# Samenvatting van Udemy cursus

# Angular 5 (formerly Angular 2) - The Complete Guide

Voorbeelden: C:\VDAB\_angular\udemymaxschwarz en <https://github.com/vdab-pierre/udemyAng.git>

## section5 components & databinding deep dive

### 68 local refs in templates

Ipv 2 way-binding kunnen we hier local refs gebruiken

#blah op het elmt en dan kan blah gebruikt worden in de template, zoals doorgegeven worden aan component functions, … Pas op blah is het html element!! Dus bv voor de waarde .value gebruiken. (zie serverNameInput in cockpit.component.html en in de ts file)

### 69 getting access to the template & dom through viewchild

Maar we kunnen blah niet zomaar in de ts file gebruiken. Daarvoor moeten we een field hebben:

@ViewChild(‘serverContentInput’) serverContentInput;

en dan in een function bv:

this.serverContentInput.nativeElement.value

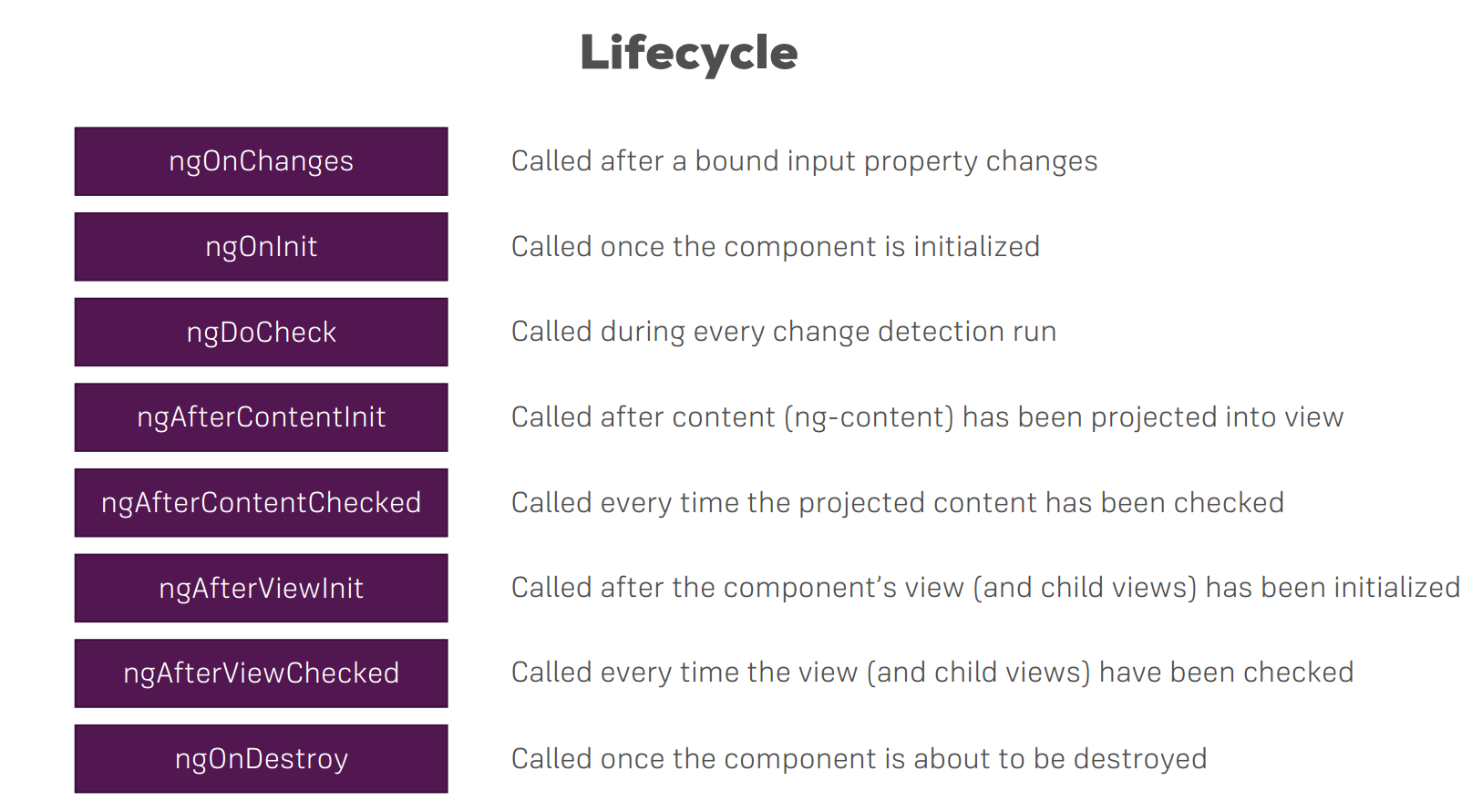
voorbeeld ook in cockpit.component.html en ts, met serverContentInput

Niet gebruiken om van de ts naar de html de waarde te veranderen zoals op deze manier bv:  
this.serverContentInput.nativeElement.value = ‘weetikveel’; beter gebruik maken van string interpolation of property binding

### 70 projecting contentinto components with ng-content

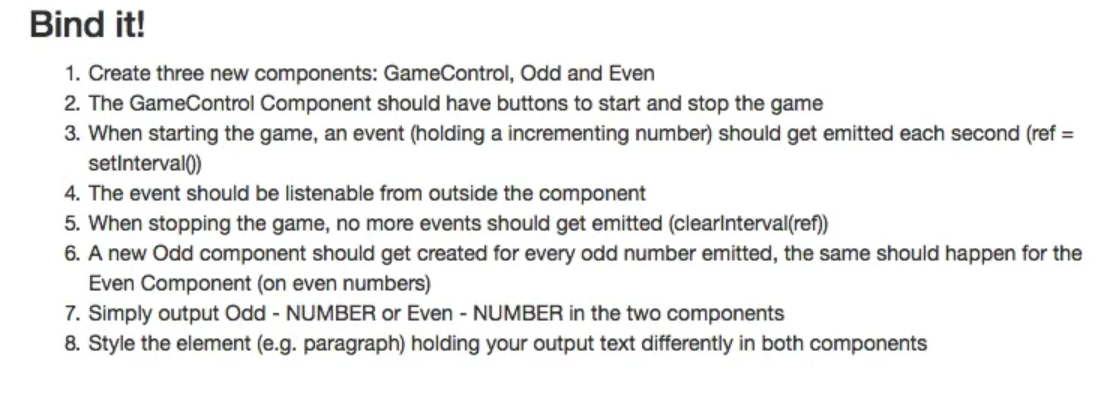
Normaal wordt alles weggedaan dat tussen de selector (waar de componenttemplate moet gedisplayed worden) staat, maar als je <ng-content></ng-content> tussen de begin en eindtag van de selector plaatst wordt dat wel weergegeven. Zie app.component.html en server-element.component.html.

### 71 Component lifecycle



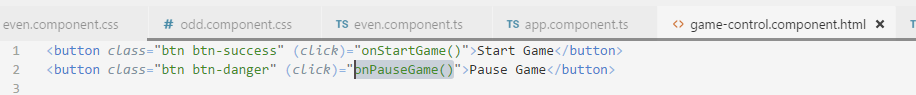
### Assignment 4

Solution: cmp-databinding-assignment-start



Dus eerst maken we nieuwe components aan, met de cli bv *ng generate component GameControl* enz. voor de andere componenten.

Dan plaatsen we 2 knoppen in de template ervan.

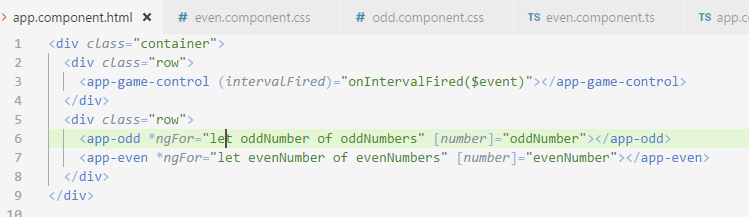


In de Start Game button binden we de *(click)* event met een component function *onStartGame*().



*In de GameControlComponent:* er wordt een *intervalFired* event aangemaakt en met een *Output()* binding kan de parent component ernaar luisteren. In *OnStartGame()* wordt om de seconde met de javascript function *setInterval* *intervalFired* “geemit” met als inhoud een getal gelijk aan *lastNumber + 1*. *LastNumber* wordt natuurlijk ook met 1 vermeerderd. Het is immers de bedoeling dat het geproduceerde getal telkens 1 meer is.

Bij klik op Pause Game wordt de function *onPauseGame()* getriggerd. Deze stopt de interval met de javascript function *clearInterval.*



In de template van AppComponent willen app-game-control weergeven. Die luistert naar intervalFired, triggert onIntervalFired en geeft $event mee, wat het getal is.

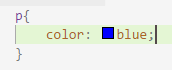


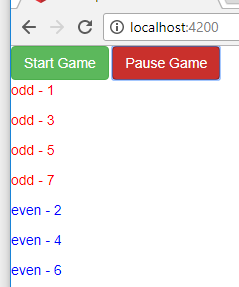
Het enige dat onIntervalFired doet is het gegeven getal (dat geïncrementeerde getal dat door de GameControlComponent wordt gegeven via het event) naargelang het even of oneven is in de juiste array zetten.

In de template van AppComponent geven we de oneven en de even getallen weer dmv de selector van de OddComponent en EvenComponent. Die geven we dan evenveel weer als er getallen zijn in de even en oneven arrays. En geven het even of oneven getal mee door middel binding. Hier kunnen we idd binden met number omdat number een Input() variabel is in de Odd – en EvenComponent.



Dankzij style encapsulation kunnen we de de <p> van odd anders stylen dan die van even. Dat doen we in even.component.css en odd.component.css.





### 77 Adding navigation with eventbinding and \*ngif

Niet echt de beste manier om te navigeren maar ook ok.

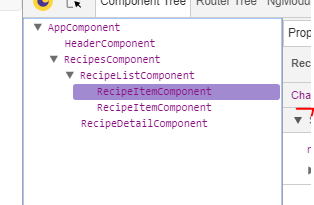
We zijn terug in de ‘Recipes’ applicatie waar recipes en shopping list componenten worden weergegeven.

* In de header (html) kunnen we de (click) event van elk menu verbinden met een function in de ts, we geven bv recipe voor het ene en shoppinglist voor het andere.   
   <li><a href="#" (click)="onMenuclicked('recipes')">Recipes</a></li>  
   <li><a href="#" (click)="onMenuclicked('shoppinglist')">Shopping List</a></li>
* In de ts file maken we een veld voor het event aan en zorgen ervoor met output dat de parent ernaar kan luisteren:  
  @Output() menuclicked = new EventEmitter<string>();
* In de function verbonden aan het click event in het menu emitten we we dat event met dus recipe of shoppinglist naargelang wat er geklikt is geweest.  
  onMenuclicked(menu:string){  
   this.menuclicked.emit(menu);  
   }
* We zorgen er dan voor dat de appcomponent naar dat event luistert en een function triggert op dat moment  
  <app-header (menuclicked)="onHeaderMenuClicked($event)"></app-header>
* We maken een function en een veld aan in de ts, de function wordt aangeroepen elke keer dat de event meniclicked gebeurt en het veld gaan we gebruiken voor het al dan niet weergeven van van shopping list en/of recipes  
   *menu:string;  
   onHeaderMenuClicked(menu){  
   this.menu=menu;  
  }*
* Het al dan niet weergeven van de shopping list en van de recipes geburt in de template van app component  
  <app-recipes \*ngIf="menu==='recipes'"></app-recipes>  
  <app-shopping-list \*ngIf="menu==='shoppinglist'"></app-shopping-list>

### 79 Passing Recipe Data with property binding

We lussen in de recipes in de list component met ngFor in de recipeitem component selector en binden de r variable met het recipe veld van de recipe item component die we via interpolation in de template tonen.

Een click op een recipe in de lijst geeft de detail weer rechts.   
Onze applicatie is opgebouwd uit een AppComponent met daarin een HeaderComponent (het menu) en een RecipesComponent. De RecipesComponent bestaat uit een RecipeListComponent die een lijst is van RecipeItemComponents en een RecipeDetailComponent.



Als we op een recept klikken in de lijst klikken we op een RecipeItemComponent. Daar moeten moeten we de click al opvangen.  
<a href="#" class="list-group-item clearfix" (click)="onSelected()">  
 @Output() recipeSelected= new EventEmitter<void>();  
onSelected(){  
 this.recipeSelected.emit();  
 }

De parent, RecipeListComponent kan luisteren naar recipeSelected (@Output).  
<app-recipe-item \*ngFor="let r of recipes" [recipe]="r" (recipeSelected)="onRecipeSelected(r)"></app-recipe-item>

En emit dan op zijn beurt een event (recipeClicked) en kan een recept ineens meegeven. De RecipesComponent kan daar naar luisteren en dat event (het recept) ineens aan RecipeDetailComponent meegeven.  
<app-recipe-list (recipeclicked)="selectedRecipe=$event"></app-recipe-list>

In RecipeDetail hebben we dan een @Input property die we dan kunnen gebruiken om de naam, description, etc van dat recept via interpolation weer te geven.

### 80 Allowing the user to add ingredients to the shopping list

Tijdelijk kunnen we dit oplossen met wat we kennen maar normaal zouden we dat met Forms oplossen.

In ShoppingListEditComponent geven we de inputs een local reference in de template (bv #nameInput en #amountInput). We kunnen hier werken met @ViewChild om in de ts file toegang te hebben tot die local template references.

We emitten een ingredientAdded event on click op de Add button, die een ingredient mee zal geven.

In de ShoppingListComponent luisteren we naar dat event en triggeren een function die dat event (een nieuw ingredient) aan de ingredients toevoegt via een array push.

## Section 7 Directives Deep Dive

Attributes directives sit on attributes and effect only that attribute, structural change the dom.

Only one structural directive on an element.

### 83 ngClass and ngStyle recap

[ngClass]="{odd: odd % 2 !== 0}"

[ngStyle]="{backgroundColor: odd % 2 !== 0 ? 'yellow' : 'transparent'}"

### 84 custom directives

We maken een nieuwe file in een nieuwe folder onder app met bv de naam basic-highlight.directive.ts.

import { Directive,ElementRef, OnInit } from "@angular/core";

@**Directive**({

selector: '[appBasicHighlight]'

})

export class **BasicHighlightDirective** implements **OnInit**{

*/\*\**

*\**

*\*/*

constructor(private elementRef:**ElementRef**) {}

**ngOnInit**() {

this.elementRef.nativeElement.style.backgroundColor='green';

}

}

We voegen BasicHighlightDirective aan de declaration array van AppModule.

We gebruiken de directive door op element het attribuut toe te voegen. De naam van dat attribuut is de selector zonder de vierkanten haken. Die vierkanten haken geven Angular mee in de definitie van de directive dat hij het element moet selecteren met dat attribuut. Cfr css selectors in jquery bv.

<p *appBasicHighlight*>Style with BasicHighlightDirective</p>

### 85 Using the Renderer to create a better directive

Het is een better practice de renderer te gebruiken omdat we de dom daar niet mee wijzigen en we hebben er niet altijd toegang toe.

import { Directive, Renderer, OnInit,ElementRef } from '@angular/core';

@**Directive**({

selector: '[appBetterHighlight]'

})

export class **BetterHighlightDirective** implements **OnInit** {

constructor(private elref: **ElementRef**, private renderer: **Renderer**) { }

**ngOnInit**() {

this.renderer.**setElementStyle**(this.elref.nativeElement,'background-color','blue');

}

}

Het gebruik ervan is hetzelfde.

### 87 Using HostListener to listen to Host Events

We kunnen reageren op events in de client met HostListener.

import { Directive, Renderer2, OnInit,ElementRef, HostListener } from '@angular/core';

@**Directive**({

selector: '[appBetterHighlight]'

})

export class **BetterHighlightDirective** implements **OnInit** {

constructor(private elref: **ElementRef**, private renderer: **Renderer2**) { }

**ngOnInit**() {

*//this.renderer.setStyle(this.elref.nativeElement,'background-color','blue');*

}

@**HostListener**('mouseenter') **mouseover**(eventData:**Event**){

this.renderer.**setStyle**(this.elref.nativeElement,'background-color','blue');

}

@**HostListener**('mouseleave') **mouseleave**(eventData:**Event**){

this.renderer.**setStyle**(this.elref.nativeElement,'background-color','transparent');

}

}

### 88 Hostbinding

Een andere manier om het vorige te doen zonder Renderer is Hostbinding.

We hebben een property gemarkeerd als HostBinding en in de method zetten we die.

@**HostBinding**('style.backgroundColor') backgroundColor: string = 'transparent';

@**HostListener**('mouseenter') **mouseover**(eventData: **Event**) {

*//this.renderer.setStyle(this.elref.nativeElement,'background-color','blue');*

this.backgroundColor = 'blue';  
}

### 89 Binding to directive properties

Indien we willen dat de default en highlight achtergrondkleur kunnen meegegeven worden dan moeten daar input properties voor voorzien, ze gebruiken on mouseover en leave en ze dan ook binden in app.component.html. We zullen ook default waarden meegeven.

@**Input**() defaultColor:string='transparent';

@**Input**() highlightColor:string='blue';

@**HostListener**('mouseenter') **mouseover**(eventData: **Event**) {

*//this.renderer.setStyle(this.elref.nativeElement,'background-color','blue');*

this.backgroundColor = this.highlightColor;

}

@**HostListener**('mouseleave') **mouseleave**(eventData: **Event**) {

*//this.renderer.setStyle(this.elref.nativeElement,'background-color','transparent');*

this.backgroundColor = this.defaultColor;

}

<p *appBetterHighlight* [*defaultColor*]="'yellow'" [*highlightColor*]="'red'">Style with BetterHighlightDirective</p>

We geven een default background backgroundColor bij het initialiseren van de component.

ngOnInit() {

this.backgroundColor=this.defaultColor;

*//this.renderer.setStyle(this.elref.nativeElement,'background-color','blue');*

}

### 90 Behind the scenes on structural directives

\* Indicates it’s a structural directive, bv voor \*ngIf gebruik deze template indien … :

<ng-template [*ngIf*]="!onlyOdd">  
 <div>  
 <ul *class*="list-group">  
 <li *class*="list-group-item" \**ngFor*="let even of evenNumbers">  
 {{even}}  
 </li>  
 </ul>  
 </div>  
</ng-template>

### 91 Building a structural directive

@**Input**() set **unless**(){

}

unless is een input **property** maar gemarkeerd als een setter method dankzij keyword set.

De directive:

import { Directive, Input, TemplateRef,ViewContainerRef} from '@angular/core';

import { } from '@angular/core/src/linker/view\_container\_ref';

@**Directive**({

selector: '[appUnless]'

})

export class **UnlessDirective** {

@**Input**() set **appUnless**(condition:boolean){

if (!condition) {

this.vcref.**createEmbeddedView**(this.templateRef)

} else {

this.vcref.**clear**();

}

}

constructor(private templateRef:**TemplateRef**<any>,private vcref:**ViewContainerRef**) { }

}

Gebruikt in de html:

<div \**appUnless*="onlyOdd">

<ul *class*="list-group">

<li *class*="list-group-item" \**ngFor*="let even of evenNumbers">

{{even}}

</li>

</ul>

</div>

### 92 ngSwitch

ngSwitch, ngSwitchCase en ngSwitchDefault kunnen gebruikt worden om afhankelijk van een waarde een bepaalde html weer te geven, dus bv als we een value veld zouden hebben in de component, dan zouden we dit in de html kunnen gebruiken:

<div [*ngSwitch*]="value">

<p \**ngSwitchCase*="5">value is 5</p>

<p \**ngSwitchCase*="10">value is 10</p>

<p \**ngSwitchCase*="100">value is 100</p>

<p \**ngSwitchDefault*>value is default</p>

</div>

### 93 Building and using a dropdown directive

Dropdowns hebben we in de header, de ‘manage’ button, en in recipe details, de manage recipe button. Deze divs hebben al een bootstrap class nl ‘btn-group’, indien we ook de ‘open’ class toevoegen, dan is de dropdown open. Kunnen toggelen is het gedrag dat we willen.

We maken een directive die de class toevoegt of wegdoet.

import { Directive, HostListener, HostBinding } from '@angular/core';

@**Directive**({

selector: '[appDropdown]'

})

export class **DropdownDirective** {

@**HostBinding**('class.open') isOpen = false;

@**HostListener**('click') **toggleOpen**() {

this.isOpen = !this.isOpen;

}

constructor() { }

}

En markeren de betreffende div en li met deze nieuwe directive:  
in recipe-detail

<div *class*="btn-group" *appDropdown*>

en in de header

<li *class*="dropdown" *appDropdown*>

## Section 9: Services & Dependency Injection

### 94 intro

Typical use cases: duplication of code, data storage, logging …

### 96&97 creating a logging service & inject

De service:

**import** { Injectable } **from** '@angular/core';  
  
@Injectable()  
**export class** LoggingService {  
 logStatusChange(status: **string**) {  
 console.log('A server status changed, new status: ' + status);  
 }  
 **constructor**() { }  
}

en in onze component, we injecteren de LoggingService in de constructor. Dus angular zal daar voor ons een instance van maken, maar daarvoor moet angular die class in de providers array kunnen vinden. Die instance gebruiken we om te loggen.

**import** {Component, EventEmitter, Output} **from** '@angular/core';  
**import** {LoggingService} **from** '../logging.service';  
  
  
@Component({  
 selector: 'app-new-account',  
 templateUrl: './new-account.component.html',  
 styleUrls: ['./new-account.component.css'],  
 providers:[LoggingService]  
})  
**export class** NewAccountComponent {  
 @Output() accountAdded = **new** EventEmitter<{ name: **string**, status: **string** }>();  
  
 **constructor**(**private** logging: LoggingService) {  
 }  
  
 onCreateAccount(accountName: **string**, accountStatus: **string**) {  
 **this**.accountAdded.emit({  
 name: accountName,  
 status: accountStatus  
 });  
 **this**.logging.logStatusChange(accountStatus);  
 }  
}

### tem 100

… Services are hierarchical … de Providers array vertelt angular welke instances en de injection zelf (bv in de ctor) maakt die aan.  
Dus stel dat we in de providers array van AppComponent BrolService zouden vermelden dan zal Brol Service kunnen geïnstantieerd, ge¨njecteert eigenlijk in de ctor van subcomponent van AppComponent en dat zonder BrolService opnieuw in de providers array van die componenten want daarmee zouden we telkens een nieuwe instance aanmaken.

### 101 Injecting services into services

Indien we a service in een service willen injecteren moeten we die service van metadata voorzien, nl ‘@Injectable()’. Daarmee vertellen we angular dat iets hierin kan geïnjecteerd worden. Je voegt dus @Injectable aan de service waar je een aandere service in injecteert en NIET aan de service die geïnjecteert wordt. (‘the receiving service’)

### 102 Using services for cross component communication

We zouden met events kunnen werken een event triggeren in het ene component, het andere luistert en geeft het event door via property binding … ingewikkeld! Ipv daarvan heeft de service een een eventemitter, de component emit een event, een ander component luistert en reageert …

We willen dat een alert tevoorschijn komt indien de status van een account gezet wordt.

We voegen een eventemitter toe aan onze AccountService

…

statusUpdated = **new** EventEmitter<**string**>();  
…

We laten dan AccountComponent de event emitten.

…

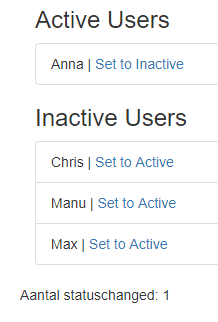
onSetTo(status: **string**) {  
 **this**.accountService.updateStatus(**this**.id, status);  
 /\*this.logging.logStatusChange(status);\*/  
 **this**.accountService.statusUpdated.emit(status);  
}  
…

En laten AppComponent luisteren en bv een alert doen.

**…**

**constructor**(**private** accountsService: AccountService) {  
 **this**.accountsService.statusUpdated.subscribe(  
 (status: **string**) => alert('New Status: ' + status)  
 );  
}  
…

### Assignment services



Een klik op een gebruiker in één van die lijsten verwijdert die van die lijst en zet hem in de andere. Het aantal veranderingen wordt afgebeeld vanonder.  
Momenteel wordt alles geregeld met input properties en events. De opdracht is: De app optimaliseren dmv services, en ook een CounterService voorzien die gebruikt wordt in de UserService, kwestie van de theorie in praktijk te brengen ;)

We maken twee services aan.  
We registreren ze in de providers array van AppComponent ( de componenten die daar gebruik van gaan maken zullen toch children zijn van AppComponent) of in AppModule (dan zijn die services app-wide te gebruiken). We injecteren ze in de betreffende componenten.

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

import { CounterService } from './counter.service';

@**Injectable**()

export class **UsersService** {

numOfStatusChanges: number;

activeUsers = ['Max', 'Anna'];

inactiveUsers = ['Chris', 'Manu'];

constructor(private counterService: **CounterService**) { }

**setToInactive**(id: number) {

this.inactiveUsers.**push**(this.activeUsers[id]);

this.activeUsers.**splice**(id, 1);

this.counterService.counter++;

this.counterService.counterChanged.**emit**(this.counterService.counter);

}

**setToActive**(id: number) {

this.activeUsers.**push**(this.inactiveUsers[id]);

this.inactiveUsers.**splice**(id, 1);

this.counterService.counter++;

this.counterService.counterChanged.**emit**(this.counterService.counter);

}

}

import { Injectable,EventEmitter } from '@angular/core';

@Injectable()

export class CounterService {

counter:number=0;

counterChanged=new EventEmitter<number>();

}

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

import { UsersService } from '../users.service';

@**Component**({

selector: 'app-active-users',

templateUrl: './active-users.component.html',

styleUrls: ['./active-users.component.css']

})

export class **ActiveUsersComponent** implements **OnInit** {

users:string[];

constructor(private usersService: **UsersService**) { }

**onSetToInactive**(id: number) {

this.usersService.**setToInactive**(id);

this.usersService.numOfStatusChanges++;

}

**ngOnInit**(){

this.users=this.usersService.activeUsers;

}

}

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

import { UsersService } from '../users.service';

@**Component**({

selector: 'app-inactive-users',

templateUrl: './inactive-users.component.html',

styleUrls: ['./inactive-users.component.css']

})

export class **InactiveUsersComponent** implements **OnInit** {

users:string[];

constructor(private usersService:**UsersService**) {}

**onSetToActive**(id: number) {

this.usersService.**setToActive**(id);

}

**ngOnInit**(){

this.users=this.usersService.inactiveUsers;

}

}

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

import { CounterService } from './counter.service';

@**Component**({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class **AppComponent** implements **OnInit** {

aantStatusChanged: number;

constructor(private counterService: **CounterService**) {

this.counterService.counterChanged.**subscribe**(

(counter: number) => this.aantStatusChanged = counter

);

}

**ngOnInit**() {

this.aantStatusChanged=this.counterService.counter;

}

}

De html wordt ook opgekuisd, we hebben immers geen input property binding meer nodig.

<div *class*="container">

<div *class*="row">

<div *class*="col-xs-12 col-md-8 col-md-offset-2">

<app-active-users></app-active-users>

<app-inactive-users></app-inactive-users>

</div>

<div *class*="row">

<div *class*="col-xs-12 col-md-8 col-md-offset-2">

Aantal statuschanged: {{aantStatusChanged}}

</div>

</div>

</div>

Wat de counter betreft. We laten AppComponent luisteren naar een EventEmitter van de CounterService die de counter doorgeeft als deze verandert. AppComponent update een property aantStatusChanged elke keer, die property wordt via interpolation in de html weegegeven. Om ervoor te zorgen dat we de eerste keer laden ook de counter zien voorzien we hetzelfde in de OnInit.

### 107 en 108 ShoppingListService

In dit hoofdstuk wordt het gebruik van services om het communiceren tussen componenten verdergezet. Als we naar de shopping list surfen zien de lijst van de ingredienten onder het scherm om een ingredient toe te voegen. Wat we moeten doen om met services hetzelfde te bereiken is een ShoppingListService toevoegen die de ingredients bijhoudt, een function toevoegen om ze te krijgen en nog een function om er één toe te voegen.

We voegen de service toe aan de providers array van een component hoog genoeg in de hiërarchie om geïnjecteerd te kunnen worden in de shopping list compnenten, maar ook in de recipe componenten, dus vermoedelijk (we zullen zien ;)) is het voldoende om dat te doen in AppComponent (anders is het AppModule maar ik vind het een beetje overkill, AppModule is immers voor de ganse app).

import { Injectable, EventEmitter } from '@angular/core';

import { Ingredient } from '../shared/ingredient';

@**Injectable**()

export class **ShoppingListService** {

ingredientsChanged = new **EventEmitter**<**Ingredient**[]>();

private ingredients:**Ingredient**[]=[

new **Ingredient**("Apples",5),

new **Ingredient**("Tomatoes",10)

];

constructor() { }

**getIngredients**(){

return this.ingredients.**slice**();

}

**addIngredient**(ingr:**Ingredient**){

this.ingredients.**push**(ingr);

this.ingredientsChanged.**emit**(this.ingredients.**slice**());

}

}

…

@**Component**({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css'],

providers:[ShoppingListService]

})

…

We injecteren de service in de ShoppingListComponent. We hebben nog steeds een ingredients array nodig maar gewoon leeg van het type Ingredient[] en vullen ingredients met de ingredients van de service.

…

export class **ShoppingListComponent** implements **OnInit** {

ingredients:**Ingredient**[];

constructor(private shoppingListService: **ShoppingListService**) { }

**ngOnInit**() {

this.ingredients=this.shoppingListService.**getIngredients**();

…

We zouden zo de lijst terug moeten krijgen.

Om een ingredient toe te voegen wordt op de knop Add geklikt in de ShoppingListEditComponent. Daar moeten we nu de service gaan gebruiken.

import { Component, OnInit,ViewChild,ElementRef } from '@angular/core';

import { Ingredient } from '../../shared/ingredient';

import { ShoppingListService } from '../shopping-list.service';

@Component({

selector: 'app-shopping-list-edit',

templateUrl: './shopping-list-edit.component.html',

styleUrls: ['./shopping-list-edit.component.css']

})

export class ShoppingListEditComponent implements OnInit {

@ViewChild('nameInput') nameInputRef:ElementRef;

@ViewChild('amountInput') amountInputRef:ElementRef;

constructor(private shoppingListService: ShoppingListService) { }

ngOnInit() {

}

onAdd(){

const ingrName=this.nameInputRef.nativeElement.value;

const ingrAmount=this.amountInputRef.nativeElement.value;

const newIngredient = new Ingredient(ingrName,ingrAmount);

this.shoppingListService.addIngredient(newIngredient);

}

}

De Output eventEmitter property mag weg, de imports ook, en in de HTML moeten we er ook niet meer naar luisteren.

Dat zou moeten werken … maar we zien ons toegevoegd ingredient niet in de lijst eronder!

We hebben immers een copy gekregen (slice()) en niet de originele array van de service. We zouden natuurlijk ook de slice() kunnen wegdoen, dat zou niet per se fout zijn als we toch echt geen wijzigingen aan die array waanbrengen, maar het is wel best pratice van zo te werken. In plaats daarvan kunnen we een event emitten iedere keer een ingredient aan de array wordt toegevoegd en ernaar luisteren als we de lokale ingredients in ShoppingListComponent initialiseren.

import { Injectable, EventEmitter } from '@angular/core';

import { Ingredient } from '../shared/ingredient';

@**Injectable**()

export class **ShoppingListService** {

ingredientsChanged = new **EventEmitter**<**Ingredient**[]>();

private ingredients:**Ingredient**[]=[

new **Ingredient**("Apples",5),

new **Ingredient**("Tomatoes",10)

];

constructor() { }

**getIngredients**(){

return this.ingredients.**slice**();

}

**addIngredient**(ingr:**Ingredient**){

this.ingredients.**push**(ingr);

this.ingredientsChanged.**emit**(this.ingredients.**slice**());

}

}

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

import { Ingredient } from '../shared/ingredient';

import { ShoppingListService } from './shopping-list.service';

@**Component**({

selector: 'app-shopping-list',

templateUrl: './shopping-list.component.html',

styleUrls: ['./shopping-list.component.css']

})

export class **ShoppingListComponent** implements **OnInit** {

ingredients:**Ingredient**[];

constructor(private shoppingListService: **ShoppingListService**) { }

**ngOnInit**() {

this.ingredients=this.shoppingListService.**getIngredients**();

this.shoppingListService.ingredientsChanged.**subscribe**(

(ingredients:**Ingredient**[])=>{

this.ingredients=ingredients;

}

);

}

}