OEFENINGEN ANGULAR FUNDAMENTALS

01 – EENVOUDIG BEGINNEN

1. Het algemene Github‐adres met voorbeeldcode is <https://github.com/petereijgermans11/workshop-angular7>

b) Maak/start een eigen Hello World‐app. Dit kan op verschillende manieren:

o Gebruik als basis 100-helloworld.

o OF: ga zelf naar cli.angular.io, installeer deze tool en maak een nieuw project en start deze. Je vindt deze ook in de Angular QuickStart, op https://angular.io/guide/quickstart.

o Codevoorbeeld: 100-helloworld.

c) Probeer eens of je een nieuwe component kunt maken en deze bootstrappen. Volg deze stappen:

o *Handmatig?* ‐ maak een nieuw bestand, bijvoorbeeld \app\nieuwe.component.ts.

Importeer de juiste afhankelijkheden.

Schrijf een **selector** en een **HTML‐template**.

o *Of gebruik de Angular CLI?* ‐ gebruik de volgende opdracht in de root van het project:

ng generate component <component-naam>

o Pas de bootstrapper (= app.module.ts) aan, zodat daarin het nieuwe component wordt geïmporteerd en gestart.

o Pas de index.html aan, zodat de juiste **selector** wordt gebruikt.

o Draai npm install **en** npm start om je nieuwe component te testen.

.

02 – DATABINDING

a) Breid je app uit oefening 1 (Hello World) uit met een field/property. Bindt deze property in het template dat gebruikt wordt.

o Doe dit expliciet met aparte declaratie en initialisatie in de ngOnInit()

lifecycle-hook

o Lees de werking en de volgorde van Angular lifecycle hooks op https://angular.io/guide/lifecycle-hooks.

o Codevoorbeeld (Solution): 101-databinding.

b) Maak een array van objecten. Bind ze in het template met de ***directive*** \*ngFor.

o Geef via TypeScript expliciet op dat de property uit een array van strings of een array van objecten moet bestaan (maak zelf deze keuze).

o Codevoorbeeld: 101-databinding.

c) Maak expliciet een Model voor de inhoud van je array. Dat mag een object met één of meerdere properties zijn. Zorg er voor dat de inhoud van de array bestaat uit objecten van het type <Model>.

o Codevoorbeeld: 101-databinding‐ citiesVolgensModel

.

d) Bedenk zelf een toepassing voor de directive \*ngIf en gebruik deze in combinatie met je class en properties.

e) Breng de template naar een **externe** .html‐file. Gebruik vervolgens in @Component de property

templateUrl om hier naar te verwijzen. Test of de databinding blijft werken.

03 – EVENT BINDING

a) Voeg een element met event‐binding toe aan je applicatie. Bijvoorbeeld een (click)op een knop afvangen. Roep een event handler in de component aan als de event optreedt. Laat deze bijvoorbeeld een alert()tonen of een console.log()‐melding.

o Codevoorbeeld: 102-eventbinding.

b) Test meerdere typen DOM‐events, bijvoorbeeld blur, focus, keypress, mousemove, enzovoort.

Zie eventueel Mozilla.org (https://developer.mozilla.org/en-US/docs/Web/Events) voor een overzicht.

c) Maak een formulierveld (bijvoorbeeld een tekstveld) met een ***local template variable***. De notatie hiervoor is #myFieldName. Geef de variabele door aan de event handler en verwerk hem in een alert(), console.log()‐melding, om een waarde in het DOM te updaten, enzovoort.

d) Maak een eenvoudige client‐sided CRUD‐applicatie: users kunnen elementen toevoegen aan een array

(namen, producten, enzovoort) en verwijderen uit de array.

o Gebruik voor verwijderen de JavaScript‐functie ArrayName.splice(…)of ArrayName

- ArrayName.filter(item => …).

Codevoorbeeld: 102-eventbinding\...\

app-02-complete.component.ts.

04 ‐ ATTRIBUTE BINDING EN TWO‐WAY BINDING

a) Maak een component met een tekstvak. Als de user een (Engelstalige) kleur in het tekstvak invult en op een knop klikt, krijgt een bijbehorende <div> deze achtergrondkleur.

o Onderzoek hoe dit werkt als de kleuren uit een serie radio buttons gekozen kunnen worden, of uit een selectielijst/dropdown.

o Algemeen codevoorbeeld attribute binding: 103-attributebinding en

105-selectielijst-change

b) Optioneel: onderzoek zelf hoe de Angular‐concepten *class binding* en *style binding* worden gebruikt.

Aanwijzingen hiervoor staan op <https://angular.io/guide/template‐syntax#ngclass>.

c) Maak een tekstveld in je component met two‐way binding. De syntaxis hiervoor is [(ngModel)].

Toon de waarde van de ingevoerde tekst in de pagina.

o Denk aan het toevoegen van FormsModule aan app.module.ts.

o Maak een kopieerfunctie: Maak twee tekstvakken op de pagina. Tekst die in het ene tekstvak wordt ingevuld, verschijnt ook in het andere tekstvak.

o Codevoorbeeld: 104-twowaybinding.

05 ‐ SERVICES

a) Als het goed is heb je inmiddels enkele componenten met hierin diverse data. Verplaats deze data vanuit de component naar een service.

o Injecteer de service in de constructor van de component waarin je de data wilt gebruiken.

o Denk aan de property [providers] in de [@ngModule‐annotatie.](mailto:@ngModule)

o Codevoorbeeld : 200-services-static.

b) Werken met async‐services: maak zelf een .json‐bestand met data en laadt dit in je applicatie. Denk onder meer aan:

o het injecteren van HttpClientModule in de module;

o het aanpassen van de component waarin de service wordt geconsumeerd:

private http:HttpClient injecteren in de constructor

en de observable‐notatie met:.subscribe()gebruiken.

* Codevoorbeelden: 0201-services-http.

c) Meer over RxJS: gebruik ook andere operators binnen .pipe(). Test bijvoorbeeld:

o retry (), delay(), eventueel filter()en andere in het Component.

* Doe dat in codevoorbeeld : 202-services-rxjs.

d) Lees de **README.md** door van 205-services-http-CRUD om de applicatie op te starten. Tevens wordt gebruik gemaakt van een *json-server*. Met een json-server kan je een json-file veranderen in een RESTFull database.

**Opdracht**: ‘getCity’, ‘deleteCity’ en ‘updateCity’ zijn niet geimplementeerd in de cityService. Maak deze werkend.

e) Werken met Live API's: gebruik een API naar keuze om live data op te halen. Maak een nieuwe applicatie en gebruik hierin deze data.

o API's zijn bijvoorbeeld beschikbaar op openweathermap.org/API of

[www.omdbapi.com.](http://www.omdbapi.com)

o Zie ook de site https://github.com/toddmotto/public‐apis en het bestand JavaScript

APIs.txt voor meer API‐endpoints.

o Codevoorbeeld: 210-services-live.

06 ‐ APPLICATIES MET MEERDERE COMPONENTEN

a) Maak een nieuwe applicatie, of ga uit van je applicatie uit de eerdere oefeningen.

b) Voeg een detail‐component toe. De details worden getoond na een muisklik op de hoofdcomponent.

o Volg het stappenplan (4 stappen) uit de presentatie, waarin de component wordt gemaakt, ingesloten in de hoofdcomponent en op de juiste wijze in HTML wordt aangeroepen.

o OF: gebruik de CLI om nieuwe component te genereren. Gebruik de selector om hem in te voegen in de HTML van AppComponent (bijvoorbeeld <mijn-nieuwe- component></mijn-nieuwe-component>).

o Codevoorbeeld: 300-components.

c) Gebruik de annotatie @Input()om data vanuit de hoofdcomponent door te geven aan de detailcomponent.

o Zorg er voor dat een object wordt doorgegeven aan de detailcomponent. Gebruik hiervoor de notatie [objName]="objData"in de template van de hoofdcomponent.

o Toon de inhoud van het object in de detailcomponent.

* Codevoorbeeld: 301-components-inputs.
* Maak ook een CityListComponent. Geef de data (lijst) door via @Input aan dit component.

d) Gebruik de annotatie @Output()om events vanuit de detailcomponent door te geven aan de hoofdcomponent. Denk bijvoorbeeld aan het doorgeven van een rating (zoals in het voorbeeld), of iets anders.

o Codevoorbeeld: 302-components-outputs.

e) **Advanced:** Complete applicatie: maak een eenvoudige eCommerce‐applicatie met de volgende onderdelen:

o Een Store. Schrijf een service voor de store die een 4 of 5 producten levert (dit mag statische data zijn).

o Een Detailcomponent. Bij klikken op een product worden details over het product getoond.

o Een Shopping Cart. De user kan het product in een winkelmandje plaatsen. De inhoud van het winkelmandje is steeds zichtbaar in de hoofdcomponent.

o Een "Order"‐knop. Als hierop wordt geklikt, wordt de inhoud van het winkelmandje getoond en het totaalbedrag berekend.

o Codevoorbeeld: 303-pubsub-ordercomponent.

07 ‐ ROUTING

a) Maak je applicatie uit de vorige oefeningen geschikt voor routing, of begin een nieuwe applicatie. De eisen zijn de volgende:

o Zorg er voor dat de applicatie een rootComponent heeft, waarin het "hoofdmenu" met opties wordt getoond.

o Zorg er voor dat verschillende componenten worden geladen als een keuze wordt gemaakt in het hoofdmenu.

o **Gebruik Codevoorbeeld**: 400-router.

b) Optioneel: maak een geheel nieuwe applicatie via de CLI en stel direct in dat deze applicatie routing gebruikt. De syntaxis hiervoor is ng new <applicatie>

o Bekijk hoe de CLI een app-routing.module.ts heeft gegenereerd en wat de syntaxis hierbij is. Bekijk ook hoe deze wordt gebruikt in de hoofdmodule app.module.ts.

c) Maak je applicatie geschikt voor het werken met route‐parameters. Denk bijvoorbeeld aan het maken van een detailroute voor een bepaald product in je applicatie. Denk aan:

o het uitbreiden van app.routes.ts met parameters;

* het insluiten van ActivatedRoute om de parameters te kunnen oppikken in de detailcomponent;
  + Detail component toevoegen aan app.module.ts !!!

o Het aanpassen van de HTML om parameters te kunnen meegeven ([routerLink]=…)

o In de detailcomponent gegevens opvragen op basis van de meegegeven parameter.

o Codevoorbeeld: 401-router-parameter.

08 ‐ PIPES

1. Test enkele standaard pipes uit en maak een custom pipe.

Gebruik code voorbeeld: 601-pipes

09 ‐ FORMULIEREN ()

**a) Goal: build a template driven form**

o Create a simple HTML5 form in a component, or **Use: /forms-template-driven, Component0-base**. And import FormsModule into your app.module.ts

o Add the local template variable #myForm="ngForm" to the <form> tag.  
o Add the directives ngModel to the separate form fields. You don't need two‐way databinding with [()].  
o Write for example myForm.value to the user interface, or show the contents of the form in an alert (or in the console) when a button is clicked.  
o Demo code available at /500-forms-template-driven, Component 1.

**b) Goal: address individual controls inside the form and add HTML5 validators.**

o Assign a local template variable to the form fields.  
o Bind ngModel to the local template variable. The code can look like: #email="ngModel"  
o Retrieve the values from the local template variable and show them in the user interface, for example its value and its validity.  
o Add the HTML5 attribute required to the form fields and see how this affects the state of the form field. Write its validity to the user interface.  
o Demo code available at /500-forms-template-driven, Component 2.

**c) Goal: combining individual form fields to an ngModelGroup**

o Add some field to the form (for example some extra text fields or checkboxes).  
o Groep them inside a <div>, assign the <div> the directive ngModelGroup. The code can look like:

<div ngModelGroup="customer" #customer="ngModelGroup">

o Run the code and identify the model group in the returned form value object.  
o Optional: set the value of a form field from inside your class, by using the local template variable and bind to [ngModel].  
o Demo code available at /500-forms-template-driven, Component 3.

**d) Goal: submitting template driven forms**

o Add a submit button to the form.  
o Make sure the submit button is only active when the form as a whole is valid. Your code can look like:

<button type="submit"

(click)="onSubmit(myForm)"

[disabled]="!myForm.valid">

...

</button>

o Demo code available at /500-forms-template-driven, Component 4.

**e) Goal: working with model driven forms**

o Start with a simple form, for example build a form on your own

o Import ReactiveFormsModule into your app.module.ts.

and Import FormGroup, FormBuilder and AbstractContro into your app.component.ts.

import {FormGroup, FormBuilder, Validators, AbstractControl} from '@angular/forms';

o Add the [formGroup]="..." directive to the <form> tag, add formControlName="..." to the individual controls.  
o Import FormGroup and FormBuilder into your class and build the form, based on the layout of your HTML.  
o Submit the form and write the value to an alert box or to the console.  
o Demo code available at /forms-model-driven, Component 1.

**f) Goal: add validation to model driven forms**

o Import Validators into your app.component.ts.

import {FormGroup, FormBuilder, Validators, AbstractControl} from '@angular/forms';

o Add ‘default’ validations to the form fields.

o Add ‘custom’ validations to the form fields. For example validate your email pattern or a passwordMatcher.  
o Demo code available at /forms-model-driven, Component 2.

**g) Unittest Reactive forms**

See: <https://codecraft.tv/courses/angular/unit-testing/model-driven-forms/>

**h) Optional:**

Add/change **rxjs/operators** to the Demo app: /forms-typeahead, Component 1