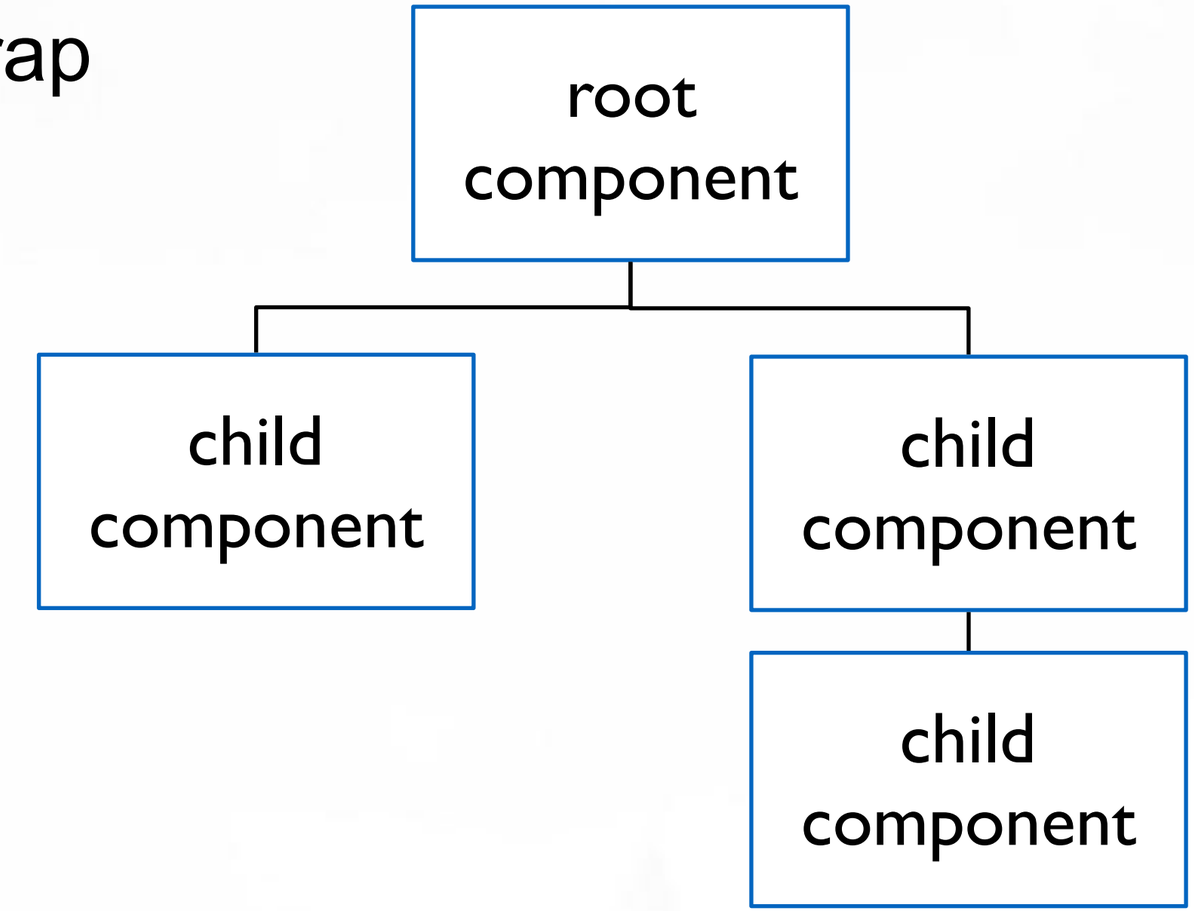
<h1 (click)="onClick()">{{name}}</h1>
<my-component> </my-component>

Decorator und Metadaten

- Dekoratoren bereichern eine Klasse mit Metadaten
- Vor Klassendefinition
- Funktionsaufruf beginnend mit einem @NAME
- Parameter
  - Metadaten

Komponentenbasierte Anwendung

- bootstrap



Bootstrap

- in der main.ts
- platformBrowserDynamic
  - bootstrapModule
    - AppModule
      - bootstrap der Komponenten

Bootstrap

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

Komponent Metadaten

- ng g c user
  - selector
    - Knoten
  - Vorlage
    - templateUrl (file)
    - template (backticks)

Komponent Metadaten

- Style
  - styleUrls (filelist)
  - styles (backtick-list)
- Spezieller Style
  - :host
  - >>>

Komponent Metadaten

- Style
  - encapsulation - Umgang mit Webkomponenten
    - ViewEncapsulation.Emulated
    - ViewEncapsulation.Native
    - ViewEncapsulation.None

Komponent Lebenszyklus

- constructor
- ngOnChanges
- ngOnInit
- ngDoCheck
  - ngAfterContentInit
  - ngAfterContentChecked
  - ngAfterViewInit
  - ngAfterViewChecked
- ngOnDestroy

```
export class UserListComponent

<userList [data]=“userList">

<userList>

<user> </user>

    <user> </user>

</userList>
```

UserListVorlage

Komponent - Content

- Inhalte Transklusieren (transclude)
  - ng-content
    - Knoten in Vorlage
    - Attribut
      - select="nt-table-caption"

Komponent Content

- @ContentChild
  - Kind-Komponente einer Komponente erreichen
    - Parameter
      - Komponentenklasse
  - ngAfterContentInit
    - Hook ab wann der Wert erreichbar ist.

Komponent Content

- @ContentChildren
  - Analog zu ContentChild -> QueryList<Komponenten>

Komponent View

- @ViewChild
  - Knoten einer Komponente erreichen, die in der Vorlage definiert wurden.
    - Parameter
      - Komponentenklasse
      - Referenz (string) zu einer mit #NAME ausgezeichneten Knoten

Komponent View

- @ViewChild
  - ngAfterViewInit
    - Hook ab wann der Wert erreichbar ist.

Komponent Content

- @ViewChildren
  - Analog zu ViewChild -> QueryList<Komponenten>

Bindung

Bindung

- Ausdrücke interpolieren
- Eigenschaften binden
- Style-Eigenschaften binden
- CSS-Klassen binden
- Attribute binden
- Ereignisse binden
- Komponenten-Eigenschaften
- Komponenten-Ereignisse
- HostBinding

Ausdrücke interpolieren

- Ausdruck in geschweiften Klammern
  - {{ AUSDRUCK }}
- Erlaubte Ausdrücke
  - Eigenschaften, Zeichenketten, Operatoren
  - Methodenrückgabe
    - mit bedacht nutzen

Eigenschaften

- Erlaubt Zuweisung über Eigenschaften eines HTML-Elementes
- [EIGENSCHAFT]=„AUSDRUCK“
- Erlaubte Ausdrücke
  - Eigenschaften, Zeichenketten, Operatoren
  - Methodenrückgabe

Attribute

- Erlaubt Zuweisung über Knoten-Attribute eines HTML-Elementes
- [attr.EIGENSCHAFT]=„AUSDRUCK“
- Erlaubte Ausdrücke
  - Eigenschaften, Zeichenketten, Operatoren
  - Methodenrückgabe

Styles

- Erlaubt Zuweisung über StyleEigenschaften eines HTML-Elementes
- [style.EIGENSCHAFT.EINHEIT]=„AUSDRUCK“
- Erlaubte Ausdrücke
  - Eigenschaften, Zeichenketten, Operatoren
  - Methodenrückgabe

Class

- Erlaubt styling über CSS-Klassen
  - [class.KLASSENNAME]=„BOOL-AUSDRUCK“
  - [class]=„AUSDRUCK“
- Erlaubte Ausdrücke
  - Eigenschaften, Zeichenketten, Operatoren
  - Methodenrückgabe

Ereignis

- Erlaubt Bindung an native Ereignisse eines HTML-Elementes
  - (EVENT)=„METHODE($PARAM)“
- Parameter
  - $event -> reicht Ereignis durch
- Beispiel
  - (click)=„clickHandler($event)

Eltern-Kind-Kommunikation

Eltern-Komponente

```
export class UserListComponent {
    userList: User[];
    selectUser (user: User) {}
}
```

```
<user
    [userData]="userList[0]"
    (onSelect)="selectUser($event)"
>
```

Kind-Komponente

```
export class UserComponent {
    @Input() userData: User;
    @Output() onSelect: EventEmitter;
}
```

Eigenschaft

- Komponenteneigenschaften lassen sich über den Eigenschaftsdekorator anlegen
  - @Input (OPT_ATTR_NAME) NAME : Type
- Auch für Setter nutzbar
- Beispiel
  - @Input('selected-usr') selectedInd: number = 1;
  - <comp [selected-usr]=„2" ….

Ereignisse

- Komponentenereignisse lassen sich über den Eigenschaftsdekorator anlegen
  - @Output (OPT_ATTR_NAME) NAME : EventEmitter
- Auch für Getter nutzbar
- EventEmitter sendet Wert via emit
- Wenn dem NAME der Ausdruck Change folgt ist eine Bidirektionale Bindung möglich. (**vermeiden**)

HostBindings- und Listender

- Mittels Eigenschaftsdekorator lassen sich auch Bindungen direkt in der Komponentenklasse definieren
  - @HostBinding (bind) NAME : boolean = true
  - @HostListener (EVT_NAME, [‚$event‘]) HANDLER : Function = (evt)=>{}

Direktiven

Direktiven

- Definition
- Hauseigenen
  - ngIf
  - ngFor
  - ngClass und ngStyle
- Eigene Direktiven

Direktiven

- Direktiven sind „Komponenten ohne Vorlagen“ und werden als Attribut verwendet
- Typen
  - Strukturell
    - Modifiziert DOM
  - Attribute
    - Modifiziert die Funktionalität

Direktiven

- Selector bestimmt, wie Direktiven angewandt werden
  - Attribut <div selector …
  - Mit Bindung <div [selector]=“wert”
  - Mit Zuweisung <div selector=“wert”
  - Klasse <div class=“selector“ …

Strukturelle Direktiven

- modifizieren den DOM
- sind mit Asterix * oder über einen Template-Knoten Nutzbar
  - [ngIf]=„AUSDRUCK“
    - Hängt den Knoten aus dem DOM wenn der Ausdruck false ist

Strukturelle Direktiven

- [ngFor]=„AUSDRUCK“
  - Wiederholt den Knoten anhand einer Iteration
  - Ausdruck
    - Beschreibt Iterator und kann zusätzliche Werte durchreichen
      - index, first, last, even, odd

Attribute Direktiven

- [ngClass]=„AUSDRUCK“
- [ngStyle]=„AUSDRUCK“
  - Erweitert style und class Attribut eines Knotens

Direktive erstellen

- @Directive
  - selector
    - Attribut z.B. [‚myDirective‘]
    - Klasse z.B. ‚.my-class‘ (auch als Liste)
  - class optional mit DI von ElementRef
    - nativeElement - Refferenziert dann das Element

Pipes

Pipes

- Definition
- Hauseigene Pipes
- Pipes benutzen
- Eigene Pipes erstellen

Pipes

- Modifiziert die Ausgabe
- Syntax
  - Ausdruck | PipeName : Parameter
- Beispiel
  - {{name | uppercase}}

Pipes

- Hauseigene
  - Uppercase
  - Lowercase
  - Date
  - …

Pipes

- Verwenden
  - Ausdruck | PipeName : Parameter
- Beispiel: {{name | uppercase}}
- Im Code - nach Provide
  - const reversePipe : ReversePipe = new ReversePipe ();
    console.info (reversePipe.transform(123));

Pipes

- Erstellen
  - @Pipe
    - name: string
- class NAME implements PipeTransform
  - transform(value: any, args?: any): any {

rxjs

Observer / Observable

- Observable
  - Iterierbares Objekt, welches filter- und registrierbar ist, um async. Prozesse zu verfolgen
- Observer
  - Objekt das neue Werte pushen
    - In Form eines Subject Quelle des Observable
      - asObservable();

Subscription

- an Observable
  - next
  - error
  - complete
- unsubscribe
- siehe: http://rxmarbles.com/

Beispiel

- observable = Observable.range (1, 5);
- observable.subscribe (
  - next => console.info ('next %s', next),
  - error => console.info ('error %s', error),
  - () => console.info ('complete')
-)

Service

Service

- Definition
- DI
- Hauseigene Service
- Einbinden
- Erstellen
- Injectable
- HTTP-Service

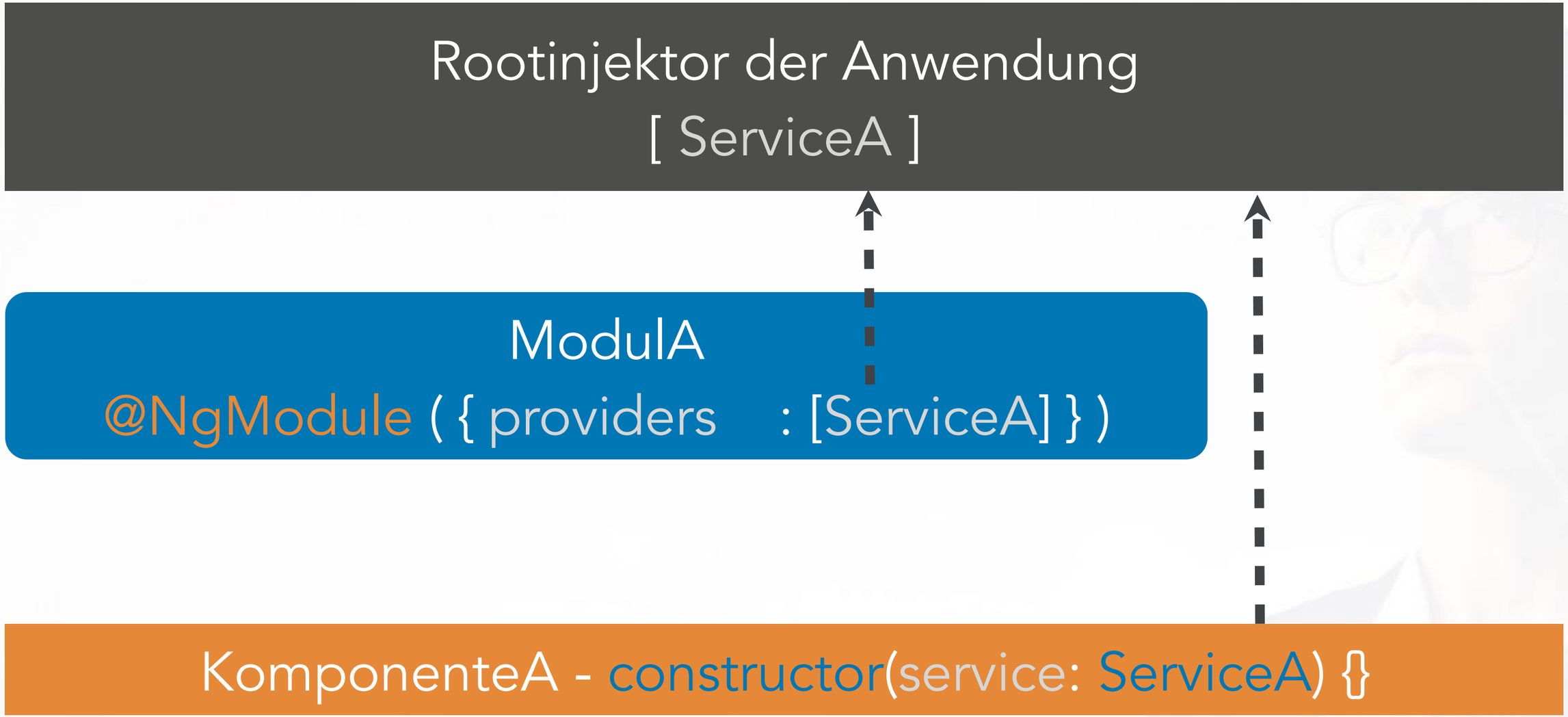
Service

- View-unabhängige Business-Logik
- Keine Angular-Spezifikationen
  - Außer bei Service-Konstruktoren
- Bereitstellung über DI

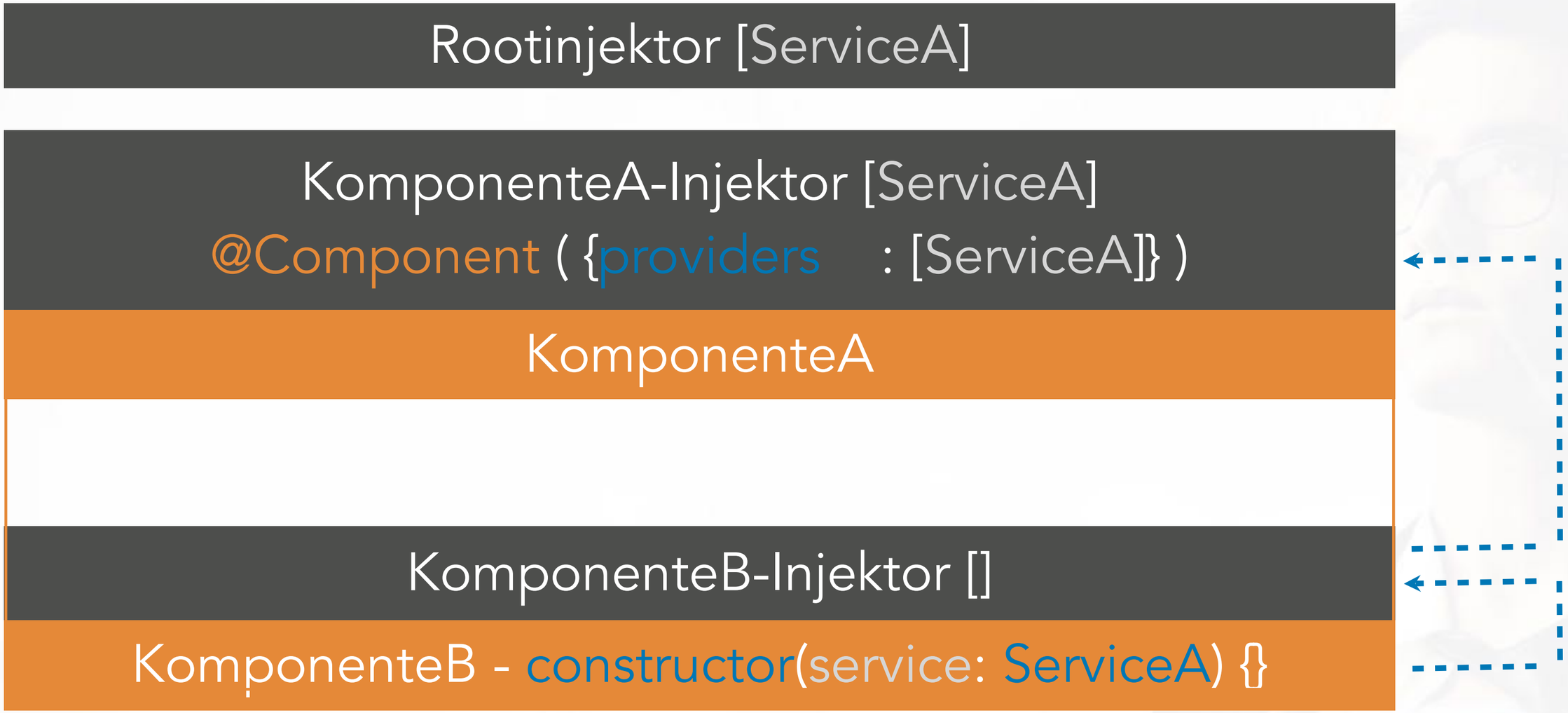
Erstellen

- Klasse erzeugen
- Injectable verwenden falls Konstruktor DI benötigt
  - seit ng6 -> Injector Angabe
- In Provider einbinden

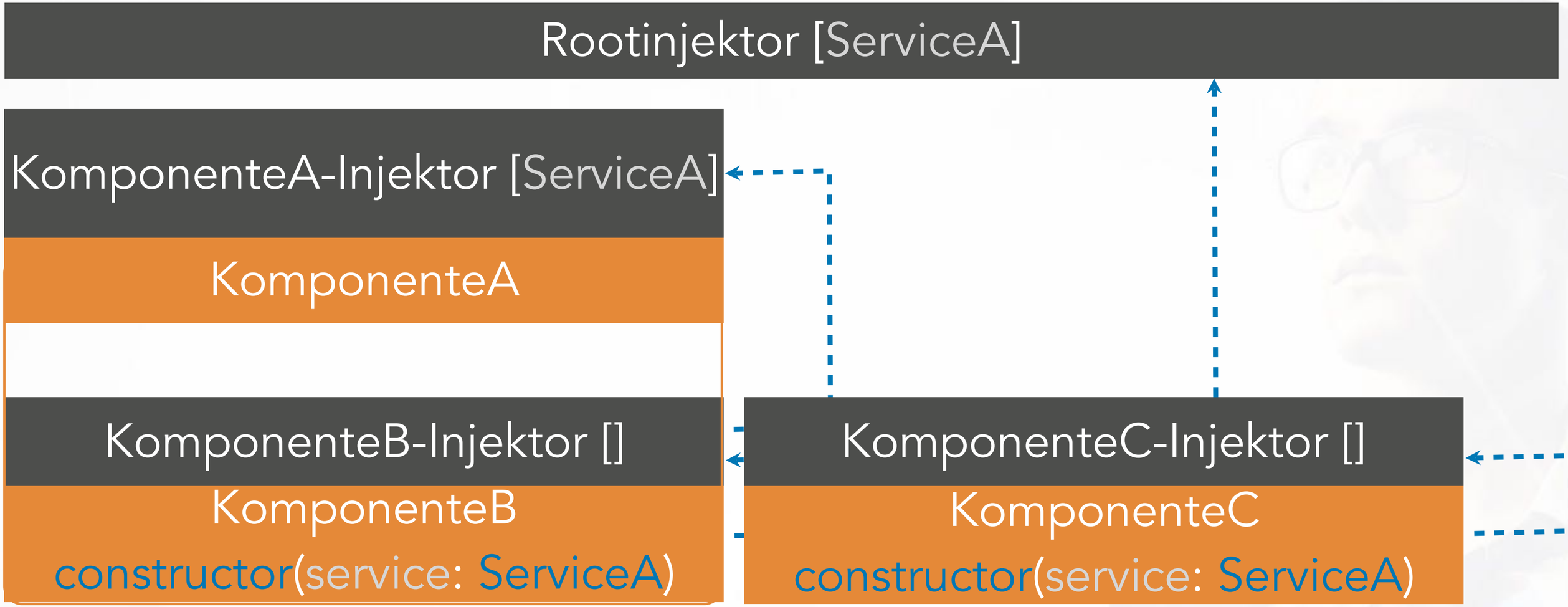
Dependency Injection (DI)



Dependency Injection



Dependency Injection



Dependency Injection (DI)

app.component.ts:19 -> http Injected Http {}

Dependency Injection (DI)

```
@NgModule({
  imports:      [ BrowserModule, HttpModule  ],
  providers:    [ MyService ]
})
export class AppModule {}
```

Singletons (unless Lazyloaded Module)

HTTP-Service

- HTTPClientModul importieren
- HttpClient-Service injizieren
- Verwendbare Methoden
  - request - Basis aller Methoden

HTTP-Service

- Useable methods
  - [C] post
  - [R] get
  - [U] put
  - [D] delete

HTTP-Service

- Request params
  - method: string,
  - url: string,
  - options?:

HTTP-Service

- Request **options** param object: {
  - body?: any;
  - headers?: HttpHeaders;
    - set (key, value)
  - params?: HttpParams;
    - set(key, value)

HTTP-Service

- **Request options params: {**
  - reportProgress?: boolean;
  - responseType?: 'arraybuffer' | 'blob' | 'json' | ,text';

Routing

Router

- Basis einer Single Page Application
- Routen bestimmen, welche Komponenten angezeigt werden.
- Router-Modul wird von Angular bereitgestellt

Routing

- Routing Vorbereitung
- Routing Konfigurieren
- Router Module
- Navigation über Router Direktiven
- Navigation über Router Service
- Childs
- Events
- CanActive
- Resolve
- Parameter
- Lazy Modules

Erste Schritte

- Modul über RouterModule.forRoot einbinden
  - Routes
    - path
    - component
  - { useHash: false }
- <router-outlet></router-outlet> einbinden

redirect

- initial
  - path: '',
    pathMatch: 'full',
    redirectTo: 'list'
- 404
  - path: '**',
    redirectTo: 'list'

routerLink

- Direktive
  - Wert
    - Pfad | [Pfad, …Params]
- routerLinkActive
  - Wert
    - CSS-Klassenname

Navigation über RouterService

- Router Service Injizieren
- router.navigate Methode
  - Params
    - Liste
      - Pfad
      - Parameter

Parameter

- Route mit Parameter definieren
  - path: 'details/:id',
    component: UserDetailsComponent
- In Komponente ActivatedRoute Service injizieren
  - this.subscription = this.route.paramMap.pipe (
    .map (paramMap => paramMap.get ('id') || 'None')
    .subscribe(id => this.param_id = id));

Resolve

- Daten vor Routenwechsel beschaffen
  - ResolveService auf Basis des Resolve Interface anlegen, einbinden und in Route einbinden
    - path: 'details/:id',
      component: UserDetailsComponent,
      resolve: {
        user: ResolveService
      }

Resolve

- Observable des ResolveService befüllt die Eigenschaft
- Abruf der Daten auf ngInit über ActivatedRoute Service
- this.route
  .data.pipe (
  .map (data => data['user']))
  .subscribe(user => this.user = user);

CanActive

- Genehmigung der Aktivierung einer neuen Route
- Hierfür wird ein auf dem CanActive-Interface basierender Service erstellt und eingebunden
  - canActivate (route : ActivatedRouteSnapshot, state : RouterStateSnapshot) : Observable<boolean>|Promise<boolean>|boolean

CanActive

- Service wird in die Routendefinition implementiert
  - path: 'home',
    component: HomeComponent,
    canActivate: [CanActiveService]

Ereignisse

- Router Service injizieren
- events Observable<Event> subscriben
  - constructor (router: Router) {
    router.events.subscribe(event => console.log (event));
    }

Child

- Eine Route kann Unterrouten haben
- Diese müssen in der Config unter der Eigenschaft
  - children
    - analog zur vorhanden Konfiguration angelegt werden.

Lazy Module

- loadChildren ermöglich im CLI Kontext die einfache Umsetzung
- path : 'dash',
  loadChildren: './dash/dash.module#DashModule'
  - Der Pfad zu dem Modul und der Klassenname muss übergeben werden
    - PFAD#MODUL_NAME

Lazy Module

- Im Modul selbst wird die Route mit der darzustellenden Komponente definiert

- RouterModule.forChild ([

```
  {
    path      : '',
    component: DashComponent
  }
] )
```

FORM

Formulare

- Umsetzbar auf zwei Wege
  - Vorlagen-getrieben
    - Dabei gibt die Vorlage das Formularmodel und die Validatoren vor (ähnlich AngularJS)
  - Reaktiv (Daten-getrieben)
    - Hierbei werden die Formularelemente vorab geplant und an ein Formular in der Vorlage gebunden

Formulare - Vorlagen-getrieben

- Vorbereitend: Einbindung des FormsModuls zur
- Anschließend sind Formular-Direktiven in der Vorlagen-Schicht nutzbar:
  - ngModel, required, minlengt, ...
  - zur Bindung von Validatoren und Werten ins Formular-Model
  - All dies wird ohne zusätzliche Programmierung realisiert

Formulare - Vorlagen-getrieben

- ngForm – wird genutzt, um das Formular auszuzeichnen.
  - Direktive verfügt über ein exportAs d.h. wir können dies für einen #Hash-Id zuordnen #myForm='ngForm'
    - Ermöglicht den Zugriff auf Control-Eigenschaften
      - valid, invalid, value etc.
      - myForm.valid

Formulare - Vorlagen-getrieben

- ngModel kann auf drei Arten genutzt werden
  - Als Attributs-Direktive ngModel kombiniert mit einer Namensdefinition über das name Attribut.
    - Dadurch wird automatisch ein Formular-Model erzeugt
    - myForm.value = {name: Input-Feld-Wert}
  - Als Attributs-Direktive mit Bindung eines Initial-Wertes [ngModel]

Formulare - Vorlagen-getrieben

- Vermeide: Nutzung als Attributs-Direktive mit Zweiwege-Bindung [(ngModel)]. Dadurch wird der Initial-Werte aktualisiert. D.h. es gibt zwei Modelle ☹️
- Als Zuweisung für eine #Hash-Id z.B. #mail='ngModel'
  - Ermöglicht kombiniert mit der ngModel Direktive den Zugriff auf: valid, invalid, value etc.
    - mail.valid

Formulare - Vorlagen-getrieben

- ngModelGroup Direktive zur Gruppierung von Model-Informationen
  - Die Direktive muss hierarchisch in der Vorlage genutzt werden.
    - Die input-Knoten des Direktiven-Elementes erzeugen die Gruppenelemente.

Form

```
<form novalidate #myForm="ngForm">
  <input type="text"
         autocomplete="name"
         placeholder="name"
         name="name"
         #name="ngModel"
         ngModel
         >
  <span ngModelGroup="credentials">
    <input name="email"
           #email="ngModel" ngModel>
    <input name="password"
           #password="ngModel" ngModel>
  </span>
</form>
```

Model

ngForm -> myForm

ngModel -> name

ngModelGroup -> credentials

ngModel -> email

ngModel -> password

Form

```
<form novalidate #myForm="ngForm">
  <input type="text"
         autocomplete="name"
         placeholder="name"
         name="name"
         #name="ngModel"
         ngModel
         >
  <span ngModelGroup="credentials">
    <input name="email"
           #email="ngModel" ngModel>
    <input name="password"
           #password="ngModel" ngModel>
  </span>
</form>
```

Model

```
myForm.value = {
  name: '...',
  credentials {
    email: '...',
    password: '...',
  }
}
```

Formulare – Controls

- ngForm und ngModel – sind Control-Direktiven mit folgenden Eigenschaften:
  - value - Wert
  - valid, invalid - Valide
  - touched, untouched - Berührt
  - dirty, pristine – Benutzt/Unbenutzt
  - errors? – Validator-Fehler

Formulare – Controls

- Control Methoden:
  - setValue, reset – Wert
  - markAsTouched, markAsUntouched - Berührt
  - markAsDirty, markAsPristine – Benutzt/Unbenutzt
  - setErrors? – Validator-Fehler

Formulare – Validatoren

- Validatoren lassen sich über Direktiven einbinden
  - required – erforderlicher Wert
  - email – Gültige Mail
  - minlength, maxlength – Längen-Prüfung
  - pattern – Ausdrucks-Prüfung

Formulare – Validatoren

- Validatoren legen im errors Objekt des Controls Fehlerinformationen in abh. zum Validator ab.
  - Fehlermeldungen lassen sich entsprechend darstellen
  - <div *ngIf="*email*.errors?.required">...
    - Das Fragezeichen bindet optionale Werte

Formulare – Daten senden

- (submit) – Verwenden auf dem Formular das Submit-Ereignis
  - Nutzen als Auslöser im Formular einen <button> oder <a> vom Typ submit
    - Verwende auf dem Auslöser zusätzlich die disable-Direktiven, zum Deaktivieren bei ungültigen Formularen.

```
<form novalidate #myForm="ngForm" (submit)="send(myForm)">
<button type="submit" [disabled]="myForm.invalid">senden</button>
```

Formulare – Daten zurücksetzen

- (reset) – Verwenden auf dem Formular das Reset-Ereignis
  - Nutzen als Auslöser im Formular einen <button> oder <a> vom Typ reset
    - Verwende auf dem Auslöser zusätzlich die disable-Direktiven, zum deaktivieren, wenn noch keine Formularwerte eingetragen sind Formularen.

```
<form novalidate #myForm="ngForm" (submit)="send( myForm )"
      (reset)="reset( myForm, $event )">
<button type="reset" [disabled]="!myForm.dirty">reset</button>
```

Formular CSS-Klassen

- Angular fügt an input-Elemente autom. CSS-Klassen, die den Status des Controls wiederspiegeln.
  - ng-untouched, ng-touched
  - ng-pristine, ng-dirty
  - ng-invalid, ng-valid

Model-Optionen

- Die gleichnamige Direktive beeinflusst das Model-Handling
  - [ngModelOptions]="{name: ‘name‘}“
    - ersetzt das setzen des name-Attributes
  - [ngModelOptions]="{standalone: true}"
    - Wert wird dem übergeordneten Form nicht mitgeteilt

Model-Optionen

- [ngModelOptions]="{updateOn : ‘blur ‘}“
  - Definiert einen Form-Hook (change, submit, blur) bei dem das Model aktualisiert werden soll.
  - debounce - angekündigt: Update nach timeout.