

# Angular Komplett

## Saban Ünlü

# Zwei Worte zu mir

# Saban Ünlü

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

# TypeScript

# Grundlagen

- Programmiersprache basierend von ES6 (ES2015)
  - Entwickelt durch Microsoft
- Exporte in andere ECMA-Script Versionen
- Export in unterschiedliche Modul Handlings
- Typsicherheit
- Nutzung Experimenteller Annotationen

# Variablen

- Definition
  - let
  - const
- Typen
  - Native-Typen
  - Datentypen

# Klassen

- Klassen als Schablone eines JS Objektes
- **constructor**
- Eigenschaften und Methoden
- Instanziieren
- Setter und Getter
- Parameterübergabe

# Vererbung

- Klassen können von anderen Klassen erben
  - **extends**
- Gültigkeitsbereiche
  - **private, public & protected**
- Überschreiben
  - **super**

# Interfaces

- Interfaces sind die Schablonen einer Klasse
  - Interfaces können erben - **extends**
- Implementiert wird ein Interface über
  - **implements**

# Abstrakte Klasse

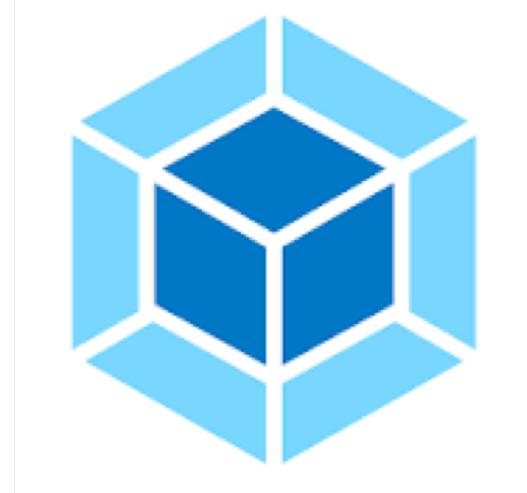
- Implementieren Basis Funktionen und Eigenschaften
- Dient als Vorlage für ein Derivat (Vorlage)
- Kann nicht instanziert werden

# Syntax Magie

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

# Technologien

# Technologien im Überblick



# Node.js

- JavaScript-Laufzeitumgebung
- Verfügbar für unterschiedliche Betriebssysteme
- Benötigt:
  - Testen
  - Veröffentlichen

# webpack

- Bündelt statische Inhalte in Pakete
- Im Angular-Kontext
  - ES-Module, Styles, Vorlagen
    - JavaScript-Pakete
- Vereinfachte Veröffentlichung
- Optimierte Ladeprozesse

# SASS

- Erweiterungssprache für CSS
  - Präprozessor für CSS
- Unterstützt
  - Variablen, Funktionen, Erweiterung, Imports uvm.
- Sehr steile Lernkurve

# Jasmine

- Entwicklungs-Framework zum Testen von JavaScript-Code
  - Unabhängig von weiteren Frameworks
  - Benötigt kein DOM
- Ermöglicht die Definition von verhaltensorientierten Tests
  - Erwartung wird definiert und geprüft
    - `expect(a).toBe(true);`

# Karma

- Framework zum Steuern von JavaScript-Tests
  - Bereitgestellt vom Angular-Team
  - Unterstützt: Jasmine, Mocha und QUnit
- Ermöglicht das Testen auf Geräten
- Sehr gute Integration in Continuous Integration z.B. mit Jenkins



netTrek



NewElements

# Protractor

- Framework für End-to-End-Tests
  - Entwickelt von Google für Angular
  - Tests im echten Browser
  - Simuliert einen Benutzer
  - Benutzerereignisse z. B. Klicks oder Eingaben
  - Wartet auf asynchrone Ereignisse

# Polyfills

- JavaScript-Files
- Überprüft die Existenz bestimmter Funktionen in Browsern
- Falls nicht vorhanden, wird die Funktion erweitert
  - Workaround für ältere Browser

# core-js

- Polyfill für ES6 (ES2015) Funktionen
- Häufig benötigt von weniger modernen Browsern
- Insbesondere der IE benötigt hier Hilfe
- Für die Nutzung von Dekoratoren werden im JIT-Kontext auch ES7/reflect benötigt

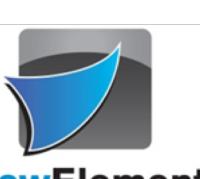
# web-animations

- Angular-Animationen basieren auf der 'Web Animations API'
- Die API wird bislang nur von Chrome, Firefox und Opera gut unterstützt
- Alle anderen benötigen das web-animations-Polyfill



# Zone.js

- Framework ermöglicht die Definition eines Ausführungskontexts für JavaScript
  - Vergleichbar Domains in Node.js
- Wird in Angular als Abhängigkeit genutzt
- Überwacht und steuert die Ausführung
  - Hilft beim Debugging



# ReactiveX

- Framework, um Ereignisse und asynchrone Prozesse zu überwachen
- Wird für unterschiedliche Programmiersprachen angeboten
- RxJS ist die JavaScript-Variante
- In Angular als Abhängigkeit genutzt, unter anderem für **HTTP** und **EventEmitter**



netTrek



NewElements

# Tooling

- VSCode
  - <https://marketplace.visualstudio.com/items?itemName=denvboots.angular-productivity-pack>
- Chrome
  - Augury

# Projektsetup

# Erste Schritte

- Mac
  - XCODE installieren
  - node.js installieren ( $\geq 10.9.x$ )
- Win
  - node.js installieren ( $\geq 10.9.x$ ) [environment Path setzen]
  - Git installieren (inkl. Bash)



# Erste Schritte

- Environment Testen
  - node –version
  - npm –version
  - git --version
- Angular CLI Installieren: npm install -g @angular/cli
- Testen: ng --version

# Erste Schritte

- Proxy?
  - npm config set proxy <http://PROXYURL>
  - npm config set https-proxy <https://PROXYURL>

# Erste Schritte

- ng new [name] --prefix [name]
- ng serve (jit)
  - -- aot
  - -- prod
- ng build
  - -- prod

# Decorator

- Funktionen mit vorangestelltem @-Symbol
- Wird vor einer Deklaration verwendet
- Decorators in Angular haben gleiche Kernfunktionalitäten
  - Speichern von Metainformationen
  - Manipulation nachfolgender Deklaration

```
@HostListener('click')
onHostClick() { /* */ }
```

# Decorator

- Decorator-Typ
  - Klassen dekorieren
  - Eigenschaften dekorieren
  - Methoden dekorieren
  - Parameter dekorieren

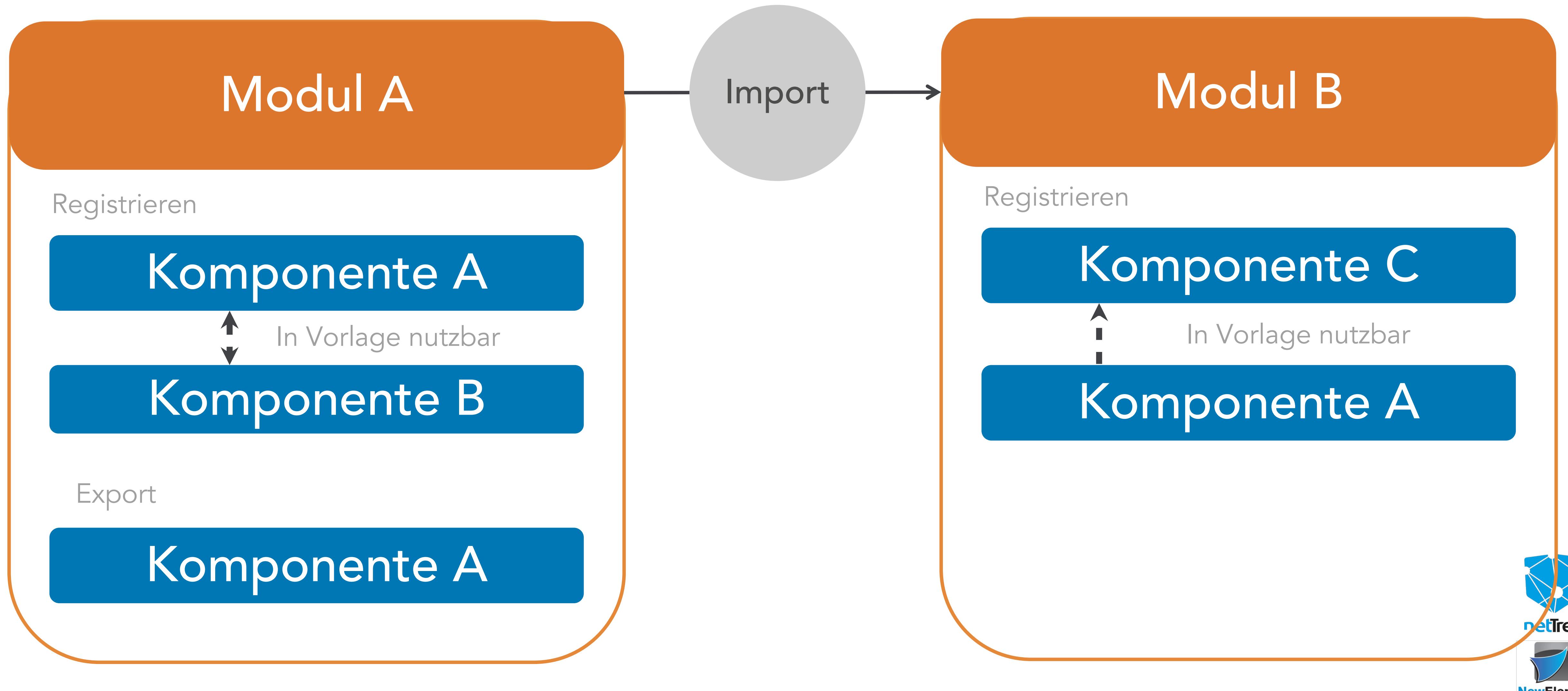
# Architektur

## Module

# Modulare Entwicklung

- Angular-Module
  - Perfekt für Teamwork
  - Wiederverwendbar
    - Export/Import
  - Container (zugänglich)
    - Komponenten, Direktiven, Pipes, Services

# Modulare Entwicklung



# 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)
  - HttpClientModule (XHR)
  - FormsModule (Formulare)
  - RouterModule (Komponenten-Router)

# angular-cli

- ng generate [schematic]
- module [name]
  - -- project
  - -- module
  - -- routing

# Module

- @NgModule
  - imports
    - definiert Module die in diesem Modul benötigt werden
  - declarations
    - benötigte Komponenten, Direktiven, Pipes

# 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



# Module

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



# Module - Bootstrap

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

# Architektur Komponenten

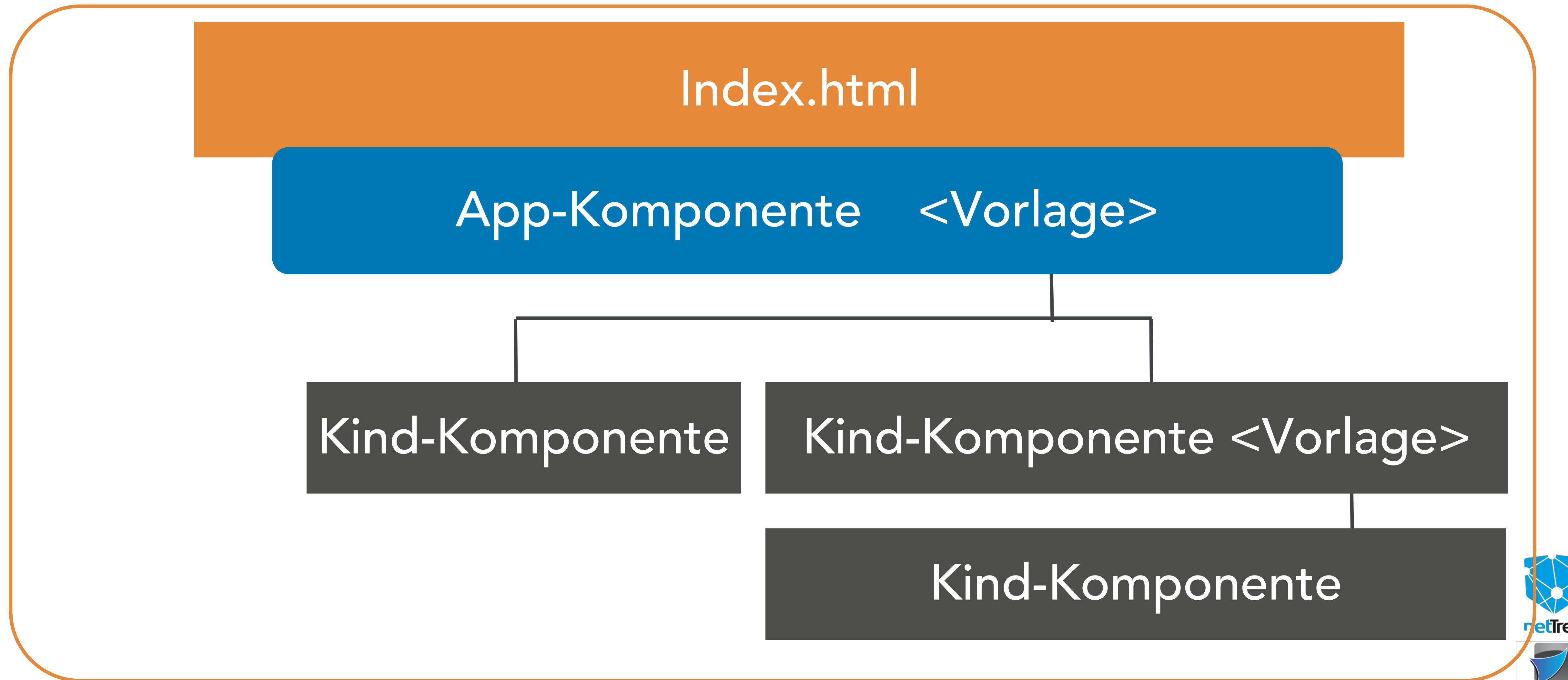
# Komponentenbasierte Entwicklung

- Komponente entspricht eigenen HTML-Knoten
  - Logik
  - Vorlage (HTML)
  - Style (optional)
- Kind-Komponente
  - Verwendung von Komponenten innerhalb einer Vorlage

# Vorlagen

- HTML-Schnipsel
  - Stellt Benutzeroberfläche einer Komponente dar
  - Definierbar als
    - Zeichenkette oder externe Dateien
    - Als Metainformation einer Komponente template oder templateUrl

# Komponentenbasierte Entwicklung



1

## Logik (TS)

```
export class UserComponent {  
  
  name = 'Saban Ünlü';  
  chgName () {  
    this.name = 'Peter Müller';  
  }  
  
}
```

2

## View (HTML)

```
<h1>{{name}}</h1>  
  
<button>(click)="chgName()">  
  Ändern  
</button>
```

3

## View (Style)

```
h1 {  
  color: darkslategray;  
}  
button {  
  background-color: yellowgreen;  
}
```

# Komponente erzeugen

- Komponenten Klasse (ts) anlegen
  - `export class ComponentName`
- Klasse mit Metainformationen versehen
  - `@Component ( { /*meta*/ } )`  
`export class ComponentName`



netTrek



NewElements

# angular-cli

- ng generate [schematic]
- component [name]
  - -- project
  - -- spec
  - -- export

# Komponente erzeugen

- `@Component` – Decorator (Metainformationen)
  - `selector` – HTML-Knotenname
  - `templateUrl` oder `template` – Vorlagen der Komponente
  - `Styles` oder `styleUrls` – Liste der Style-Definitionen



netTrek



NewElements

# 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.None
  - ~~ViewEncapsulation.Native~~ (deprecated)
  - ViewEncapsulation.ShadowDom

# Komponenten-Lebenszyklus

constructor

ngOnChanges

ngOnInit

ngDoCheck

ngAfterContentInit

ngAfterContentChecked

ngAfterViewInit

ngAfterViewChecked

ngOnDestroy

export class UserListComponent

<userList [data] = "userList">

<userList> Vorlage

<user></user><user></user>

<user> </user>

</userList>



# Komponent Content

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

# Inhalte transkludieren

- Komponenten stellen eine View dar.
- Beschrieben wird die View in der HTML-Vorlage
- Innerhalb der Vorlage können Kinds-Komponenten mit Inhalts-Knoten versehen werden
- `<user-list>`  
`<user-item>name</user-item>`  
`</user-list>`



netTrek



NewElements

# Inhalte transkludieren

- Inhalts-Knoten werden transkludiert, wenn Vorlagen die **ng-content** Direktiven (Knoten) nutzen.
  - Der Knoten stellt dabei einen Platzhalter dar
  - Mittels select Attribut lässt sich definieren, für welchen Inhalt der Platzhalter greifen soll



1

## View

```
<user-list>  
  <user-header></...><br/>  
  <user-item></...><br/>  
  <user-item></...><br/>  
  <user-item></...><br/>  
</user-list>
```

2

## UserList - Template

```
<h3>user-list</h3>  
  
<ng-template><user-header></...><br/>  
  <select="user-header"><br/>  
    <user-item></...><br/>  
  </ng-content><br/>  
<user-item></...><br/>  
<ng-content></ng-content>
```

# Zugriff auf transkludierte Inhalte

- Über Eigenschafts-Decorator
  - @ContentChild
    - Parameter: Komponentenklasse
    - Optionales Options-Objekt mit Read-Eigenschaft
      - { read:  
ElementRef | ViewContainerRef | Directive | Service }

# Zugriff auf transkludierte Inhalte

- Über Eigenschafts-Dekorator
- Zugriff erst nach Hook
  - `ngAfterContentInit`

# Zugriff auf transkludierte Inhalte

- Über Eigenschafts-Dekorator
  - @ContentChildren
    - Parameter: Komponentenklasse
    - Erzeugt
      - QueryList<Type>
        - changes -> Observable

# Vorlagen Elemente ermitteln

- Über Eigenschafts-Decorator
  - @ViewChild
    - Parameter: Komponentenklasse / Hash-ID
    - Optionales Options-Objekt mit Read-Eigenschaft
      - { read:  
ElementRef | ViewContainerRef | Directive | Service }

# Vorlagen Elemente ermitteln

- Über Eigenschafts-Dekorator
- Zugriff erst nach Hook
  - `ngAfterViewInit`

# Vorlagen Elemente ermitteln

- Über Eigenschafts-Dekorator
  - @ViewChildren
    - Parameter: Komponentenklasse
    - Erzeugt
      - QueryList<Type>
      - changes -> Observable

# Bindungen

# Bindung

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

## Logik (TS)

```
export class UserComponent {  
  
  name = 'Saban Ünlü';  
  chgName () {  
    this.name = 'Peter Müller';  
  }  
  
}
```

## View (HTML)

```
<h1>{{name}}</h1>  
  
<button (click)="chgName()">  
  Ändern  
</button>
```

# Bindungen

- Werte und Methode in Vorlagen binden
  - Mittels Ausdrucksinterpolation
  - `<h1>{{name}}</h1>`
  - `<h1>{{getName()}}</h1>`
  - ``

# Bindungen

- Werte und Methode in Vorlagen binden
  - Als Eigenschaft binden
    - <img [src]= "imgPath">
  - Als Attribut binden
    - <img [attr.alt]= "imgAlt">



# Ausdrücke interpolieren

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

# Eigenschaften

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



netTrek



NewElements

# Attribute

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



netTrek



NewElements

# Styles

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



netTrek



NewElements

# Class

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

# Ereignis

- Erlaubt Bindung von Ereignissen
  - (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;  
}
```



# Komponentenattribute

- Benutzerdefinierte Attribute lassen sich über den Eigenschaftsdekorator anlegen
  - `@Input(OPT_ATTR_NAME)` name: Type
- Auch für Setter nutzbar
- `ngOnChanges` : Hook informiert über neue Werte
  - `SimpleChanges`

# Komponentenereignisse

- Benutzerdefinierte Ereignisse lassen sich über den Eigenschaftsdekorator anlegen
  - `@Output(OPT_ATTR_NAME)` name: `EventEmitter<T>`
- `EventEmitter` sendet Wert via `emit`
- Elter-Komponenten können sich an das Ereignis hängen
  - `$event` – Übertragener Ereigniswert



# HostBindings- und Listener

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

# Direktive

# Direktiven

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

# Direktiven

- Direktiven lassen sich innerhalb einer Vorlage nutzen
- Sie werden als Attribute ausgezeichnet
- Es gibt zwei Typen von Direktiven
  - Strukturelle Direktiven, die den DOM manipulieren
  - Attribut-Direktiven, die das Aussehen und/oder Verhalten eines Elements manipulieren

# Direktiven

- Strukturelle Direktiven sind durch ein Asterisk (\*) vor dem Attributnamen erkennbar:
  - <img \*ngIf="showImg">
  - <li \*ngFor="let label of labels">

# Direktiven

- Attribut-Direktiven ohne Wert:
  - <input matInput>
- Attribut-Direktiven mit Wertzuweisung:
  - <textarea matAutosizeMinRows="2">
- Attribut-Direktiven mit gebundener Wertzuweisung
  - <input [ngClass]=""inputClass">

# Strukturelle Direktiven - nglf

- [ngIf]= „AUSDRUCK“
  - Hängt den Knoten aus dem DOM wenn der Ausdruck false ist

# Strukturelle Direktiven - ngFor

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

# Attribute Direktiven

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

# angular-cli

- ng generate [schematic]
- directive [name]
  - -- project
  - -- spec
  - -- export

# Direktive erstellen

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

# Pipe

# Pipes

- Pipes dienen der Manipulation von Ausgaben
- Sie werden überwiegend in Vorlagen genutzt
  - Ausdruck | PipeName : Parameter
- Die Nutzung auf Code-Ebene ist aber auch möglich
  - DI oder new und transform Methode der Instanz

# Pipes

- Beispiel
  - <h1>{{name | uppercase}}</h1>
- Pipes lassen sich auch in Kette schalten
  - <h1>{{createdAt | date : 'long' | uppercase}}</h1>

# Pipes

- Hauseigene
  - Uppercase
  - Lowercase
  - Date
- ...

# angular-cli

- ng generate [schematic]
- pipe [name]
  - -- project
  - -- spec
  - -- export

# Pipes erstellen

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

# Pipes erstellen

- Pipes sind **pure** d.h. wir haben eine Singleton und die Ausführung erfolgt bei Datenänderung.
- In den MetaDaten kann eingestellt werden das für pure false verwendet wird.
  - Somit ist die Pipe kein Singleton
  - Kann eigene Zustände somit handeln
  - Und wird durch die Änderungserkennung ausgelöst.



**rxjs**

<https://github.com/ReactiveX/rxjs>

<https://www.learnrxjs.io/>

<http://rxmarbles.com/>

# rxjs - Observable

- Grundlage der reaktiven Programmierung. Datenströme (Ereignisse, Client-Server uvm.) lassen sich Beobachten (Observable)
- Cold
  - Observable wartet auf Subscription
- Hot
  - Observable arbeitet bereits

# rxjs - Observable

- Sendet Werte, Fehler und Status in den Datenstrom
  - next
  - error
  - complete

# rxjs - Subject

- Sowohl Observer als auch Observable
  - Damit registrierbarer Datenstrom
  - Und Sender in einem

# rxjs - Subscription

- Registrierung an Observable
  - next
  - error
  - complete
- unsubscribe (Deregistrierung)
- siehe: <http://rxmarbles.com/>



# rxjs – Erstellung eins Observables

- new
- of
- range
- fromEvent
- ...

# rxjs – Operationen am Datenstrom

- Pipe
  - map
  - filter
  - find
  - scan
  - ...

# Dependency Injection

## Service und Provide Grundlagen

# Services

- Sind View-unabhängige Logiken
  - z.B. Client-Server-Kommunikation
- Sind TypeScript-Klassen
  - Instanzbereitstellung über Dependency Injection
    - provide
    - Typisierter Parameter im Konstruktor

# Dependency Injection

- Services, Werte und Funktionen können injiziert werden
- Benötigt: Bereitstellung innerhalb eines Containers (**Injector**)
  - Bereitstellung durch Anhänge in **providers**-Liste
    - Innerhalb von Metadateninformationen für
      - Module
      - Komponenten



1

## ModulA

- Register (**declarations**)
  - KomponenteA
- Bereitstellen (**providers**)
  - ServiceA

2

## KomponenteA

```
constructor(  
    service: ServiceA  
) {
```

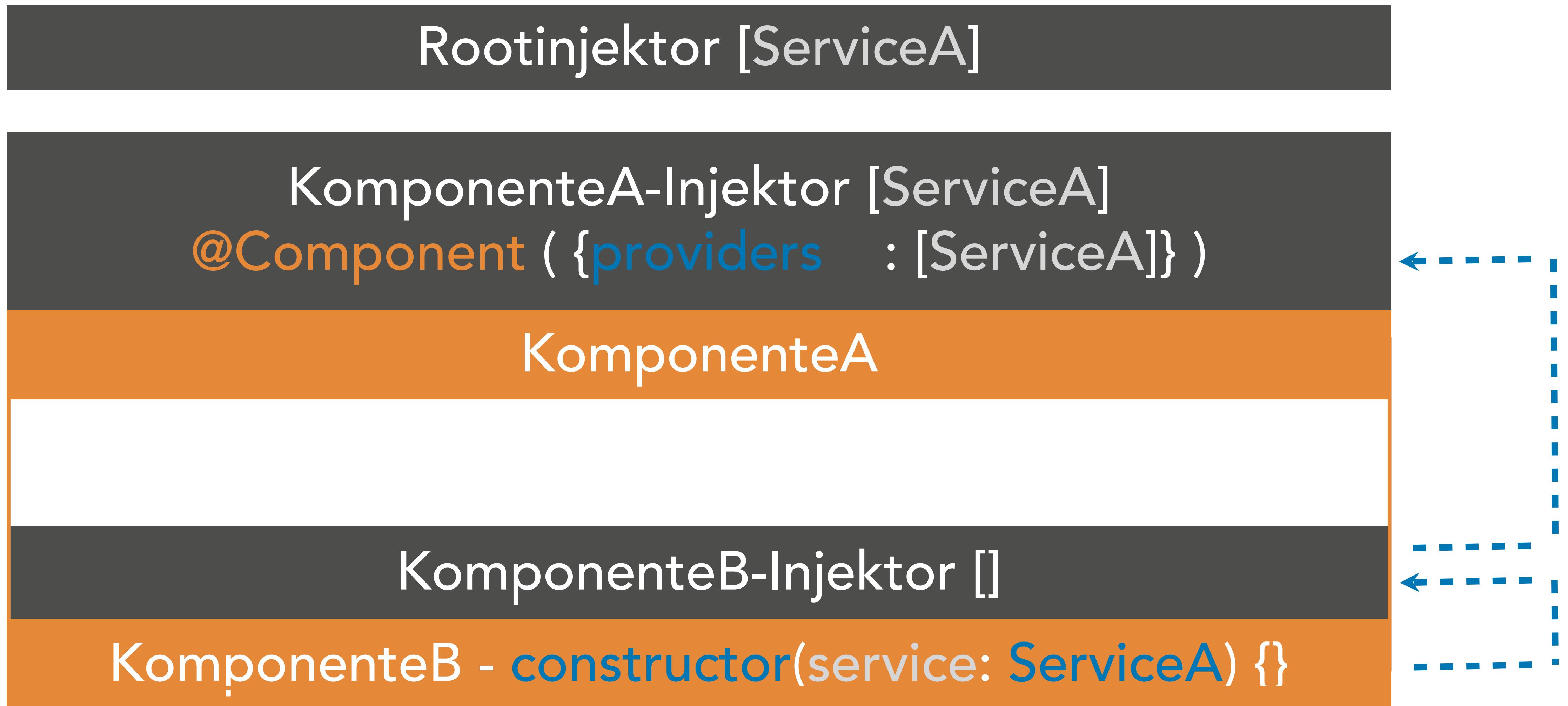
# Dependency Injection

Rootinjektor der Anwendung  
[ ServiceA ]

ModulA  
`@NgModule ( { providers : [ServiceA] } )`

KomponenteA - `constructor(service: ServiceA) {}`

# Dependency Injection

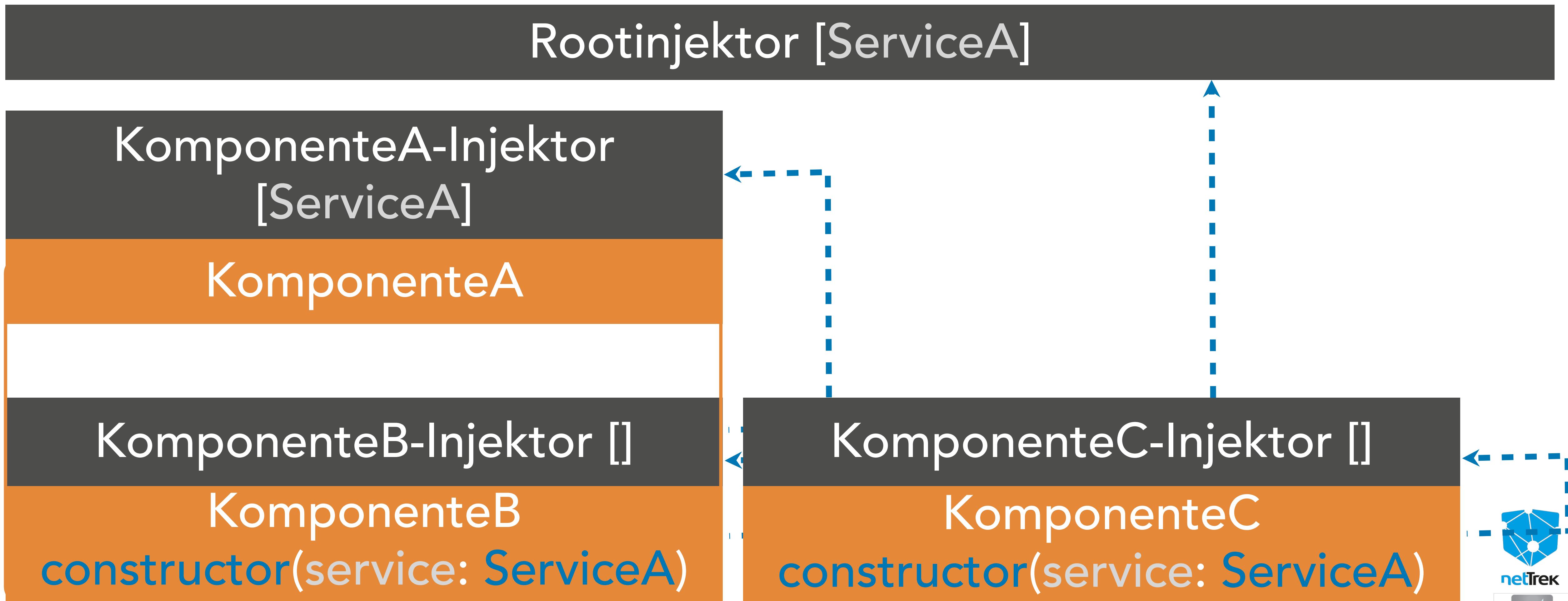


netTrek



NewElements

# Dependency Injection



# Provide von Werten im Injector

- Nutzung von **StaticProvider** Typen statt Klassen
  - ValueProvider
  - ClassProvider
  - ExistingProvider
  - FactoryProvider

# ValueProvider

- Werte im Injector registrieren
  - **provide**: any
    - Referenz zum injizieren
  - **useValue**: any –
    - Wert
  - **multi?**: boolean
    - Nutzung als Liste

# Injizierten-Wert nutzen

- Werte die im Injector bereitgestellt wurden lassen sich Injizieren
  - @Inject Decorator
  - Referenz
  - Token

# ClassProvider

- Klassen im Injector registrieren
  - Wie ValueProvider
  - `useClass: Type<any>` – statt ~~useValue~~
  - Klasse
    - sollte für aot im ES6-Modul exportiert sein

# ExistingProvider

- Existierende Werte nutzen erneut registrieren
  - Wie ValueProvider
  - `useExisting: any` – statt ~~useValue~~
  - Referenz zu einem bereits registrierten Objekt



# FactoryProvider

- FactoryMethode zum registrieren im Injector
- Wie ValueProvider
  - `useFactory`: Function – statt `useValue`
  - Factory-Methode
  - `deps`: [any]
    - Liste von Abh.

# DI-Decoratoren

- **@Injectable** – Zeichnet Service-Klassen aus, damit diese wiederum die DI im Konstruktor nutzen können. Def. Ziel-Injector
- **@Inject** - injiziert anhand eines Tokens
- **@Optional** – wir vor @Inject verwendet, ermöglichen optionale Injizierung
- **@Self, @Host, @SkipSelf** – wird genutzt, um das Injector-Bubbling zu kontrollieren

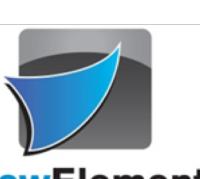


# InjectionToken

- Erzeugt Referenz-Token zu einer DI
  - Generische Type verweist auf Werte-Typ der DI

# ModuleWithProviders

- Erzeuge ein Modul
  - MetaDaten enthalten allg. Imports und Deklarationen etc.
  - Modul-Klasse verfügt über eine statische Factory
    - Rückgabe ist das Modul selbst mit konfigurierten Provides => ModuleWithProviders



# ModuleWithProviders

- `@NgModule ( {} )`

```
export class MyModule {  
    static forRoot ( config ): ModuleWithProviders {  
        return {  
            ngModule : MyModule,  
            providers: [ ... ]  
        };  
    }  
}
```



netTrek



NewElements

# HTTP

## CRUD via HttpClient

# HTTP-Service

- `HTTPClientModul` importieren
- `HttpClient-Service` injizieren
- Methoden
  - `request` – Basis aller anderen Methoden
    - `HttpRequest`

# HttpRequest Methoden

- method: string,
- url: string,
- init?:
  - body?: any;
  - headers?: HttpHeaders;
  - set ( key, value )

# HttpRequest Methoden – init

- params?: `HttpParams`;
  - `set( key, value )`
- reportProgress?: boolean
- responseType?:
  - `'arraybuffer' | 'blob' | 'json' | 'text'`;

# HTTP-Service Methoden

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

# HttpInterceptor

- Beeinflusst Request global
- Service, dass **HttpInterceptor** implementiert
  - `intercept ( req: HttpRequest<any>, next: HttpHandler ): Observable<HttpEvent<any>>`
  - `return next.handle(req);`

# HttpInterceptor - provider

- provide:
  - HTTP\_INTERCEPTORS,
- useClass:
  - Name of Interceptor-Service,
- multi :
  - true

# HttpInterceptor - NoCache

- intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {  
  
    req = req.clone({  
        setHeaders: {  
            'Cache-Control': 'no-cache',  
            Pragma: 'no-cache',  
            Expires: 'no-cache',  
            'Content-Type': 'application/json',  
            Accept: 'application/json'  
        }  
    });  
  
    return next.handle(req);  
}



netTrek



NewElements

# HttpInterceptor – Progress & Error

- intercept ( req: HttpRequest<any>, next: HttpHandler ): Observable<HttpEvent<any>> {  
    console.log ( 'running Requests (start new)', ++numOfRunningRequests );  
    return next.handle ( req )  
        .pipe(  
            tap( ( event: HttpEvent<any> ) => {  
                if ( event instanceof HttpResponse ) {  
                    console.log ( 'running Requests (end success)', --  
                numOfRunningRequests );  
                }  
            }, ( error: any ) => {  
                if ( error instanceof HttpErrorResponse ) {  
                    console.log ( 'running Requests (end err)', --numOfRunningRequests  
                );  
                }  
            }  
        );  
    }  
}



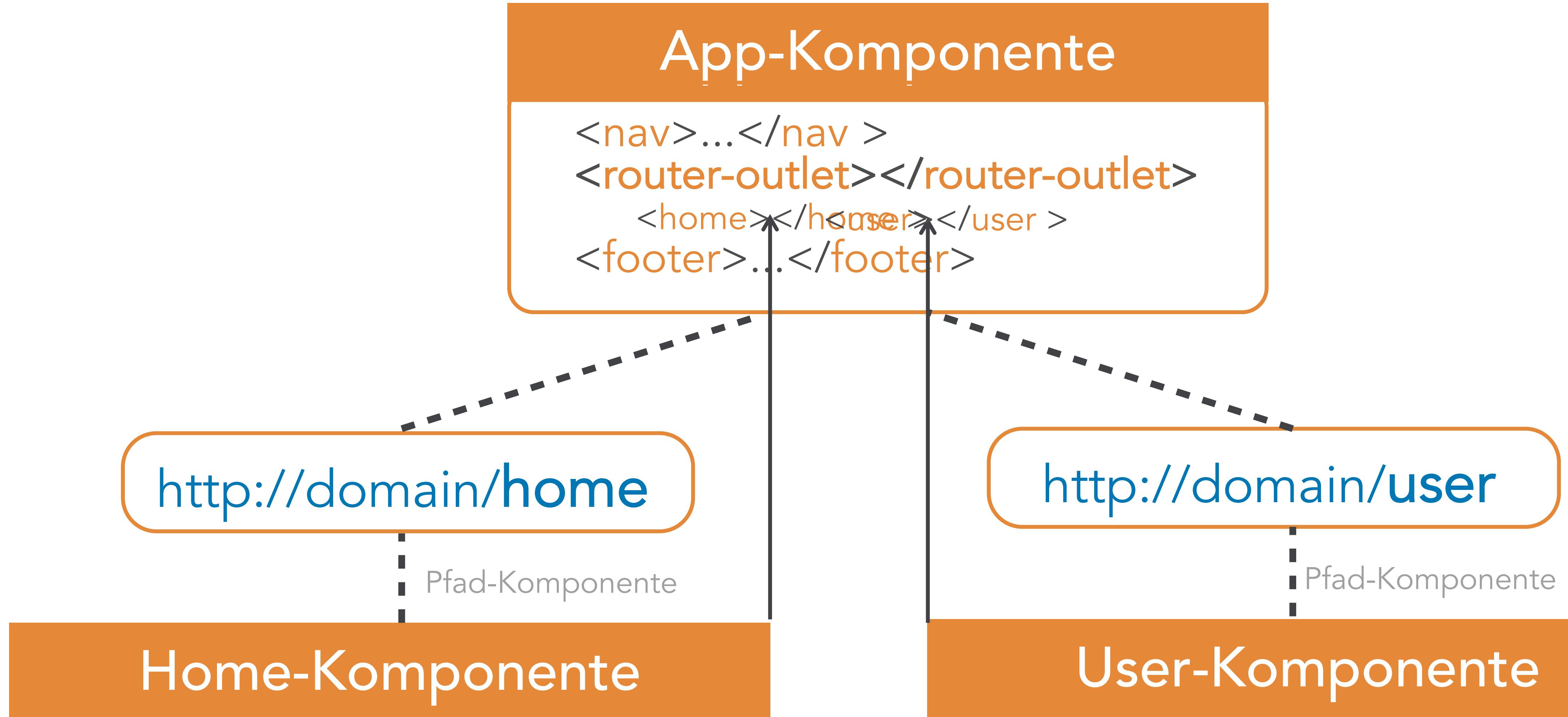
# Routing

## Basis einer SPA

# Routing

- Bestandteil des Routing-Moduls
- Basis einer Single-Page-Application
- Bestimmt, welche Komponenten bei welchem Pfad angezeigt wird

# Routing



# Modul import und Route-Def

- Modul über RouterModule.forRoot einbinden inkl. Config
  - `Routes { path, component }`
  - `{ useHash: false }`
- Optional sind Routen auch über den **Router**-Service und der **config** Liste zur Laufzeit konfigurierbar.
- `<router-outlet></router-outlet>` einbinden

# Redirect

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

# Navigation

- **routerLink** - Directive
  - path | [ path, ...params: any[] ]
- **routerLinkActive** - Directive
  - CSS class name

# Navigation – über RouterService

- DI Router Service
- `navigate` Methode
  - Params
  - List
    - path
    - params

# Lazy Module

- `loadChildren` ermöglicht 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`  
    }  
])

# Lazy Module

- Module Vorladen
- `RouterModule.forRoot( [], opt )`
  - `opt`
    - `enableTracing: true,`
    - `preloadStrategy: PreloadAllModules`

# 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-Guard

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



netTrek



NewElements

# CanActivate - Guard

- 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`

# CanActivate - Guard

- 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 Unteroutes haben
- Diese müssen in der Config unter der Eigenschaft
  - `children`
  - analog zur vorhanden Konfiguration angelegt werden.



# Formulare

# 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, minlength, ...**
  - 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`



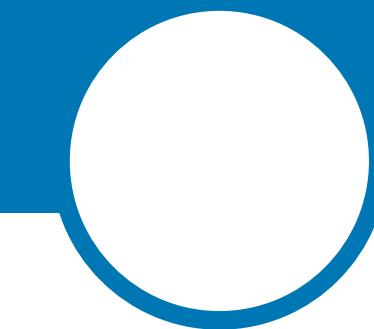
netTrek



NewElements

# 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



netTrek



NewElements

# 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">...</div>`
- 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>
```



netTrek



NewElements

# 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.

# Eigene Validatoren

- Validatoren bestehen aus zwei Schichten
  - ValidatorFn
    - Funktion übernimmt die Überprüfung und reicht aus, wenn die Nutzung ausschließlich reaktiv ist.
  - Direktive
    - Nutzt die zuvor beschrieben Funktion und ist für vorlagengetriebene Formulare notwendig

# Validator-Funktion

- Funktion wird über eine Factory erzeugt, welche optional die Prüfungsbedingung entgegennimmt.
- ValidatorFn - erwartet: AbstractControl
  - gibt ein Fehlerobjekt (ValidationErrors) oder null zurück



netTrek



NewElements

# Validator-Funktion

```
export class EqualValidator {  
    static isEqual ( compare: any ): ValidatorFn {  
        return ( control: AbstractControl ): ValidationErrors | null => {  
            if ( control.value === null || compare === null ) return null;  
            return compare !== control.value ?  
                { 'equal': { 'is': control.value, 'should': compare } } : null;};  
    }  
}
```



# Validator-Direktive

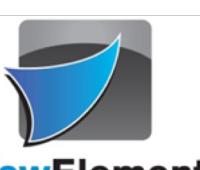
- Validierung wird in `NG_VALIDATORS` durch eine neue Direktive erweitert
- Erweiterung wird im Injector der Direktive bereitgestellt.
- Direktive muss das `Validator` Interface implementieren.
  - `validate ( c: AbstractControl ): ValidationErrors | null`
    - Wird zur Prüfung ausgeführt und gibt Fehlerobjekt zurück

# Validator-Direktive

- `registerOnValidatorChange (fn: any): void;`
- Methoden übermitteln eine Referenz zu den, `onChange` Callback
  - Notwendig, wenn Prüfungsbedingungen sich ändern

# Validator-Direktive – Injector erweitern

- Erweitere den `NG_VALIDATORS`
- Benutze hierfür den `ExistingProvider`
- Als Provide-Token nutze: `NG_VALIDATORS`
- Der Wert wird über `useExisting` auf die Direktiven-Klasse gesetzt, da diese nicht unmittelbar im Injector existiert wird die `forwardRef` Methode genutzt
- `multi: true` erweitert die `NG_VALIDATORS` Liste



# Validator-Direktive – Injector

- ```
export const EQUAL_VALIDATOR = {  
  provide: NG_VALIDATORS, multi: true,  
  useExisting: forwardRef(() => MyDirective) };
```
- ```
@Directive ( { selector: ,[equalValidator][ngModel]',  
  providers: [EQUAL_VALIDATOR] } )
```
- ```
export class MyDirective implements Validator {
```



# Reaktive Formulare

- Im Gegensatz zu Vorlagen-getriebenen Formularen vermeiden wir Direktiven wie: `ngModel`, `required`, `minlength`
- Statt dessen werden zuvor Controls erzeugt und anschließend in der Vorlage gebunden via:
  - `formGroup`, `FormControl`, `FormControlName` ...
  - Als Vorbereitung muss das `ReactiveFormsModule` eingebunden werden.



netTrek



NewElements

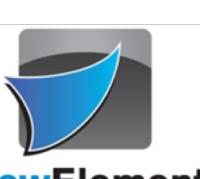
# Reaktive Formulare – Model erzeugen

- Erzeuge Controls für Werten über **FormControl**
  - Konstruktor erwartet **Wert** und **Validatoren**
- Erzeuge Gruppen von Werten über **FormGroup**
  - Konstruktor erwartet ein **Key-Value-Pair Objekt**
    - **Key:** Name des Controls oder der Untergruppe
    - **Value:** Instanz des Controls oder der Untergruppe



# Reaktive Formulare – Direktiven

- **[formGroup]** – Bindet die unterste Wert-Gruppe
- **formGroupName** – Bindet Untergruppe anhand des Names, das im Key-Value-Pair Objekt definiert wurde.
- **formControlName** – Bindet Control anhand des Names, das im Key-Value-Pair Objekt definiert wurde.
- **[formControl]** – Bindet eine Control-Instanz.



# Form

```
<form novalidate [FormGroup]="myForm">  
  <input type="text"  
    formControlName="name"  
  >  
  <span formGroupName="credentials">  
    <input type="email"  
      formControlName="email">  
    <input type="password"  
      formControlName="password">  
  </span>  
</form>
```

# Model

```
this.myForm = new FormGroup ( {  
  name: new FormControl ( 'Saban',  
    Validators.required ),  
  credentials: new FormGroup ( {  
    email : new FormControl (  
      'us@netTrek.de',  
      [ Validators.email,  
        Validators.required ] ),  
    password: new FormControl ( ... ) } );
```

# Form

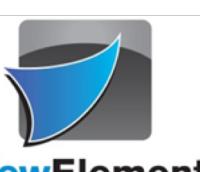
```
<form novalidate [FormGroup]="myForm">  
  <input type="text"  
    formControlName="name"  
  >  
  <span formGroupName="credentials">  
    <input type="email"  
      formControlName="email"  
    <input type="password"  
      formControlName="password">  
  </span>  
</form>
```

# Model

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

# Formulare – Helfer – FormBuilder

- FormBuilder (DI) –Service vereinfacht die Model Erstellung und den Umgang mit FormControl und FormGroup
  - Anstelle von `new FormGroup()` nutzen wir die `group` Methode vom FormBuilder und übergeben ein Key-Value Objekt.
  - `Key`: Name des Controls oder der Untergruppe
  - `Value`: Eigenschafts-Array oder Untergruppe via `group` Methode



# Formulare – Helfer – FormBuilder

- **Value:** Eigenschafts-Array
  - Erstes Element – Startwert
  - Zweites Element: Validator oder Validator-Array

# Formulare – Helper – FormBuilder

```
this.myForm = this.fb.group({  
    name: [ 'Saban', Validators.required ],  
    credentials: this.fb.group ( {  
        email: ['us@netTrek.de', [ Validators.email,  
            Validators.required ]],  
        password: ['test1234', Validators.required ]  
    })  
});
```



# Formulare – Helfer – Control

- **get:** Methode gibt ein Control aus dem Model zurück
  - Parameter:
    - Name des Controls
    - oder Pfad (Names-Array) zu einem Control
  - `this.myForm.get( ['credentials', 'email'] ) as FormControl;`

# Formulare – Helfer – Control - Fehler

- **hasError** : Methode gibt ein Boolean zurück, ob ein bestimmter Validator-Fehler existiert
- Parameter:
  - Name des Errors z.B. **required**, **email** ...
  - Name des Controls oder Pfad (Names-Array) zu einem Control



netTrek



NewElements

# Formulare – Helfer – Control - Werte

- `setValue( value: any, opts?): void;`
  - `onlySelf?` : boolean [default: false]
    - Validation nur auf Control nicht auf Eltern-Komponente
  - `emitEvent?` : boolean [default: true]
    - `valueChanges` Event wird vom Control gefeuert



netTrek



NewElements

# Formulare – Helfer – Control - Werte

- `setValue( value: any, opts?): void;`
- `emitModelToViewChange?: boolean`
  - View wird via `onChange` über die Änderung informiert
- `emitViewToModelChange?: boolean`
  - Model wird via `ngModelChange` über die Änderung informiert



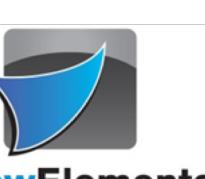
netTrek



NewElements

# Formulare – Helfer – Control - Werte

- `reset( value, opts?: { onlySelf?: boolean;  
emitEvent?: boolean; })`
- Setzt Control zurück
  - `value` = null oder Wert
  - Zustand wird auf `pristine` & `untouched` gesetzt



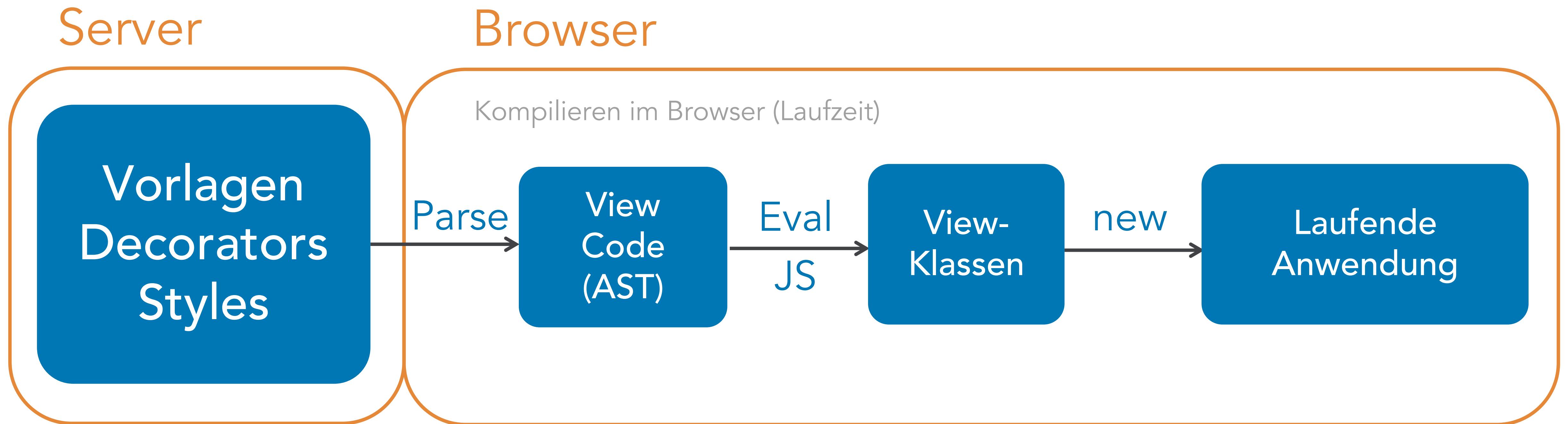
# Formulare – Helper – Control - Status

- markAsTouched( opts?: { onlySelf?: boolean; }): void;
- markAsUntouched( opts?: { onlySelf?: boolean; }): void;
- markAsDirty( opts?: { onlySelf?: boolean; }): void;
- markAsPristine( opts?: { onlySelf?: boolean; }): void;
- disable(opts?: { onlySelf?: boolean; emitEvent?: boolean; })
- enable (opts?: { onlySelf?: boolean; emitEvent?: boolean; })

# Veröffentlichen

## JIT, AOT und mehr

# JIT



netTrek

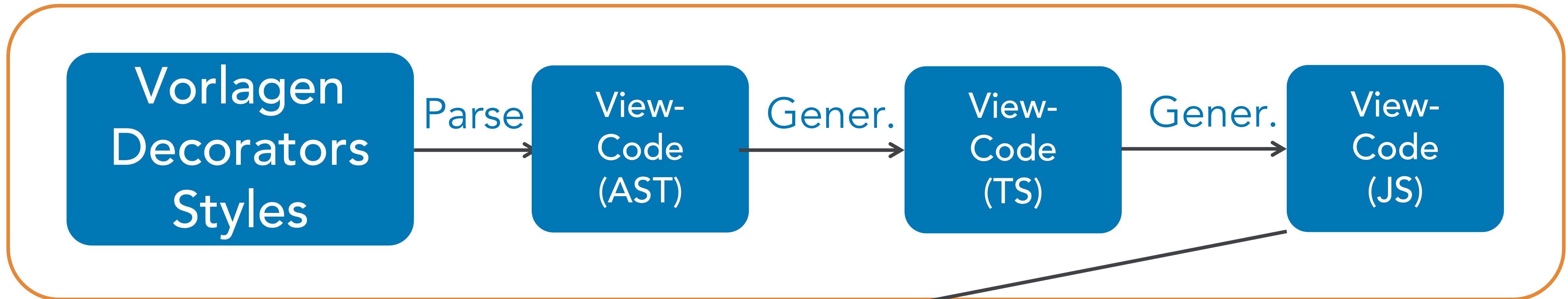


NewElements

# AOT

Entwickler

Angular Compiler - vorkompilieren



Server

Browser



# Veröffentlichen (cli < 6)

- ng build
- --target (-t) production | development
  - --dev (-t=development e=dev)
  - --prod (-t=production e=prod)
- --environment (-e) prod | dev <custom>
- --aot

# Veröffentlichen

Option	--dev	--prod
--aot	false	true
--environment	dev	prod
--output-hashing	media	all
--sourcemaps	true	false
--extract-css	false	true
--named-chunks	true	false
--build-optimizer	false	true with AOT and Angular 5

# Veröffentlichen (cli >= 6)

- ng build [app-name]
  - --prod
    - configuration (production)
    - --aot = true
    - --build-optimizer (true)
      - optimization (UglifyJS)

# Veröffentlichen (Environment)

- angular.json
  - configurations Bereich der App erweitern (neuerName)
  - fileReplacement definieren
- ng build [app-name]
  - configuration
    - neuerName

# Veröffentlichen – Packet Analyse

- ng build [app-name]
  - --stats-json
    - # webpack-bundle-analyzer

# Unit-Testing

# Grundlagen

- Setup
  - angular.json
  - projects > [name] > architect > test
    - main > **test.ts**
  - tsConfig > src/tsconfig.spec.json
  - karmaConfig > src/karma.conf.js

# Grundlagen

- test.ts
  - Einstellung für TestBed (engl. Testumgebung)
    - JIT Setup
    - Spec Definitionen

# Grundlagen

- tsConfig
  - TypeScript Compiler Einstellungen
    - ~~module > commonjs~~
- karmaConfig
  - Test-Runner für Unit Tests z.B. mit Jasmine
  - Setup des Karma-Umgebung

# Grundlagen

- Setup des Karma-Umgebung
  - Pfade und Frameworks
    - Jasmine & @angular-devkit/build-angular
  - Setup der Plugins
    - Framework, Reporter und Launcher
  - Setup generelle Setups

# Spec Dateien

- describe Block für einen Test erstellen
  - description: string
  - callback handler
- describe callback hat zwei Haupt-Phasen
  - beforeEach (jasmine) zu vorbereiten der Test-Assets
  - it (jasmine) zum testen

# Spec Dateien

- `beforeEach` (jasmine) erwartet ein callback handler, der Test-Assets vorbereitet
- `it` (jasmine function) erwartet zwei Parameter
  - `description: string` -> dargestellt während der Tests
  - `describe:description + it:description`
    - Componente + shold ... (Ausdruck steht vor Testauswertung)
    - callback Handler führt die Tests aus



# Spec Dateien

- callback - body
  - Innerhalb des Handlers werden die Tests ausgeführt über
  - expect (jasmine) Methode
  - parameter – zu testender Wert
  - return Instanz zur Prüfung der Übereinstimmung



netTrek



NewElements

# Spec Dateien

- Test der Übereinstimmung
  - `toBe (val)` -> vergleichbar `==`
  - `not.toBe(val)` -> vergleichbar `!=`
- `toEqual(val)` -> vergleicht Objekt und alle Felder
- `toMatch(regExp)` -> vergleichbar `regExp`
- `toBeDefined (val)` -> vergleichbar `!= undefined`

# Spec Dateien

- Test der Übereinstimmung
  - **toBeUndefined (val)** -> vergleichbar === undefined
  - **toBeNull (val)** -> vergleichbar === null
  - **toBeTruthy(val)** -> vergleichbar === Boolean(val)
  - **toBeFalsy (val)** -> vergleichbar !== Boolean(val)
  - **toContain (val)** -> vergleichbar indexOf !== -1



# Spec Dateien

- Test der Übereinstimmung
  - `toBeLessThan (val)` -> vergleichbar < val
  - `toBeGreaterThan (val)` -> vergleichbar > val

# TestBed

- TestBed ([Angular test utils](#)) im beforeEach konfigurieren
  - **configureTestingModule** Factory für Testing Modules
  - Einsatz vor jedem Test im Ruhezustand
  - als Parameter wird ein NgModule MetaData-Objekt übergeben
  - **compileComponents()** - kompiliert alle Komponenten im Module zu Inline JavaScript

# TestBed

- `compileComponents()`
  - und alle Übereinstimmungsmethoden
    - geben ein `Promise` zurück
  - Handelt für `beforeEach`, `afterEach` und `it` werden daher oft
    - in der `async` Methode gekapselt.
      - `beforeEach(async(() => { ... }) )`

# TestBed

```
beforeEach(async(() => {
  TestBed.configureTestingModule({
    imports: [
      RouterTestingModule
    ],
    declarations: [
      AppComponent
    ],
  }).compileComponents();
}));
```

# TestBed

- `createComponent( Komponenten Klasse)` Methode schließt die TestBed-Konfiguration und gibt eine `ComponentFixture` Instanz zurück.
  - `fixture = TestBed.createComponent ( AppComponent );`
  - bietet Zugriff auf ein Debug-Element und die Instanz der Komponente.
    - `componet = fixture.componentInstance;`
    - `fixture.debugElement.componentInstance`



# TestBed

- **detectChanges ()** Methodw der ComponentFixture Instanz führt die Änderungserkennung aus & rendert das Template
  - `fixture.detectChanges();`
- Das Fixture Debug-Element bietet das nativeElement der Komponente
  - `fixture.debugElement.nativeElement`



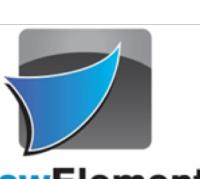
netTrek



NewElements

# TestBed

- Das **nativeElement** unterstützt **querySelector**.
- Alternativ kann **query**, vom **debugElement** mit der **By** kombiniert werden.
  - `fixture.debugElement.query ( By.css ( 'h1' ) );`



# TestBed

```
beforeEach ( () => {
  fixture = TestBed.createComponent ( AppComponent );
  component = fixture.componentInstance;
  fixture.detectChanges ();
  debugElement = fixture.debugElement.query ( By.css (
  'h1' ) );
  htmlElem      = debugElement.nativeElement;
} );

it ( 'should render title in a h1 tag', () => {
  component.changeTitle( 'test' );
  fixture.detectChanges();
  expect ( htmlElem.textContent )
    .toContain ( `Welcome to test!` );
} );
```



netTrek



NewElements

# Services testen

- Service im `TestBed` Modul providen
  - via `TestBed.get` oder `inject` Methode anfordern
  - `beforeEach ( () => { TestBed.configureTestingModule ( { imports : [ HttpClientTestingModule ], providers: [ UserService ] } ); service = TestBed.get ( UserService ); httpMock = TestBed.get ( HttpTestingController ); } );`



netTrek



NewElements

# HttpClientTestingModule

- Verwende das Modul als Abh. in TestBed
- Nutze den HttpTestingController zum mocken
- Erzeuge einen **TestRequest** mit dem Ctrl.
  - `testRequest = httpMock.expectOne( url )`
  - sende Response mittels **flush** Methode.
    - `testRequest.flush( body, opts? );`

# HttpClientTestingModule

- it ('should getUsers', () => {  
 const dummyUsers: User[] = [  
 { name: 'saban', age: 33 }, { name: 'peter', age: 22 }, ];  
  
 service.getUsers(); const testRequest =  
 httpMock.expectOne(` \${environment.endpoint}/users`);  
  
 testRequest.flush( dummyUsers );  
 expect( service.users ).toBe( dummyUsers );  
 });



# Helper - HttpClientTestingModule

- Im afterEach nicht genutzte Requests entfernen
- `afterEach( () => {  
 httpMock.verify();  
});`

# e2e-Testing

# Protractor

- Blackbox e2e Tests
- Entwickelt von Google aus Basis
  - Selenium
  - Webdriver
- kein Ersatz für Unit-Tests
  - sehr gute Ergänzung

# Protractor - Konfiguration

- Anular.json
  - Eigenes Projekt für e2e
    - protractorConfig
    - devServerTarget
    - tsConfig

# Protractor - ProtractorConfig

- Einstellungsmöglichkeiten für
  - Test-Framework **jasmine**
  - Browser **jasmine**
    - URL
  - Tests **./src/\*\*/\*.e2e-spec.ts**
  - tsConfig **tsconfig.e2e.json**
  - uvm.

# Protractor - Tests

- Vergleichbar Unit-Tests
  - öffnen einer Seite `browser.get('/');`
  - Ausführen von Tests mit `jasmine`
    - `expect(page.getParagraphText()).toEqual('Welcome!');`

# Protractor - Helfer

- aus dem Protractor Modul
  - browser: ProtractorBrowser
    - get('/');
    - getTitle()

# Protractor - Helfer

- element (by-statement): ElementFinder
  - all(by.repeater('result in memory'));
- by: ProtractorBy
  - css( selector );
- element(by.css('pr-root h1')) => WebElement

# Protractor - Helfer

- element: WebElement
  - click( );
  - sendKeys( keys/string [] )
  - getCssValue( cssProp )
  - getText()
  - isEnabled()

# Protractor - Helfer

- element: WebElement
  - isSelected()
  - isDisplayed()
  - submit()