Inhoud

1 Kennismaken met AngularJS 1

Wat is AngularJS? **2**
Libraries en frameworks 2
Omschrijving van AngularJS 3
AngularJS op internet 5
Versies van AngularJS **6**
AngularJS in de pagina 7
AngularJS-concepten **8**
Client-sided HTML templates 8
Data binding 9
Dependency injection 10
Directives 12
Custom directives 13
Benodigde voorkennis **14**
Tips voor meer lezen 15
De ontwikkelomgeving inrichten **15**
Editor en browser 15
Een webserver 16
Andere tools 18
Debugging 20
Oefenbestanden downloaden **22**
Conclusie **22**
Praktijkoefeningen **23**

2 Beginnen met AngularJS 27

Een AngularJS-paginasjabloon instellen 28
Directive ng-app 29
Notatiewijzen voor directives 30
Hello world 31
ng-model 31
Meer over ng-model 32
ng-true-value en ng-false-value 33
De directive ng-repeat 35
Notatie van ng-repeat 37
Ng-repeat bij objecten 37
Gegevens filteren 39
uppercase en lowercase 39
orderBy 40
Omgekeerd sorteren 41
Gegevens filteren 42
Filteren op veldnaam van object 43
Resultaat niet gevonden – ng-if 44
Overige filters 45
Conclusie 45
Praktijkoefeningen 46

3 Modules en controllers 49

Wat is een module? 50
Module als setter en als getter 52
Dependencies 53
Wat is een controller? 55
De directive ng-controller 56
Wat is $scope? 56
Een controller definiëren 57
De code refactoren 59
Functies schrijven in de controller en de directive ng-click 61
Ng-click binnen ng-repeat 62
Meer kenmerken van controllers 64
Module en controller in aparte bestanden 65
Module in app.js 66
Controller in controller.js 66
Andere syntaxis om controllers te definiëren 67
Minification safe syntaxis 68
Benoemde functies 69
Werken met $inject 70

De controllerAs-syntaxis **71**
Weg met $scope 71
Var vm als viewmodel 71
Conclusie **74**
Praktijkoefeningen **75**

4 Routing in Angular-apps 79

Kennismaken met routing **80**
Single page Application of SPA 80
Routing alleen via http-server 82
Routing installeren **84**
Ng-route en ui-router 84
Module ng-route downloaden 84
Dependency aangeven 85
De routing-directive ng-view 85
Werken met $routeProvider **86**
Parameters voor .when() 87
Een pad configureren 87
.otherwise() instellen 88
Controllers maken 89
Views maken 89
Inline HTML in template 90
Samenvatting routing 91
Werken met routeparameters **91**
Het id doorgeven 92
Case: master/detailview met persoonsgegevens **93**
De views voor de app 93
Directive ng-href 95
De controllers voor de app 95
Het bestand app.js 97
Herhaling van array met persoonsgegevens? 97
Conclusie **98**
Praktijkoefeningen **99**

5 Gegevensvoorziening met factories en services 101

Waarom services en factories? **102**
Doelen van services en factories 102
Singleton 103
Dataflow in de applicatie 104
Factory of service? 105
Een factory maken **106**
De controller aanpassen 108

Detailgegevens ophalen via de factory 109
De view aanpassen 109
De detailview schrijven 109
De detailController schrijven 110
Gegevens toevoegen via de factory 111
Nieuwe view toevoegen 111
Nieuwe controller schrijven 112
Methode .addPerson() schrijven 114
Een service maken 115
Methodes rechtstreeks op de functie schrijven 115
De service injecteren 116
Meer datacomponenten: value en constant 117
Hoofdlettergebruik 118
Constanten gebruiken in een controller en view 119
Een value gebruiken 120
Conclusie 122
Praktijkoefeningen 123

6 Live data ophalen met $http 125

Wat doet $http? 126
Algemene werking van $http 126
$http retourneert geen data 128
Json-bestand laden van server 128
Persons.json 129
Userinterface maken 129
Module, controller en factory maken 130
Controller implementeren 130
Factory implementeren 131
Fouten afvangen 132
View uitbreiden 132
Controller uitbreiden 132
De module ngSanitize 134
Live API's op internet gebruiken 135
Dummy persoonsgegevens ophalen 135
Formaat voor werken met filltext.com 137
De HTML-view voor persoonsgegevens 137
De controller maken 138
De factory maken 139

Ajax-chaining met .then() **140**
Opeenvolgende Ajax-calls 141
Structuur van .then() 142
Case: filmgegevens ophalen via OMDb API **143**
De HTML-code 143
De app instellen 144
De controller instellen 145
De movieService maken 146
Gegevens tonen 148
Case: gerelateerde films ophalen **148**
HTML-code uitbreiden 149
Constante aanpassen 149
Controller uitbreiden 150
De service uitbreiden 151
Verder gaan met $http, promises en $q **153**
Error handling in .then() 153
Promises en $q 153
Meer API's om mee te experimenteren **156**
Registreren voor API key 156
Open API's 156
Conclusie **157**
Praktijkoefeningen **158**

7 Meer standaarddirectives 161

Directives voor DOM-manipulatie **162**
ng-show en ng-hide 163
ng-if 164
CSS-klassen beheren met ng-class 165
ng-href en ng-src 169
Directives voor event handling **170**
Ng-click 170
De parameter $event 171
Ng-mousedown, ng-mouseup en andere 172
Ng-keypress 173
Ng-paste 175
Directives voor formulieren **176**
Ng-submit 176
Selectielijsten maken 178
Ng-change 179
For-in-notatie voor objecten in selectielijst 179
Ng-focus en ng-blur 181
Ng-checked 182

Overige directives **183**
Ng-cloak 183
Ng-copy, ng-cut en ng-paste 185
Ng-disabled 185
Conclusie **186**
Praktijkoefeningen **187**

8 Custom directives schrijven 189

Beginnen met custom directives **190**
Een klokje maken in de pagina 191
Naamgeving van directives in HTML en JavaScript 192
De klok functionaliteit geven 193
Het Directive Definition Object (DDO) **195**
Opdrachten in het DDO 195
Isolated scope creëren in een directive **197**
Bad practice 198
Scope isoleren via het DDO 199
Gegevens als attributen doorgeven 200
Controller van de directive aanpassen 202
Onafhankelijke module maken van de directive **203**
DOM-manipulatie directives **204**
Het DDO uitbreiden 205
Werken met jqLite 206
Waarden van attributen benaderen 207
Case: een jQuery-plug-in in een AngularJS-directive gebruiken **209**
Popover van Twitter Bootstrap en jQuery 209
HTML-code schrijven 210
Directive-module schrijven 210
De functie link voor de directive schrijven 211
De directive injecteren 212
HTML-code schrijven om de directive te gebruiken 212
Conclusie 213
Kant-en-klare custom directives vinden en gebruiken **214**
Verdergaan met AngularJS **215**
Conclusie **217**
Praktijkoefeningen **219**

1 Hoofdstuk

Kennismaken met AngularJS

Van enkele eenvoudige HTML-pagina's in het begin van de jaren negentig is het web uitgegroeid tot een van de meest complexe systemen die we kennen. Internet wordt gebruikt voor relatief eenvoudige hobbysites, maar ook voor online betaalsystemen, CRM- en klantbeheersystemen, verzekerings- en schademodellen, sociale media en ontelbare andere zaken. AngularJS biedt een framework voor het programmeren van dergelijke ingewikkelde webapps. Er zijn concepten in verwerkt die het mogelijk maken code en structuur van elkaar te scheiden, modulair te programmeren en bovendien goed testbare applicaties te maken. Dit boek geeft een inleiding op al deze zaken, zodat u na afloop vol vertrouwen aan de slag kunt met uw eigen AngularJS-applicaties.

In dit hoofdstuk:

Wat AngularJS is, en wat het niet is.

AngularJS-concepten.

Benodigde voorkennis en software.

Kenmerken van AngularJS-apps.

Wat is AngularJS?

Het aloude HTML is prima om gegevens gestructureerd te tonen in de browser, maar is oorspronkelijk nooit ontwikkeld voor het maken van dynamische webapplicaties. Voor dat doel is JavaScript rond 1995 ontworpen en samen met CSS (dat rond dezelfde tijd opkwam) toegevoegd aan de toolbox van de webdeveloper. JavaScript was in het begin lastig te leren en verschillende browsers hadden elk een verschillende implementatie van JavaScript.

Libraries en frameworks

Pas sinds de opkomst van aanvullende bibliotheken als jQuery in 2006 heeft JavaScript een enorme vlucht genomen. Behalve jQuery zijn tal van andere bibliotheken ontwikkeld, elk met hun eigen doel. Er zijn libraries voor DOM-manipulatie (zoals jQuery), routing (sammy.js), data binding (knockout.js) en nog veel meer.

Maar AngularJS is geen library zoals de hiervoor genoemde. AngularJS is een compleet *framework* voor het realiseren van client-sided webapplicaties. Als we de zaken erg vereenvoudigd voorstellen, kun je zeggen dat libraries in het algemeen één ding heel goed doen. Een framework zoals AngularJS biedt oplossingen voor alle niveaus van applicatieontwikkeling. Van het structureren en binden van data, tot Ajax-communicatie met webservers, het verwerken van geretourneerde gegevens in een client-sided datamodel en het maken van herbruikbare componenten.

Libraries kunnen in het algemeen worden gecombineerd in een project om gezamenlijk het beste resultaat te bereiken. Bij frameworks maak je daarentegen één keuze en bouw je de app volgens de richtlijnen en kenmerken van het gekozen framework.

Geen combinaties

We zullen in de praktijk bijvoorbeeld nooit zien dat een app *zowel* AngularJS als Durandal (een alternatief framework) gebruikt. Ook combinaties van AngularJS met Ember (een ander alternatief) komen niet voor. Je bouwt de site ofwel in AngularJS, ofwel in Durandal. Niet in beide.

Omschrijving van AngularJS

Het Angular-framework wordt op de officiële site omschreven als:

"AngularJS lets you write client-side web applications as if you had a smarter browser. It is a structural framework for creating dynamic web apps."

Met AngularJS is het relatief eenvoudig om complexe web-applicaties te schrijven, omdat het framework als het ware een abstractielaag biedt tussen de browser, de logica van de app en de data waarmee wordt gewerkt. Van oudsher wordt dit vaak aangeduid met de term *MVC*, van *Model-View-Controller*, maar dit wordt langzamerhand een beetje losgelaten. Als u toch deze vergelijking nog wilt maken:

- **Model** De data die de applicatie binnenkomt (meestal uit een database, via een Ajax-call) heeft een bepaalde structuur en wordt het 'model' genoemd.
- **Controller** De logica in de applicatie bewerkt data in het model. Het voegt bijvoorbeeld losse velden als `firstname` en `lastname` samen tot een veld `fullname` dat in de user interface wordt getoond. Deze logica staat in een component die 'controller' wordt genoemd.

- **View** De user interface bestaat uit HTML-templates waarin de – eventueel bewerkte – gegevens worden getoond. Het HTML-template is daarmee de 'view' van de applicatie. Een app kan natuurlijk vele views hebben. Van het tonen van de homepage tot master-detailviews.

AngularJS is daarmee een compleet client-sided MVC-framework. Het is volledig in JavaScript geschreven en draait ook volledig in de browser. Idealiter is de app compleet losgekoppeld van de server en database waar de gegevens vandaan komen. Alle communicatie vindt plaats via Ajax-calls. Uiteraard gaan we hier later in dit boek nog dieper op in.

Geen 'MVC' meer

Het begrip MVC om de architectuur voor AngularJS aan te duiden wordt langzamerhand minder gebruikt. De terminologie van MVC zou te strikt zijn en niet goed passen bij de flexibiliteit van AngularJS. Als u eerder hebt gewerkt met Microsoft-technologieën als Silverlight of XAML, kent u misschien het begrip *MVVM* (*Model-View-ViewModel*). Ook dit is te vertalen naar een AngularJS-structuur. Andere design patterns zijn bijvoorbeeld *Model-View-Adapter* en *Model-View-Presenter*. Om die reden wordt AngularJS nu ook wel benoemd als een *MV**-framework (*Model-View-Whatever*). De begrippen controller en view zullen we echter zeker nog tegenkomen bij het maken van AngularJS-apps.

Afbeelding 1.1 *De homepage van AngularJS. Start hier voor officiële downloads, documentatie en meer.*

AngularJS op internet

De homepage van AngularJS is te vinden op **angularjs.org**. Hier kunt u het framework downloaden, onlinetutorials volgen, deelnemen aan discussies, video's bekijken van de diverse AngularJS-conferenties en meer. Ook is dit het startpunt voor de officiële documentatie. Kies hiervoor de optie **Develop**, **API Reference** uit het hoofdmenu, of voeg **docs.angularjs.org/api** direct toe aan uw favorieten.

Wilt u helemaal hardcore gaan, dan kunt u een eigen versie van AngularJS bouwen via de Github-pagina. Ga naar **github.com/angular/angular.js** om de broncode te downloaden, te builden en eventueel aan te passen voor eigen gebruik. In dit boek maken we hiervan geen gebruik.

Versies van AngularJS

AngularJS is rond 2009 ontstaan als intern project bij Google. Misko Hevery (@mhevery op Twitter) is de 'vader van Angular'. Samen met projectmanager Brad Green (@bradlygreen) bouwde hij AngularJS uit tot volwaardig framework dat ook door anderen kon worden gebruikt. Rond 2011 gaf Google het framework onder de MIT-licensie vrij als open source-software. In de loop der jaren zijn verschillende versienummers gebruikt.

- **0.9 - 1.1** Oktober 2010 – juni 2012. De eerste versies die voor het algemene publiek beschikbaar waren. Alle functionaliteit was gebundeld in één AngularJS-module.
- **1.2** November 2013. Verbeterde versie met onder meer uitsplitsing van deelfunctionaliteit (routing, animatie) naar losse modules.
- **1.3** Oktober 2014. Talloze bugfixes, performanceverbeteringen en enkele nieuwe opties.
- **2.0** Eind 2015? Op het moment van schrijven van dit boek wordt AngularJS 2.0 ontwikkeld. Hierover is heftige discussie gaande op internet. Angular 2.0 wordt een compleet nieuw framework en gaat talloze *breaking changes* bevatten, waardoor het upgraden van 1.3 naar 2.0 een vrijwel onmogelijke zaak wordt. AngularJS zal compleet op ES6 (de volgende versie van JavaScript) zijn gebaseerd.

Het is nog onduidelijk hoe het pad naar Angular 2.0 verder zal verlopen. In ieder geval is dat geen reden om nu niet alvast kennis te maken met dit 'superheroic framework', zoals Angular zichzelf enigszins spottend omschrijft. De concepten uit Angular 1.3 (modules, directives, views, dependency injection) zullen gelijk zijn, de implementatie wordt in 2.0 echter compleet anders, zo is de verwachting.

In dit boek: AngularJS 1.3

In dit boek gebruiken we AngularJS versie 1.3.3 uit november 2014. Dit was op het moment van schrijven de meest recente stabiele versie.

AngularJS in de pagina

Het maken van een AngularJS-applicatie is eenvoudig. Zorg ervoor dat het script is gedownload en in een map van uw website staat (wij gebruiken de map \js\vendor\angular). Daarna kunt u het met een eenvoudige JavaScript-referentie insluiten in de pagina:

```
<script src="../js/vendor/angular/angular.js"></script>
```

Wilt u helemaal niks downloaden maar rechtstreeks online ontwikkelen, dan kunt u ook verwijzen naar AngularJS op het Google CDN. Een eigenschap is dan uiteraard wel dat u alleen kunt ontwikkelen als u online bent:

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.3/angular.min.js"></script>
```

Welke methode u ook gebruikt, na het insluiten is het framework in de pagina beschikbaar en kunt u elke Angular-opdracht uitvoeren die u in dit boek tegenkomt.

AngularJS-concepten

Er is een aantal kernbegrippen waar u in elke Angular-app mee te maken krijgt. Deze concepten zijn geen van alle uitgevonden door het Angular-team zelf, maar geleend uit de andere ontwikkelomgevingen en daarna met een JavaScript-implementatie toegepast in het framework. In de loop van het boek komen al deze concepten uiteraard aan de orde. Hier noemen we alvast kort de belangrijkste.

Client-sided HTML templates

Hebt u ooit in jQuery moeizaam een HTML-fragment samengesteld als `$('#placeholder').append('<span class="tip"> + data.tiptekst + '</span>')` en dit dan vaak nog een stuk complexer? Dan zult u de functionaliteit van HTML-templates in AngularJS kunnen waarderen. In Angular zijn de data en templates waarop die data worden toegepast, van elkaar gescheiden. De templates kunt u met elke editor zelf schrijven (zoals we in dit boek doen), of op de server laten samenstellen door bijvoorbeeld PHP, ASP.NET MVC of Java Spring en dan naar de browser zenden. De data komt typisch via een Ajax-call binnen en wordt met de *double curly braces* gebonden aan de template.

Een vergelijkbare code als het jQuery-voorbeeld hiervoor ziet er als Angular HTML-template bijvoorbeeld als volgt uit:

```
<span class="tip"> {{ data.tipTekst }} </span>
```

Hierbij is `data` een JavaScript-object (het model), dat een eigenschap `tipTekst` heeft. Dit wordt binnen `{{ … }}` gebonden aan de HTML-template (de view). Een meer uitgebreide template ziet er bijvoorbeeld uit als:

```
<p class="bookDetails">{{book.description }} </p>
<footer>
    <p class="bookTitle">{{ book.title }} </p>
    <p class="bookAuthor">{{book.firstAuthorName}}</p>
</footer>
<img ng-src="{{ imgSrc }}" alt="pagina van {{ book.title }}} " />
```

Data binding

In de HTML-template worden de dubbele accolades gebruikt om gegevens te binden. Alles wat tussen {{ … }} staat, wordt door het framework opgepikt en geëvalueerd. De uitkomst ervan wordt in de user interface geplaatst. Stel dat het object `data` uit de vorige paragraaf de volgende inhoud heeft:

```
var data = {
    tipTekst : 'Klik met de rechtermuisknop voor extra opties'
}
```

En de HTML ziet eruit als:

```
<span class="tip"> {{ data.tipTekst }} </span>
```

Dan is de uitvoer in de browser (wat de bezoeker ziet):

```
<span class="tip">Klik met de rechtermuisknop voor extra opties</span>
```

Het wordt natuurlijk extra krachtig als de data binding-expressies in een lus worden geplaatst, of in twee richtingen werken. Dit wordt in Angular *two way data binding* genoemd. Het betekent dat wanneer gegevens door de bezoekers worden ingevuld of gewijzigd (bijvoorbeeld in een HTML-formulier) deze wijzigingen direct worden doorgegeven in het model. U beschikt in JavaScript dan direct over een model waarin de gegevens aanwezig zijn die de bezoeker heeft ingevuld. Het is niet nodig (opnieuw zoals in jQuery) eerst het gewenste formulierveld te selecteren en daarvan de huidige waarde uit te lezen. In het volgende hoofdstuk maakt u tot in detail kennis met data binding.

Dependency injection

In AngularJS-applicaties kan functionaliteit worden gebundeld in een module. Deze module kan vervolgens worden hergebruikt (*geïnjecteerd*) in andere modules. Dit principe staat bekend als *dependency injection*, of *DI*.

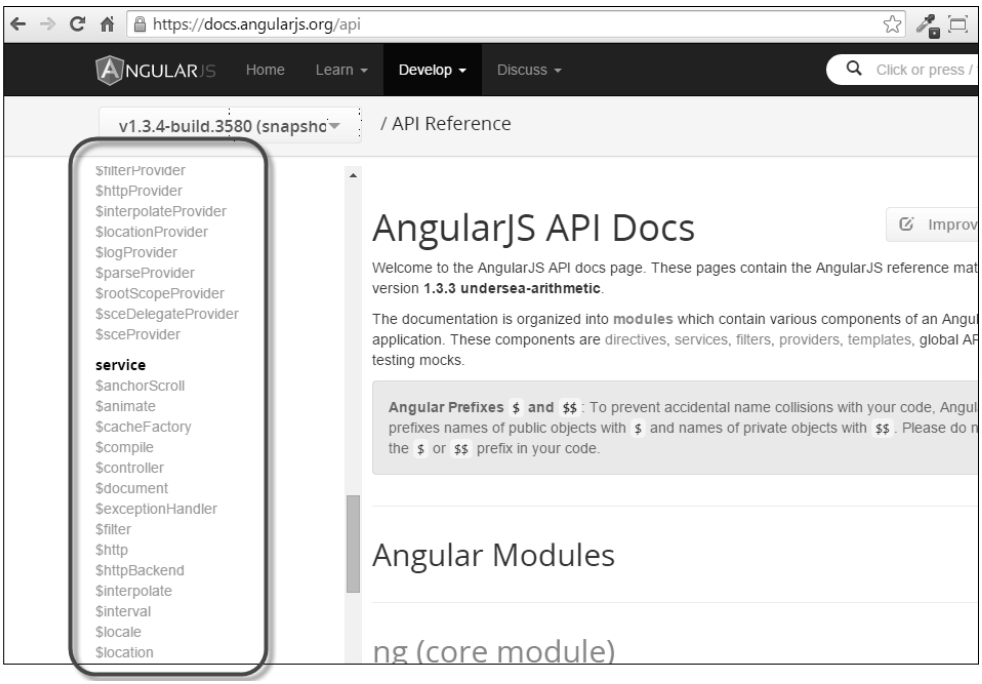
Dit betekent dat code erg flexibel wordt. Stel dat we een complete unit hebben gemaakt met tips voor een applicatie en we willen diezelfde tips gebruiken in een andere applicatie. Dan kunnen we de tips-module injecteren in de nieuwe module en hoeven we geen code te dupliceren. Het spreekt voor zich dat het in de meeste gevallen dan niet zal gaan om een eenvoudig object met strings, maar dat er complete functionaliteit, inclusief functies en server calls kunnen worden gebundeld.

Het systeem van dependency injection is erg krachtig. Vaak worden afhankelijkheden als parameter meegegeven aan een functie of module. Angular zoekt dan zelf de betreffende modules op en voegt ze in. Stel bijvoorbeeld dat we een controller schrijven die gebruik wil maken van de basis AngularJS-services `$http` (voor Ajax-calls) en `$location` (voor werken met de adresregel in de browser). Dan ziet een functiedefinitie er bijvoorbeeld als volgt uit:

```
function tipController($scope, $http, $location) {
    // code voor de controller...
}
```

Standaard AngularJS-services

Bij DI kunt u zowel eigen services en modules injecteren als standaard Angular-services. De standaardservices herkent u aan het dollarteken, zoals `$scope`, `$http` en `$location`. Ze worden standaard met AngularJS meegeleverd en u hoeft ze niet apart in te voegen. Het dollarteken geeft niks speciaals aan, het is gewoon een deel van de naam. Zo zijn ze makkelijk herkenbaar.



Afbeelding 1.2 *Standaardservices van AngularJS zijn te herkennen aan het dollarteken voor de naam. In de documentatie worden ze beschreven.*