whitehorses

member of the human network

WHITEBOOK

Dynamisch opbouwen van de user interface via Angular

Juni 2017

Auteur:
Mike Heeren
INTEGRATIESPECIALIST

Inleiding

Voor sommige applicaties kan het van pas komen dat niet vooraf de gehele user interface vormgegeven hoeft te worden. Het kan namelijk handig zijn dat de user interface pas op runtime, dynamisch wordt opgebouwd. In dit Whitebook kijken we wat Angular is en hoe Angular zich voor dit soort applicaties leent. Hierbij worden ook oplossingen op de ondervonden problemen beschreven.

Een voorbeeld van een applicatie waarbij de user interface dynamisch moet worden opgebouwd is infoForms. Dit is een (back-end) engine waarin dynamisch formulierstructuren kunnen worden opgebouwd. Een ander voorbeeld is een dashboard-applicatie, waarin de gebruiker zelf aangeeft welke informatie wel en niet getoond moet worden. Het is vervolgens aan de front-end applicatie om deze structuren op runtime om te zetten in de daadwerkelijk getoonde user interface.

Vanzelfsprekend is het ene framework handiger in het opbouwen van dergelijke user interfaces, dan het andere. Het JavaScript framework Angular maakt het mogelijk om eenvoudig eigen user interface componenten te definiëren en te gebruiken. Hierdoor lijkt Angular een goed framework om dergelijke front-end applicaties op te zetten.

Wat is Angular?

Angular is het framework dat is voortgekomen uit AngularJS en werd voorheen ook wel Angular 2, of The New Angular genoemd. Hoewel Angular dus is voortgekomen uit AngularJS, heeft het nagenoeg geen overeenkomsten met zijn voorganger. Angular is niet zomaar een nieuwe versie, maar eigenlijk is men gewoon opnieuw begonnen met de ontwikkeling van het framework. Een van de vele veranderingen ten opzichte van AngularJS is de introductie van Typescript. Voor meer informatie over AngularJS, zie ook het Whitebook De Architectuur van AngularJS.

Op internet is heel veel te vinden over Angular en hoe bepaalde problemen opgelost kunnen worden. Wat het desondanks erg lastig zoeken maakt, is dat AngularJS ook vaak wordt afgekort naar Angular. Hierdoor moet men goed opletten dat de genoemde oplossing voor de juiste versie van Angular van toepassing is. Op de officiële Angular site (angular.io, en dus niet angularjs.org) is echter veel informatie te vinden.

Angular beschrijft zichzelf op deze site als een framework, waarmee cross-platform ontwikkeld kan worden en waarbij snelheid, performance en schaalbaarheid centraal staan.

Wat is TypeScript?

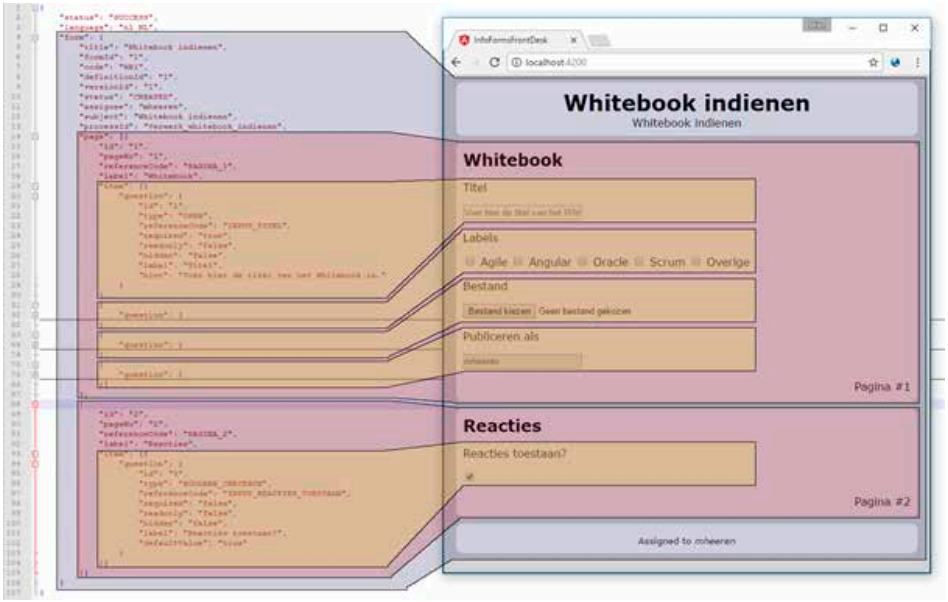
TypeScript is een superset bovenop JavaScript. Dit is ook de taal die standaard verwacht wordt door Angular bij de ontwikkeling van applicaties. Het voornaamste verschil tussen TypeScript en JavaScript is dat TypeScript type-ondersteuning biedt voor variabelen. TypeScript wordt vervolgens gecompileerd naar JavaScript, zodat de browsers ermee overweg kunnen.

Angular in de praktijk

Om te testen hoe goed Angular om kan gaan met het dynamisch opbouwen van de user interface, is een demo-applicatie gebouwd. Deze demo-applicatie consumeert een JSON over REST service, waarbij een (dynamische) formulierdefinitie wordt opgehaald. De applicatie zal de opgehaalde formulierstructuur vervolgens omzetten in (zelf gedefinieerde) user interface-componenten.

Van JSON naar user interface

De komende alinea's zal beschreven worden hoe verschillende problemen zijn opgelost in Angular. In de onderstaande afbeelding is te zien hoe de JSON structuur is opgebouwd, en hoe deze omgezet wordt naar verschillende user interface onderdelen in Angular.



Opbouw JSON mapping Angular UI

Angular CLI

Iets wat direct opvalt wanneer men begint met het ontwikkelen van Angular-applicaties, is het gemak van de Angular CLI (Command Line Interface). Deze kan worden geïnstalleerd via NodeJS en maakt het erg eenvoudig om zaken te genereren. Zo kan bijvoorbeeld de gehele structuur voor de initiële applicatie worden gegenereerd. Daarnaast is het mogelijk om individuele componenten, services en klassen te genereren.

Om de initiële applicatie te genereren, kan het volgende commando gebruikt worden:

```
ng new <applicatie-naam>
```

Wanneer vervolgens een extra component, service of klasse moet worden gegenereerd, kan ook dat via een eenvoudig commando via de CLI:

```
ng g[enerate] c[omponent] <component-locatie>
ng g[enerate] s[ervice] <service-locatie>
ng g[enerate] cl[ass] <klasse-locatie>
```

Wanneer de Angular CLI gebruikt wordt, kan door middel van het onderstaande commando worden gezorgd dat wijzigingen in de TypeScript-bestanden automatisch worden opgepakt. De TypeScript-bestanden worden dan ook direct gecompileerd naar JavaScript. De doorgevoerde wijzigingen zijn dan meteen beschikbaar en zichtbaar in de browser.

```
ng serve
```

Consumeren van JSON over REST services

Het startpunt van de demo-applicatie is dat een JSON over REST webservice wordt aangeroepen en uitgelezen. Het is erg makkelijk om JSON (JavaScript Object Notation)-structuren om te zetten naar objecten in Angular, aangezien Angular gebaseerd is op TypeScript. Ook is het eenvoudig om webservices aan te roepen in Angular.

Om dit te realiseren moeten eerst de klassen gegenereerd worden, die overeenkomen met de JSON-structuur die de webservice teruggeeft. Vervolgens kan de service worden gegenereerd via de CLI. De implementatie van de service kan daarna als volgt worden opgezet:

```
import { Injectable } from '@angular/core';
import { Http } from '@angular/http';
import { GetFormResponse } from '../get-form-response/get-form-response';
// De klasse die overeenkomt met de JSON structuur.
import { Observable } from 'rxjs/Rx';
import 'rxjs/Rx'; // Benodigd voor de map functie.
@Injectable()
export class InfoFormsService {
private static BASE_URL = 'http://localhost:4200/assets/mock/';
constructor(private http: Http) { }
public getFormResponse(): Observable<GetFormResponse> {
return this.http.get(InfoFormsService.BASE_URL + 'getFormResponse.json')
.map((res) => res.json() as GetFormResponse);
}
}
```

Wat hierbij vooral opvalt is dat de getFormResponse() methode geen object van het type GetFormResponse oplevert, maar een Observable. Deze Observable kan vervolgens op de onderstaande wijze worden geconsumeerd:

```
import { Component } from '@angular/core';
import { InfoFormsService } from './service/info-forms.service'; // De
service uit de vorige snippet.
import { Form } from './get-form-response/form'; // Het deel van de JSON
structuur/JavaScript objecten dat we nodig hebben voor de user interface.
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css'],
providers: [ InfoFormsService ] // De service wordt ook als provider
gedefinieerd.
})
export class AppComponent {
protected form: Form;
// De service moet ook via de constructor worden meegegeven
constructor(private infoFormsService: InfoFormsService) { }
ngOnInit() {
this.infoFormsService.getFormResponse().subscribe(
result => {
if (result.status === 'SUCCESS') {
this.form = result.form;
} else {
console.log('INVALID STATUS RECEIVED: [' + result.status + ']');
}
},
err => {
console.log('ERROR RECEIVED');
},
() => {
console.log('REQUEST COMPLETED');
}
);
}
}
```

Op de Observable die wordt opgeleverd door de service, kan een *subscribe()* worden gedaan. De webservice wordt dan asynchroon aangeroepen. Aan de *subscribe()* methode kunnen parameters worden meegegeven om:

- een succesvolle response van de webservice te verwerken. In het bovenstaande voorbeeld wordt gekeken of de *status* 'SUCCESS' wordt teruggegeven (of er geen functionele error is opgetreden). Daarna zal het *form* element worden opgeslagen, zodat dit gebruikt kan worden door de user interface-componenten. Zowel *status* als *form* zijn child elementen van de opgeleverde JSON structuur.
- een foutieve response (technische error) van de webservice te verwerken.
- Angular te notificeren wanneer het request volledig is uitgevoerd.

Custom user interface componenten

Wanneer via de Angular CLI componenten worden aangemaakt, krijgen deze componenten ook direct een *selector* toegekend. Door middel van deze *selector* kunnen de componenten gerendered worden in de applicatie en/of andere componenten. Wanneer het component bijvoorbeeld de *selector* 'if-page' krijgt, kan deze getoond worden door de volgende HTML code:

```
<if-page></if-page>
```

Daarnaast is het ook mogelijk om via de *@Input()* annotatie, parameters door te geven aan het component. Wanneer een pagina in het formulier moet worden getoond, zijn we bijvoorbeeld alleen geïnteresseerd in de gegevens over deze pagina, en niet over de overige gegevens uit de webservice. In het onderstaande voorbeeld is de page parameter toegevoegd. Deze kan meegegeven worden door de onderstaande HTML code.

```
<if-page [page]=het-gewenste-page-object></if-page>
```

Onderstaand een voorbeeld van de implementatie van het *(if-)*page component:

```
import { Component, OnInit, Input } from ‘@angular/core’;
import { Page } from ‘../../get-form-response/page’; // Het deel van de
JSON structuur/JavaScript objecten dat we nodig hebben voor dit specifieke
component.
@Component({
selector: ‘if-page’, // De selector
templateUrl: ‘./page.component.html’,
styleUrls: [‘./page.component.css’]
})
export class PageComponent implements OnInit {
@Input() page: Page; // De parameter die moet worden meegegeven aan het
component.
constructor() { }
ngOnInit() { }
}
```

In het root component van de applicatie (waarin onder andere de service wordt aangeroepen, en wat dus ook het startpunt is van het renderen van het formulier), kunnen de pagina’s vervolgens op de volgende manier worden toegevoegd aan de user interface:

```
<div class=”header”>
<h1>{{form.title}}</h1>
<p class=”subTitle”>{{form.subject}}</p>
</div>
<if-page *ngFor=”let page of form.page” [page]=”page”></if-page>
<div class=”footer”>
<sub>Assigned to <i>{{form.assignee}}</i></sub>
</div>
```

In het eerste en laatste blok wordt wat generieke informatie over het formulier getoond. Het middelste stuk is het gedeelte waar voor elke pagina, die uit de service terugkwam, een *if-page* component wordt toegevoegd, en waar ook direct de pagina wordt geïnjecteerd bij het aanmaken van het component.

Hierna is het eigenlijk een herhaling van zetten. Waar de application root de *if-page* componenten moet tonen, moeten de *if-page* componenten op hun beurt *if-item* componenten tonen. Deze tonen vervolgens weer *if-question* componenten, etc.

Dit samen leidt dan bijvoorbeeld tot het onderstaande scherm. In dit scherm wordt een formulier getoond dat 2 pagina's bevat. De inhoud van deze pagina's bestaat dan weer uit verschillende component typen, zoals open invoervelden, checkboxes en bestanden.

Voorbeeld formulier met UI-componenten

Door de manier van aanmaken van deze componenten, kunnen deze componenten zonder veel moeite oneindig genest worden. Het zou dan bijvoorbeeld ook mogelijk zijn om een pagina in het formulier uit meerdere (sub)pagina's/groepen te laten bestaan.

Conclusie

Angular lijkt voor het dynamisch opbouwen van de user interface een ontzettend krachtig framework te zijn. Ontwikkeling van de applicatie gaat snel door de Angular CLI. Het is eenvoudig om JSON over REST services aan te roepen en de JSON vervolgens om te zetten naar TypeScript objecten. Ten slotte werkt het aanmaken van eigen user interface-componenten erg goed. Wanneer een component eenmaal is aangemaakt, kan het vanuit elk (sub)component van de applicatie worden aangeroepen en getoond. Dit samen maakt Angular een erg krachtige oplossing voor het tonen van dynamische user interfaces.