

# Angular (8.x) Kompaktkurs

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



NewElements



netTrek

# Variablen

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



# Klassen

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



NewElements



netTrek

# 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**



NewElements



netTrek

# Abstrakte Klasse

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



# Syntax Magie

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



NewElements



netTrek

# Projektsetup

# Erste Schritte

- Mac
  - XCODE installieren
  - node.js installieren ( $\geq 10.9.x$ )
- Win
  - node.js installieren ( $\geq 10.9.x$ )
  - Git installieren (inkl. Bash)

# npm Proxy ?

- npm config set proxy http://PROXYURL
- npm config set https-proxy https://PROXYURL



NewElements



netTrek

# Setup Manuell

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

# angular-cli

- Kommandozeilen Tool
  - Initialisieren & einrichten
  - Entwickeln und Warten
  - Testen und veröffentlichen



# angular-cli - installieren

- `npm install -g @angular/cli`
- optional
  - Entwickeln und Warten
    - CLI im App Kontext



Angular Console



# angular-cli

- `ng new netTrek --prefix=nt`
- `ng serve`
- `ng g server --aot`
- `ng build`
- `ng build --prod`
- `ng lint`
- `ng test`
- `ng e2e`



# Trainings Branch

- `git clone -b training/ba`  
`https://github.com/netTrek/ng8Adv.git baTraining`



# Architektur

# Einleitung

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



NewElements



netTrek

# Architektur

## Decorator

# 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



NewElements



netTrek



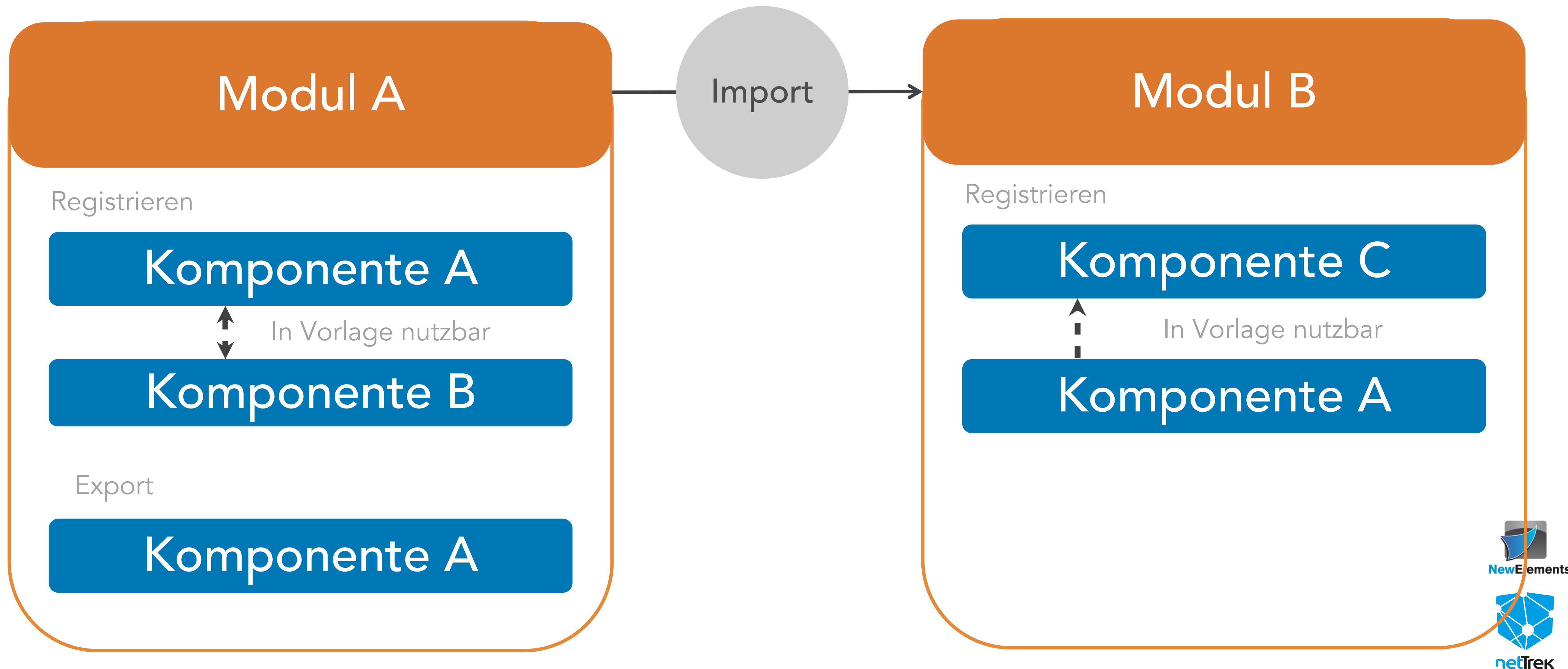
# Architektur

## 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



NewElements



netTrek

# Module

- Angular-eigene Module
  - BrowserModule (Ereignisse, DOM)
  - CommonModule (Direktiven, Pipes)
  - HttpClientModule (XHR)
  - FormsModule (Formulare)
  - RouterModule (Komponenten-Router)



NewElements



netTrek

# Module

- Module erzeugen
  - Modul-Klasse anlegen



NewElements



netTrek

# Module

```
class AppModule {}
```



NewElements



netTrek

# Module

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



# Module

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

# Module

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



NewElements



netTrek

# 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



NewElements



netTrek

# 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.



NewElements



netTrek

# Module - Bootstrap

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

# Architektur

## Komponenten

# Einleitung

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





# 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`

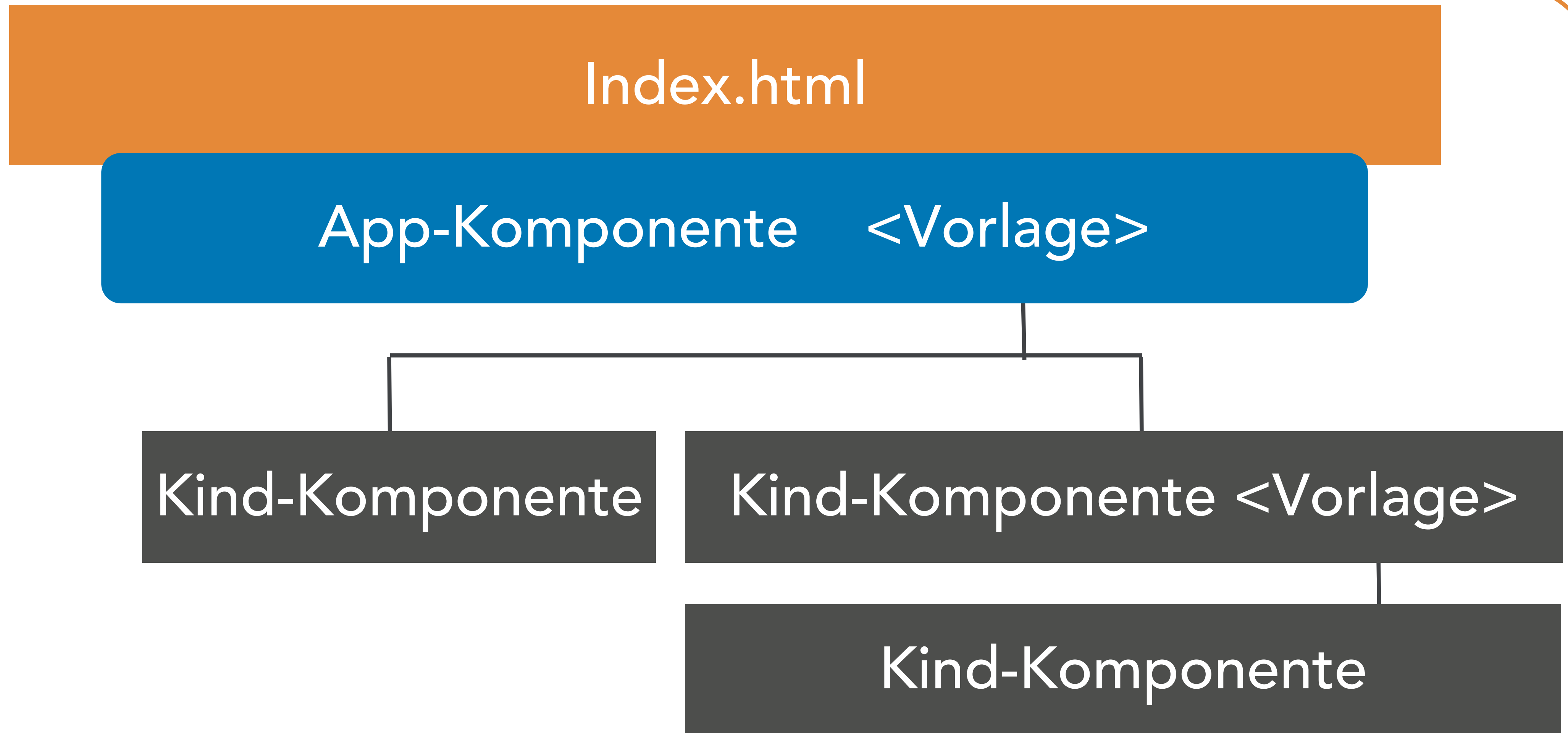


NewElements



netTrek

# 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`



NewElements



netTrek

# Komponente erzeugen

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



# Komponente erzeugen

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

# Komponente erzeugen

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

```
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']
```

```
export class  
  name = 'app works!';  
  onClick () {  
    console.log ( 'clicked' );
```



# Komponente erzeugen

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



NewElements



netTrek

# Komponente erzeugen

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



NewElements



netTrek

# Komponente erzeugen

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

```
<my-component> </my-component>
```



# Komponent Metadaten

- `ng g c user/user --export --skipTests --flat`
  - selector
    - Knoten
  - Vorlage
    - templateUrl (file)
    - template (backticks)



NewElements



netTrek

# Komponent Metadaten

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

# Komponent Metadaten

- Style
  - encapsulation - Umgang mit Webkomponenten
    - ViewEncapsulation.Emulated
    - ViewEncapsulation.None
    - ~~• ViewEncapsulation.Native (deprecated)~~
      - ViewEncapsulation.ShadowDom



NewElements



netTrek

# Bindungen

# Bindung

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



NewElements



netTrek

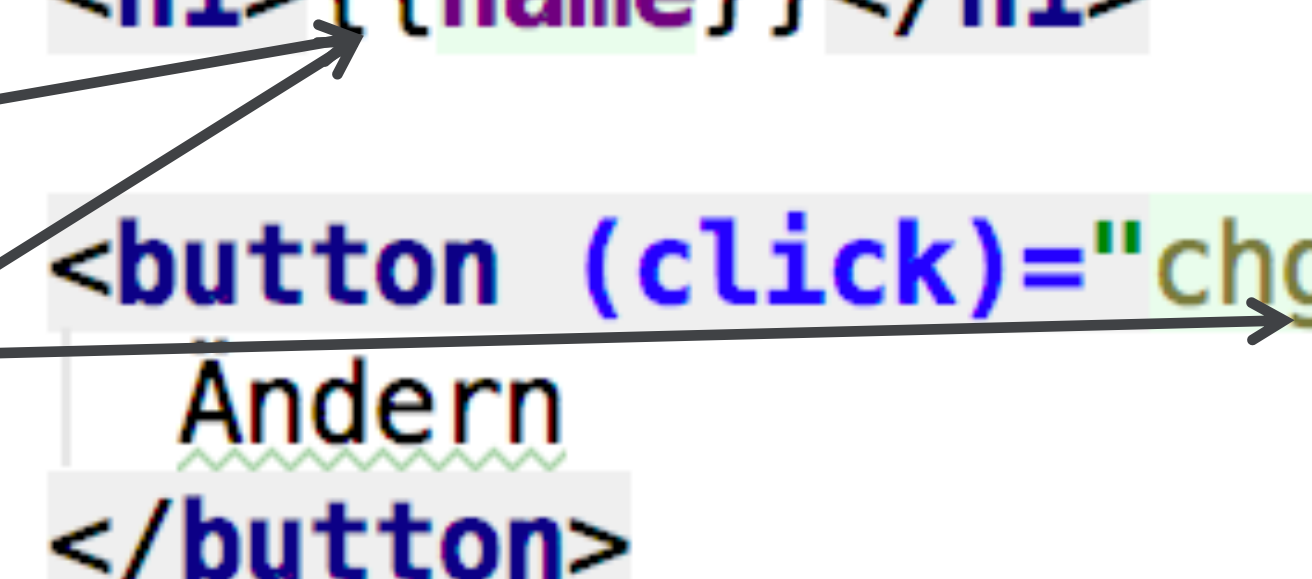


## Logik (TS)

```
export class UserComponent {  
  name = 'Saban Ünü';  
  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



NewElements



netTrek

# 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 von Ereignissen
  - (EVENT)=„METHODODE( \$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;  
}
```



New Elements



netTrek

# 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

# Komponenten-Lebenszyklus

constructor

ngOnChanges

ngOnInit

ngDoCheck

ngAfterContentInit

ngAfterContentChecked

ngAfterViewInit

ngAfterViewChecked

ngOnDestroy

```
export class UserListComponent
```

```
<userList [data]="userList">
```

```
<userList>Vorlage
```

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

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

```
</userList>
```

# HostBindings- und Listener

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

# Direktive

# Direktiven

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



NewElements



netTrek



# 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



NewElements



netTrek

# 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 - ngIf

- [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, even, odd, count

# Attribute Direktiven

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



NewElements



netTrek

# 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



NewElements



netTrek

# 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
  - ...

# 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

- Iterierbares Objekt, welches filter- und registrierbar (Subscription) ist, um async. Prozesse zu verfolgen
- 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 eines 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
  - Instanzibereitstellung 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



NewElements



netTrek

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

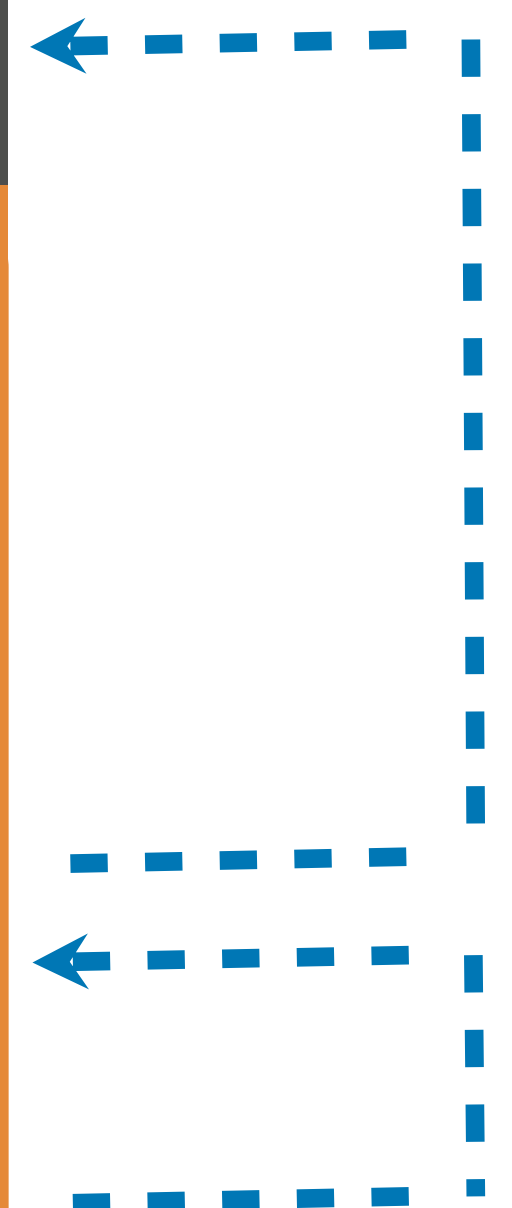
Rootinjektor [ServiceA]

KomponenteA-Injektor [ServiceA]  
`@Component ( {providers : [ServiceA]} )`

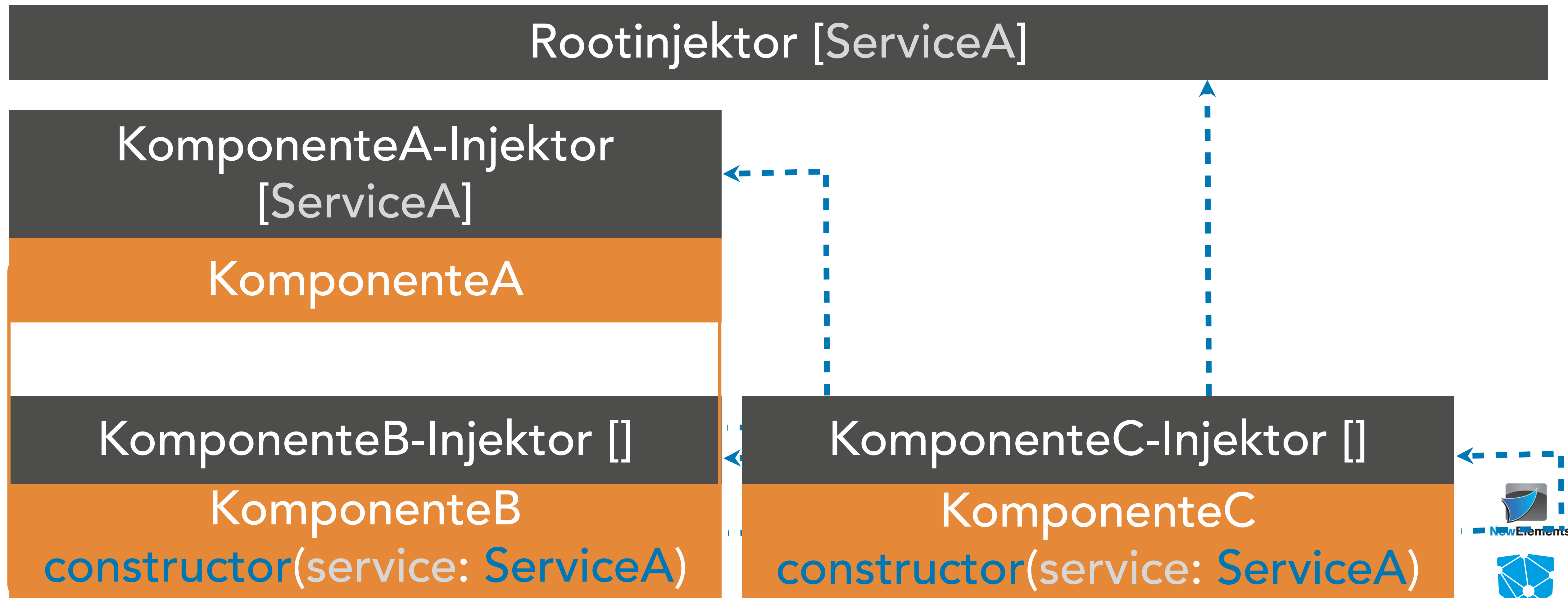
KomponenteA

KomponenteB-Injektor []

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



# Dependency Injection



# HTTP

## CRUD via HttpClient

# Benutzen

- HttpClient-Modul importieren
- HttpClient-Service injizieren
- Methoden
  - Einen der CRUDServices nutzen
    - request<R>-Methode = Basis aller anderen Methoden
      - observable<R>

# HttpRequest-Methoden

- Parameter `HttpRequest` oder:
  - `method`: string,
    - `'DELETE'|'GET'|'POST'|'PUT'`
  - `url`: string,
  - `options?`: Objekt zur detaillierten Spezifikation
- Rückgabe: `observable`



NewElements



netTrek

# Request-Optionen

- body?: any;
- headers?: HttpHeaders;
- params?: HttpParams;
- reportProgress?: boolean
- withCredentials?: boolean



# Request-Optionen

- responseType: 'arraybuffer' | 'blob' | 'json' | 'text';
- observe: 'body' | 'events' | 'response'
- Beide Parameter bestimmen Rückgabetyt für den Request

observe

responseType

return

body

arrayBuffer

Observable<ArrayBuffer>

body

blob

Observable<Blob>

body

text

Observable<string>

body

json

Observable<Object | R>



# Request-Optionen

observe	responseType	return
events	arrayBuffer	Observable<HttpEvent<ArrayBuffer>>
events	blob	Observable<HttpEvent<Blob>>
events	text	Observable<HttpEvent<string>>
events	json	Observable<HttpEvent<Object   R>>
response	arrayBuffer	Observable<HttpResponse<ArrayBuffer>>
response	blob	Observable<HttpResponse<Blob>>
response	text	Observable<HttpResponse<string>>
response	json	Observable<HttpResponse<Object   R>>



# Response-Typen

- `HttpResponse`
  - `body: T | null`
  - `headers: HttpHeaders`
  - `status: number`
  - `statusText: string`
- `url: string | null`
- `ok: boolean`
- `type: EventType.Response`



# Response-Typen

- HttpEvent
  - Sent-Anfrage gesendet
  - UploadProgress – Upload-Fortschrittseignis (geladen#gesamt)
  - ResponseHeader – Antwortstatuscode und Header empfangen
  - DownloadProgress – Download-Fortschrittseignis (geladen#gesamt)
  - Response – Vollständige Antwort inkl. Body
  - User – Benutzerdefinierte Ereignisse

# HTTP-Service Methoden

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



NewElements



netTrek

# HttpInterceptor

- Anforderungen und Antworten lassen sich abfangen
- Service, dass das `HttpInterceptor` Interface implementiert
  - `intercept` - Methode
    - req: `HttpRequest<any>`,
    - next: `HttpHandler`
      - `-> Observable<HttpEvent<any>>`
        - `return next.handle(req);`



# HttpInterceptor - bereitstellen

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



# HttpInterceptor - NoCache

- *// needed für IE 11*  
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);  
}



# 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

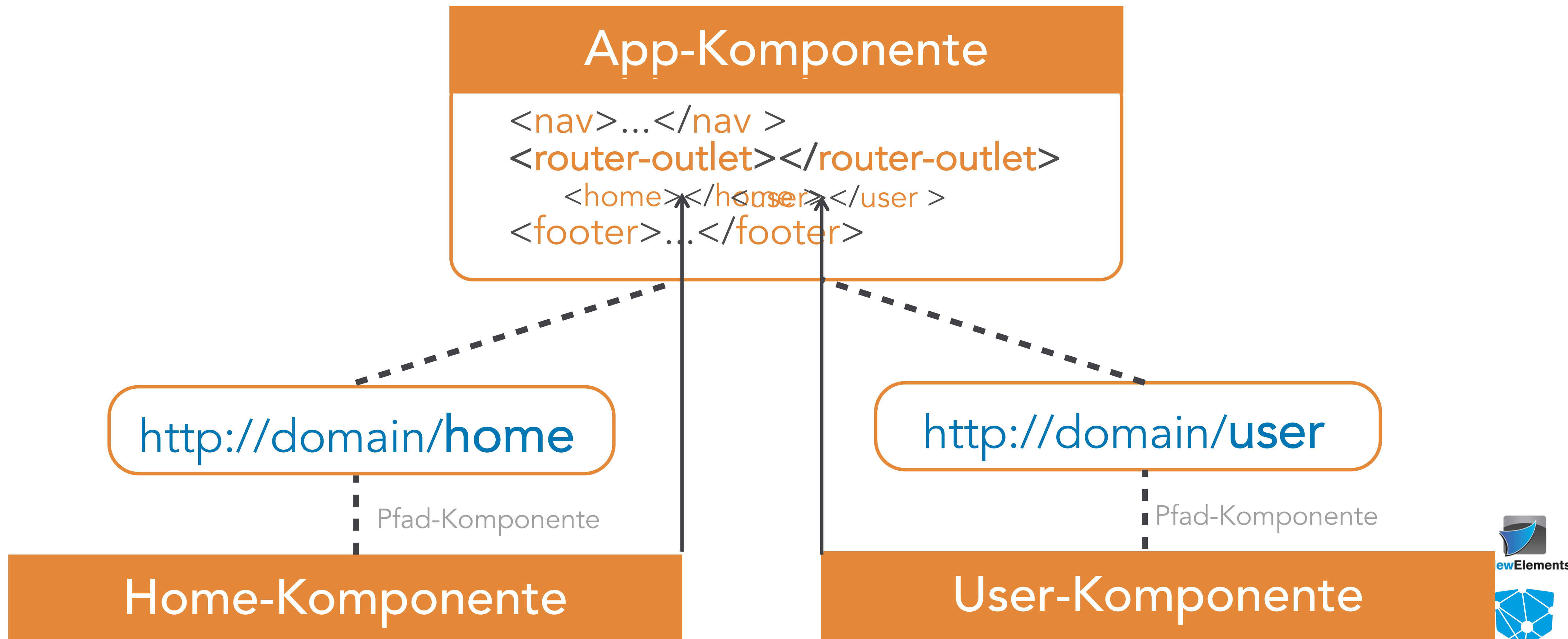


NewElements



netTrek

# 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` : `import('./dash/dash.module')`  
                  `. then(m => m.DashModule)`
- Der Pfad zu dem Modul muss importiert und die Modul-Klasse als Promise zurückgegeben werden





# 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,
    - preloadingStrategy: 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') ) )`  
    `.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  
}

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



# Named Outlet

- Outlet mit namen versehen:  
`<router-outlet name="modal"></router-outlet>`
- Pfad auf outlet-Namen beziehen  
`{ path: 'modA', component: AComponent, outlet:'modal' }`
- Navigieren
- `[routerLink]="open</a>`
- `this.router.navigate( [ { outlets: { modal: null } } ] );`



# 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

# 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.



NewElements



netTrek



## 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">...</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>
```



NewElements



netTrek

# 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 widerspiegeln.
  - `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.



# 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.



# 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



NewElements



netTrek

# 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"  
    </input>  
  </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
  - Drittes Element: AsyncValidator | AsyncValidator-Array



NewElements



netTrek

# Formulare – Helfer – 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 ]  
  })  
});
```



NewElements



netTrek

# 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



# 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



# 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



# 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 – Helfer – 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; })`



NewElements



netTrek

**Veröffentlichen**  
JIT, AOT und mehr

# JIT

## Server

Vorlagen  
Decorators  
Styles

## Browser

Kompilieren im Browser (Laufzeit)

Parse

View  
Code  
(AST)

Eval  
JS

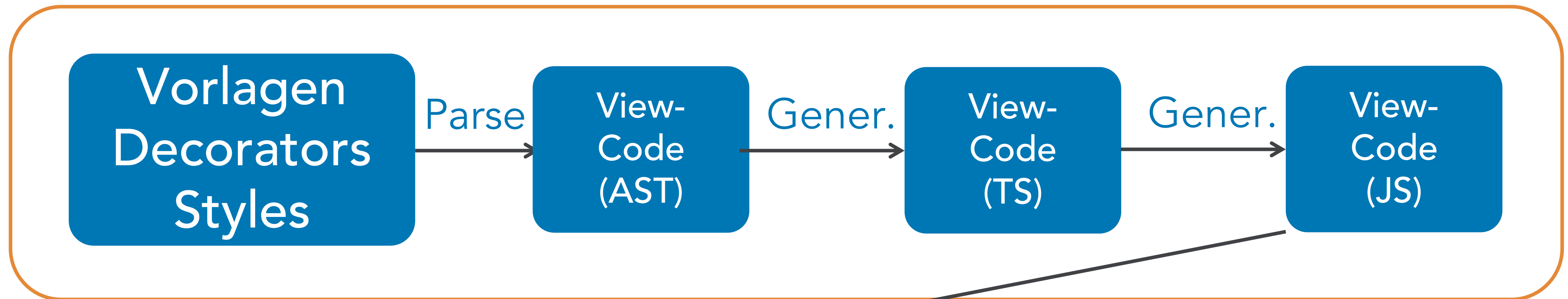
View-  
Klassen

new

Laufende  
Anwendung

# 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



NewElements



netTrek

# 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 $\geq$ 6)

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

# Veröffentlichen (Enviroment)

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



NewElements



netTrek



# Veröffentlichen – Packet Analyse

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



NewElements



netTrek