Inhoud

1 Hoofdstuk

Kennismaken met Angular 2

Van enkele eenvoudige HTML-pagina's in de jaren 1990 is het web uitgegroeid tot een van de meest complexe systemen die we kennen. Internet wordt gebruikt voor relatief eenvoudige hobbysites, maar ook voor onlinebetaalsystemen, crm- en klantbeheersystemen, verzekerings- en schademodellen, sociale media en ontelbare andere zaken. Angular 2 is een framework voor het programmeren van dergelijke ingewikkelde webapps. In Angular 2 zijn MVC-concepten verwerkt die het mogelijk maken code en structuur van elkaar te scheiden en – meer nog dan met voorganger AngularJS – modulair te programmeren. Dit boek geeft een inleiding op al deze zaken. Na afloop kunt u vol vertrouwen aan de slag met eigen Angular 2-applicaties.

In dit hoofdstuk:

Wat Angular 2 is, en wat het niet is.

Angular 2 versus AngularJS.

Concepten en kenmerken van Angular 2.

Benodigde voorkennis en software.

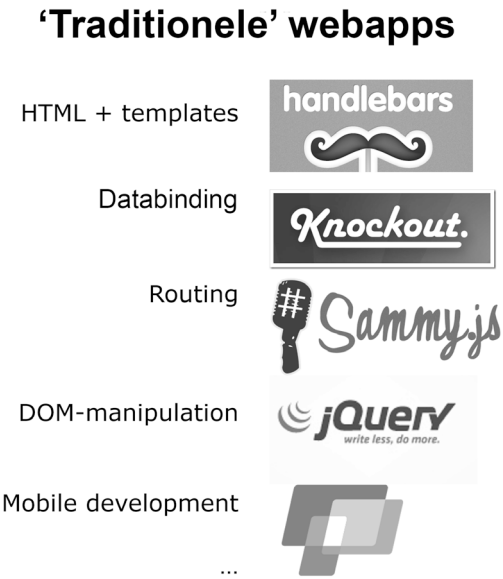
Wat hebt u nodig? De werkomgeving inrichten.

Wat is Angular 2?

Het aloude HTML is prima om eenvoudige tekst en afbeeldingen te tonen in de browser, maar is oorspronkelijk nooit ontwikkeld voor het maken van dynamische webapplicaties. Voor dat doel is JavaScript rond 1995 ontworpen. Samen met CSS (dat rond dezelfde tijd opkwam) behoort JavaScript op dit moment tot de basisvaardigheden van elke webdeveloper. JavaScript was in het begin lastig te leren en verschillende browsers hadden hun eigen ideeën over de 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

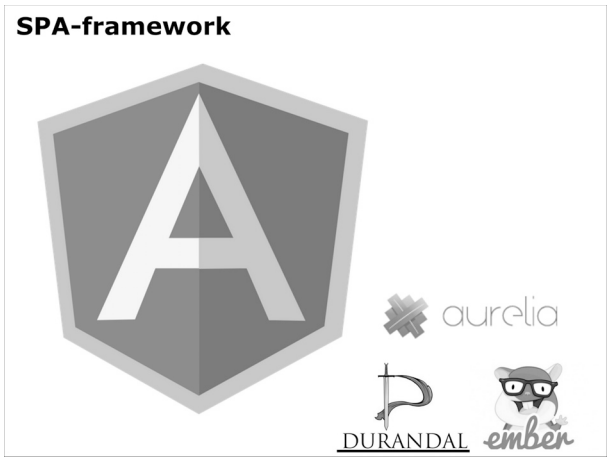


Afbeelding 1.1 *In traditionele webapplicaties neemt elke library één van de eisen die aan de applicatie wordt gesteld voor zijn rekening. Er is kans op onderlinge incompatibiliteit.*

eigen doel. Er zijn libraries voor DOM-manipulatie (zoals jQuery), routing (sammy.js), databinding (backbone, knockout.js) en nog veel meer.

Maar Angular is geen library zoals de hiervoor genoemde. Angular 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 Angular, 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 gecombineerd worden in een project om gezamenlijk het beste resultaat te bereiken. Bij frameworks daarentegen wordt één keuze gemaakt en wordt de app gebouwd volgens de richtlijnen en kenmerken van het gekozen framework.



Afbeelding 1.2 *In een framework als Angular, maar ook in alternatieven voor Angular zoals Aurelia of Ember, worden alle taken van een applicatie gebundeld en geïntegreerd aangeboden. De leercurve is steiler, maar het resultaat is een consistentere en eenvoudiger (en dus goedkoper te onderhouden) applicatie.*

Geen combinaties

We zullen in de praktijk bijvoorbeeld nooit zien dat een app zowel Angular 2 als Aurelia (een alternatief framework) gebruikt. Ook combinaties van Angular met Ember (een ander alternatief) komen niet voor. U bouwt de site ofwel in Angular, ofwel in Aurelia. Niet in beide.

Omschrijving van Angular 2

Het Angular-framework wordt op de officiële site omschreven als

"One framework. Mobile and desktop."

Dat geeft het doel voldoende aan, dachten wij zo. Met Angular is het relatief eenvoudig om complexe webapplicaties 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*, voor *Model-View-Controller*, maar dit wordt langzamerhand een beetje losgelaten. Als u toch deze vergelijking nog wilt maken:

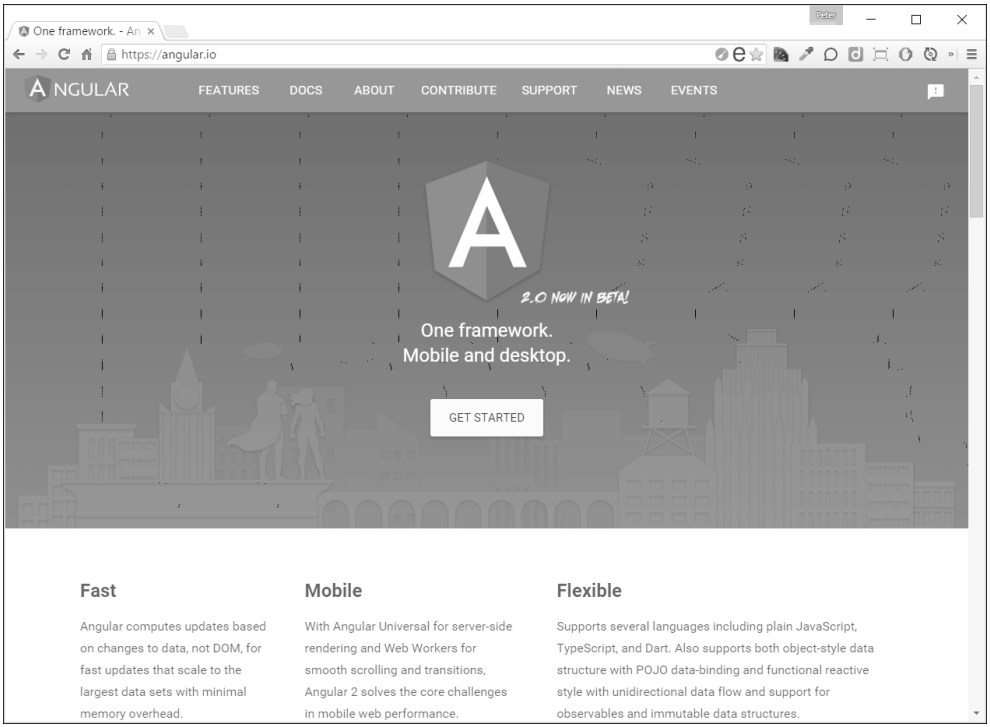
- **Model** De data die de applicatie binnenkomt (meestal uit een database, via een Ajax-call) heeft een bepaalde structuur en wordt het *model* genoemd.
  - In Angular 2 maakt u JavaScript-classes die het Model representeren.
- **Controller** De logica in de applicatie bewerkt data in het model. Er worden bijvoorbeeld losse velden zoals `firstname` en `lastname` samengevoegd tot een veld `fullname` dat in de gebruikersinterface wordt getoond.
  - In Angular 2 is ook de *controller* een JavaScript-klasse. De controller hoort altijd bij een bepaalde component.

- **View** De gebruikersinterface bestaat uit HTML-templates waarin de – eventueel bewerkte – gegevens worden getoond. De HTML-template is daarmee de *view* van de applicatie.
  - Ook nu weer behoort de view van een component tot een bepaalde JavaScript-klasse in een Angular 2-applicatie. Er zijn – met uitzondering van index.html – geen 'losse' HTML-pagina's meer in de applicatie. Alles is gebonden aan een component.

Angular 2 is daarmee, net als zijn voorganger AngularJS, 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 Angular. 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 Angular-structuur. Andere ontwerppatronen (*design patterns*) zijn bijvoorbeeld *Model-View-Adapter* en *Model-View-Presenter*. Om die reden wordt Angular ook wel aangeduid als een *MV**-framework (*Model-View-Whatever*). De begrippen controller en view zullen we echter zeker nog tegenkomen bij het maken van Angular-apps.



Afbeelding 1.3 *De homepage van Angular op angular.io. Start hier voor officiële downloads, documentatie en meer.*

Angular 2 op internet

De homepage van Angular is te vinden op **angular.io**. Hier kunt u artikelen lezen, online tutorials volgen, deelnemen aan discussies, video's bekijken van de diverse Angular-conferenties en meer. Ook is dit het startpunt voor de officiële documentatie. Kies hiervoor de optie **Docs**, **Developer Guides** of **Docs**, **API Preview** uit het hoofdmenu.

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

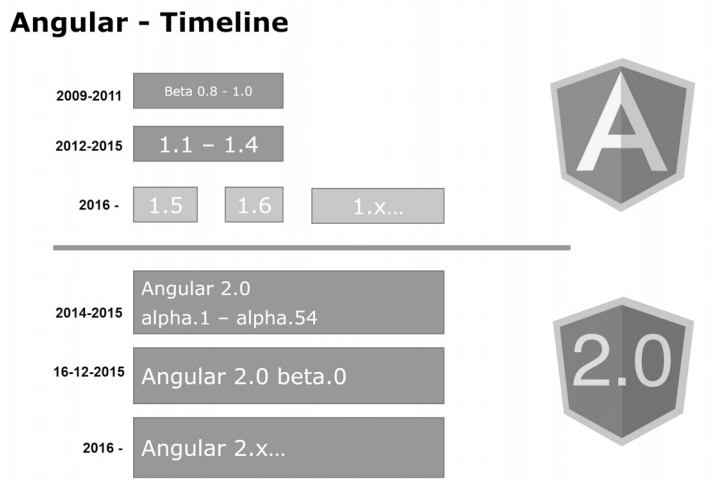
Versies van AngularJS en Angular 2

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 gebruikt kon worden. Rond 2011 gaf Google het framework onder de MIT-licensie vrij als opensourcesoftware.

Daarna volgde AngularJS min of meer het gebruikelijke upgradepad. Er werden nieuwe features toegevoegd, er waren bugfixes en prestatieverbeteringen, maar de globale werking van Angular 1.x was met elke nieuwe versie in ieder geval ongeveer gelijk. Het was relatief eenvoudig applicaties bij te werken naar de nieuwste versie.

Angular 2

Heel anders is dat met Angular 2. Er zijn eigenlijk precies zeven overeenkomsten tussen Angular 1 en Angular 2. Dat zijn de letters A, N, G, U, L, A, en R. Het is niet overdreven om te zeggen dat Angular 2 een compleet nieuw en ander framework is, met toevallig dezelfde naam. Daar zijn wel redenen voor (zie ook verderop), maar erg geliefd heeft team Angular zich met deze strategie niet gemaakt. Veel kennis uit AngularJS 1.x-applicaties kan in de prullenbak als u een Angular 2-toepassing wilt maken. Afhankelijk van de codeerstijl van een Angular 1.x-app zal het niet of slechts met een aanzienlijke tijdsinvestering (en dus kosten) mogelijk zijn een project te upgraden naar Angular 2.



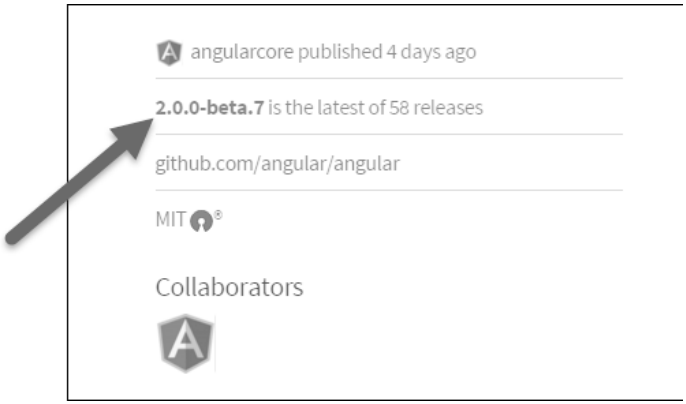
Afbeelding 1.4 *Versies van AngularJS (Angular 1.x) en Angular 2. Het upgraden van AngularJS-toepassingen naar Angular 2 is lastig. Daarom staat er een lijn tussen.*

In dit boek: Angular 2, beta 7

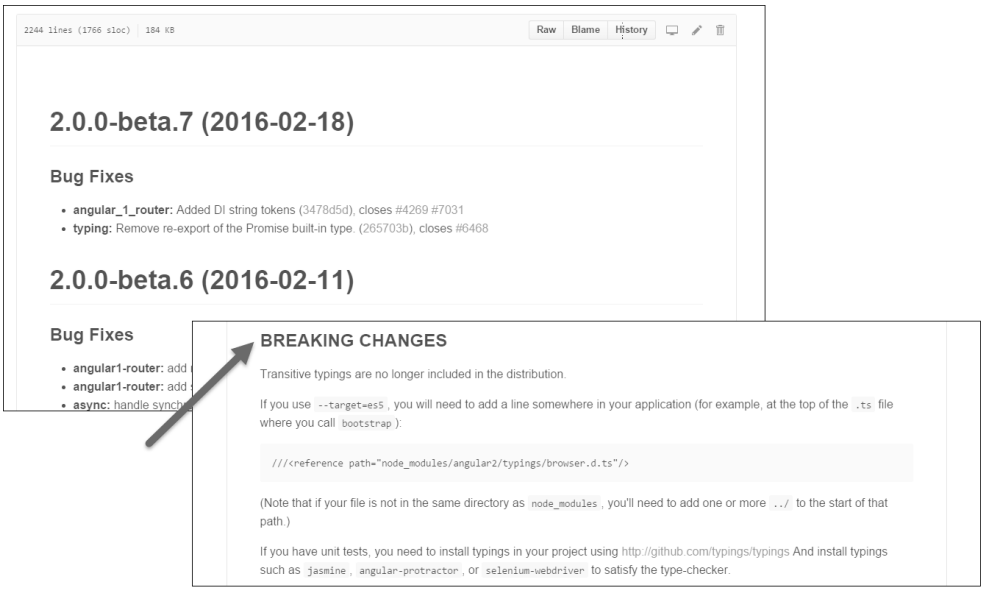
In dit boek gebruiken we Angular 2, versie 2.0.0.beta.7 uit februari 2016. Dit was op het moment van schrijven de meest recente stabiele versie. Hoewel Angular dus nog in ontwikkeling is (en hopelijk definitief is als u dit leest), is de API al 'bevoren'. Er zijn daarmee nog wel kleine implementatieverschillen mogelijk, maar de basiswerking van Angular 2 is vastgelegd en zal niet meer gewijzigd worden.

Changelog lezen

Hoewel het saai werk is, is het altijd een goed idee om de pagina changelog.md te lezen. Hierin wordt beschreven welke wijzigingen zijn doorgevoerd in Angular sinds de laatste versie en welke bugs zijn gerepareerd. Pak deze pagina er daarom bij als u zelf uw versies van Angular wilt updaten. Vergeet niet om in dat geval



Afbeelding 1.5 *Betaversies volgen elkaar in hoog tempo op. Op het moment van schrijven van dit boek was beta.7 de meest recente versie.*



Afbeelding 1.6 *Saai, maar noodzakelijk. Lees de wijzigingen en eventuele 'breaking changes'. Ze staan in changelog.md op github.com/angular/angular.*

ook de versienummers in package.json aan te passen (zie verder-op).

Direct aan de slag

Wilt u direct met een eerste Angular 2-app aan de slag? Sla dan de rest van het hoofdstuk voorlopig over en begin met de praktijk van hoofdstuk 2. Kom hier later nog eens terug als u meer wilt weten over enkele diepere achtergronden bij Angular 2.

AngularJS-concepten

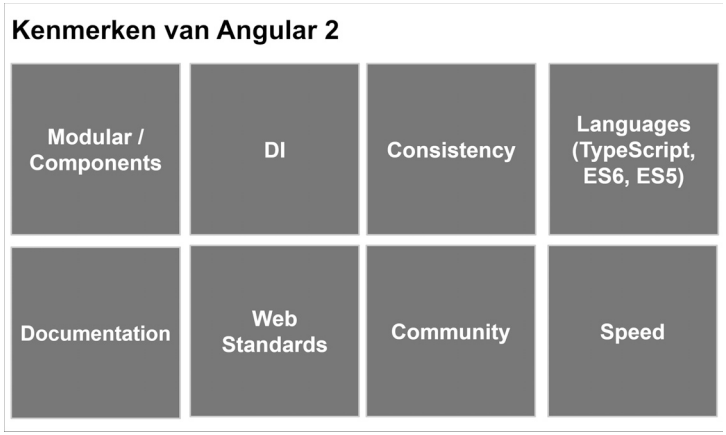
Er zijn een aantal kernbegrippen waar u in elke Angular-app mee te maken krijgt. Deze concepten zijn geen van alle uitgevonden door het Angular-team zelf, maar geleend uit de andere ontwikkelomgevingen en daarna met een JavaScript-implementatie toegepast in het framework. In de loop van het boek komen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

Waar mogelijk zullen we de verschillen met AngularJS 1.x aangeven. Het is overigens niet nodig dat u AngularJS kent voordat u met Angular 2 aan de slag gaat.

Modulair programmeren en componenten

Kernwoord in het maken van Angular-applicaties is het werken met *componenten.* Een Angular 2-app wordt volledig opgebouwd uit componenten. Een component kan bestaat uit meerdere onderdelen:

- TypeScript-componentannotatie die aangeeft hoe de component werkt (`@Component`).
- Een HTML-template met de gebruikersinterface (de *view*) van de component.



Afbeelding 1.7 *Enkele belangrijke concepten bij het maken van Angular 2-applicaties.*

- Een JavaScript-klasse met de logica van de component (de *controller*).
- JavaScript-statements `import` en `export` die aangeven welke afhankelijkheden (*dependencies*) de component heeft en welke onderdelen herbruikbaar zijn in andere componenten (geëxporteerd worden).

Nog meer dan AngularJS 1.x is een app in Angular 2 opgebouwd uit componenten. In AngularJS was er altijd één overkoepelende module die het startpunt van de applicatie was. Dit werd aangegeven met `angular.module(…)`. In Angular 2 bestaat dit concept niet meer. Elke component staat op zichzelf en kan eventueel als startpunt van een app worden gebruikt.

Componenten zijn *self contained modules* die alle logica en gebruikersinterface-informatie bevatten om de component te laten werken. In het volgende hoofdstuk maakt u direct uw eerste component en daarmee uw eerste Angular 2-app.

Java en .NET-achtergrond

Veel meer dan andere frameworks voor webdevelopment is Angular 2 een framework voor programmeurs. Vooral degenen met een Java of .NET-achtergrond kunnen hun hart ophalen. Eindelijk is het mogelijk om ook op het web met classes, constructors, getters en setters te werken. Voor hen is de overstap naar Angular 2 eenvoudiger dan bijvoorbeeld naar React, Aurelia of AngularJS 1.x. Voor degenen die gewend zijn aan traditioneel webdevelopment waarbij voornamelijk wordt gewerkt in HTML-pagina's en JavaScript, is Angular 2 behoorlijk wennen.

Dependency injection

Componenten kunnen afhankelijk zijn van andere componenten of van services die de component van gegevens (*data*) voorzien. Deze afhankelijkheden worden door Angular ingevoegd op het moment dat de component daar in de code om vraagt. Dit principe heet *dependency injection*. Het wordt vaak afgekort met DI, dat doen we ook in dit boek.

Anders dan in Angular 1 zijn componenten zelf verantwoordelijk voor het opvragen van afhankelijkheden. In Angular 1 moeten afhankelijke modules op het moment van instantiëren van de applicatie ingevoegd worden. U kent in dat geval code als:

```
angular.module('myApp', ['ngRoute', 'ngCharts', …]); // DI in Angular 1
```

In Angular 2 vindt DI niet plaats op applicatieniveau, maar op component-/moduleniveau. Met nieuwe keywords als `import` en in de `constructor()` van een klasse worden afhankelijkheden geïnjecteerd. De opdracht om bijvoorbeeld een `ProductService` te injecteren in een klasse en direct te instantiëren ziet er zo uit:

```
import {ProductService} from './services/product.service';
class myProducts{
    …
    constructor(private productService : ProductService){
        …
    }
}
```

Ook met DI gaat u in dit boek nog uitgebreid aan de slag en als het goed is denkt u er straks niet eens meer bij na.

Consistentie

AngularJS 1.x is het resultaat van een gestage, jarenlange ontwikkeling, waarbij telkens 'nieuwe ballen in de kerstboom werden gehangen'. Immers, de eerste versie van AngularJS stamt al uit 2009. Het was destijds een revolutionair framework en bood mogelijkheden die geen enkel ander framework had. Maar achteraf gezien zou het in sommige gevallen beter zijn geweest om eerst van tevoren wat langer na te denken over het implementeren van bepaalde concepten.

Wie heeft in AngularJS 1.x bijvoorbeeld niet geworsteld met het principe van `.factory()`, `.service()` en `.provider()`? Dit zijn concepten met een verschillende naam, maar ze doen allemaal – ongeveer – hetzelfde.

Of denk aan de filters uit AngularJS 1.x. Deze doen van alles, behalve data filteren! Ja, er is één filter, met de naam `filter` (het `filter`-filter) dat wél filtert. De rest formatteert alleen maar. Ook dat is een designbeslissing in het framework waarvan je achteraf zegt dat ze dat beter anders hadden kunnen doen.

In Angular 2 is dat gedaan. Het voordeel van Angular 2 is dat er van tevoren ruim anderhalf jaar is nagedacht over hoe het framework moet werken en wat de naamgeving van componenten moet zijn. Het framework is daarmee veel consistenter en eendui-

diger dan AngularJS 1.x ooit zal worden. Ook de werking van data-binding, attribuutbinding en eventbinding is veel consistenter dan in Angular 1.

Dat is voor AngularJS-veteranen even schrikken ('wat is er met mijn directives gebeurd!'), maar voor iemand die Angular 2 als nieuw framework moet leren is het uiteindelijk veel eenvoudiger geworden.

Programmeertalen

De voorkeursprogrammeertalen in Angular 2-apps zijn ECMAScript 2015 en TypeScript. ECMAScript 2015 (voorheen *ES6*) is de nieuwe versie van JavaScript, die ondersteuning biedt voor concepten als classes, arrow functions, block scope en meer. In dit boek zullen we ook TypeScript en ECMAScript 2015 gebruiken.

Afbeelding 1.8 *ECMAScript 2015 (voorheen ES6) en TypeScript zijn de voorkeursprogrammeertalen voor Angular 2-toepassingen.*

Tegelijkertijd kan echter ook nog worden gewerkt met 'ouderwets' JavaScript, of met Dart, een andere programmeertaal van Google. Hierin is Angular 2 flexibeler dan Angular 1.

Webstandaarden

Veel beter dan AngularJS is Angular 2 voorbereid op het gebruik van nieuwe webstandaarden. Ook dit heeft weer met de geschiedenis te maken. De basis van AngularJS stamt al uit 2009. Toen hadden we juist de iPhone 3GS in Nederland. Android 2.x (!) en het mobiele web stonden nog in de kinderschoenen. Nu (2016)

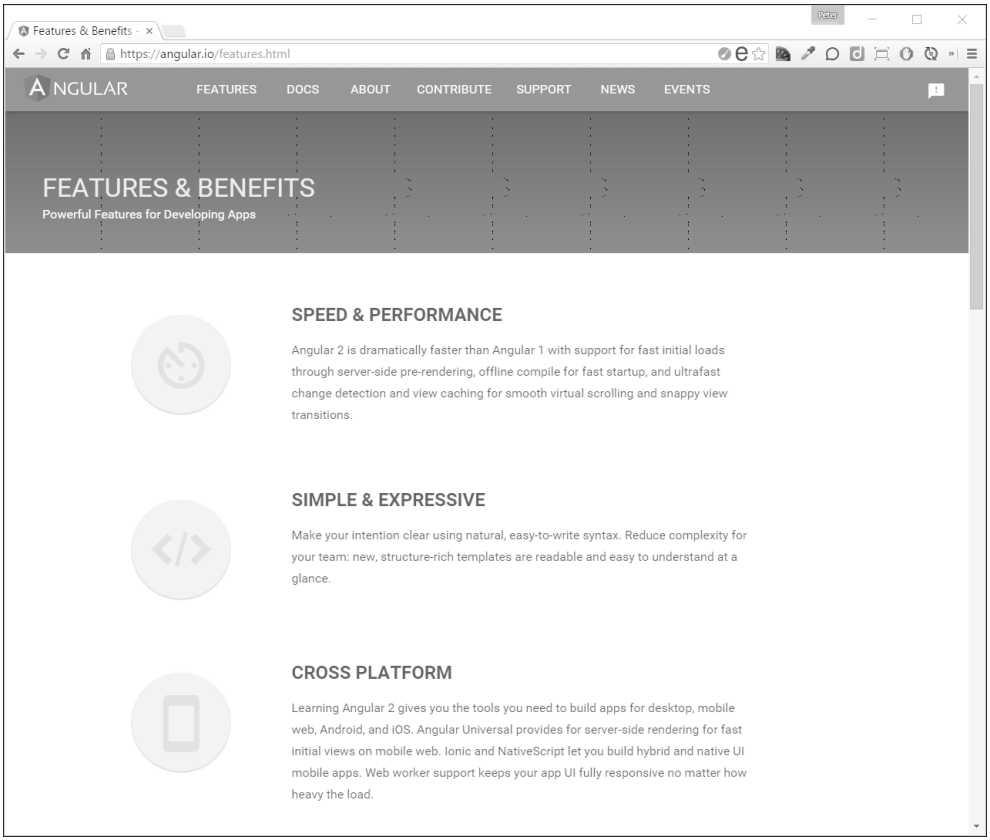
hebben we TypeScript, ECMAScript 2015, standaarden voor websockets, webworkers, serviceworkers, lokale opslag, nieuwe HTML5-API's en nog veel meer. Hier is Angular 2 veel beter op voorbereid.

Snelheid

Tot slot is snelheid een van de uitgangspunten van Angular 2 geweest. Team Angular streeft naar optimale snelheid op meerdere fronten:

- **Aan de gebruikerskant** Angular 2-apps voelen snel aan in gebruik, door het DOM alvast in het geheugen is aan te passen op basis van nieuwe data (shadow DOM) en door minimaal geheugengebruik en waar mogelijk native technieken te gebruiken op mobiele apparaten.
- **Aan de netwerkkant** Team Angular streeft ernaar de *footprint* van een minimale Angular 2 Hello World-applicatie (libraries plus uitvoerbare code) kleiner dan 30KB te maken, zodat de code snel geladen wordt en snel verwerkt kan worden door de pc, smartphone of tablet. Op dit moment, in de bètafase, is dat nog niet het geval, maar de code wordt nog geoptimaliseerd.
- **Aan de ontwikkelaarskant** Omdat Angular 2 zich veel consistenter gedraagt, kunnen ook de tools zoals compilers en editors daarvan profiteren. Door TypeScript te gebruiken kunt u bijvoorbeeld als ontwikkelaar *compile-time* al zien dat er nog fouten in staan, in plaats van af te wachten totdat een fout run-time optreedt. De tijd die u in de Chrome Developer Tools doorbrengt zal in Angular 2 hopelijk een stuk minder zijn dan in een vergelijkbare AngularJS 1-applicatie.

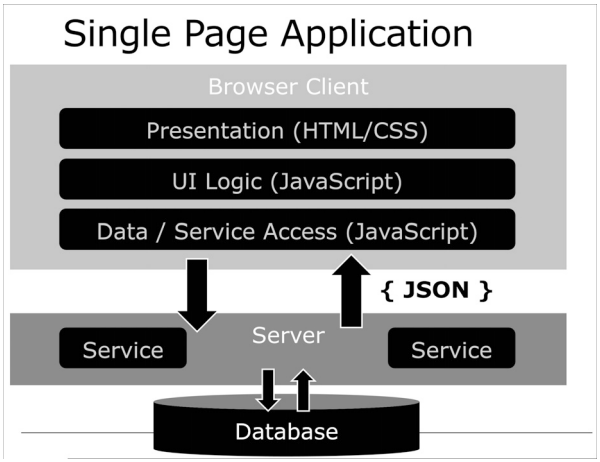
Meer kenmerken en eigenschappen van Angular 2 leest u bijvoorbeeld op de pagina Features & Benefits, te vinden op **angular.io/features.html**.



Afbeelding 1.9 *Lees meer over de Angular 2-features op internet.*

Architectuur van Angular 2-applicaties

In principe zijn Angular 2-applicaties toepassingen die volledig zelfstandig in de browser draaien. Er zijn ook varianten, zoals Angular 2 in een Node.js-applicatie op de server of in een stand-alone app die is gemaakt met Ionic of NativeScript. Daar gaan we in dit boek echter niet op in. We concentreren ons op webapps met een architectuur zoals in afbeelding 1.10 te zien is.



Afbeelding 1.10 *De architectuur die we nastreven in Angular-applicaties: de logica- en presentatielaag draaien in de browser, data-access en authenticatie draaien op de server.*

Single page application

De architectuur zoals in afbeelding 1.10 wordt getoond, wordt ook wel het principe van *single page applications* genoemd. Dit betekent dat in één pagina (index.html) de logica van de applicatie geladen wordt en dat deze pagina steeds zichtbaar is in de browser. Vanuit index.html vindt alle navigatie plaats.

De app zelf bestaat natuurlijk uit veel meer dan één pagina. Elke component staat in zijn eigen bestand, er zijn CSS-bestanden, afbeeldingen, enzovoort.

- De rol van de browser is:
  - *presentatielaag* tonen aan de gebruiker, via de view van componenten. Dit is gewoon HTML en CSS zoals u kent, verrijkt met Angular-specifieke toevoegingen;
  - *gebruikersinterfacelogica*, via de controller (class) van een view. Hierin staat logica om de gebruikersinterface samen te stellen, events zoals muiskliks te verwerken of het verwerken van Ajax-calls en meer;

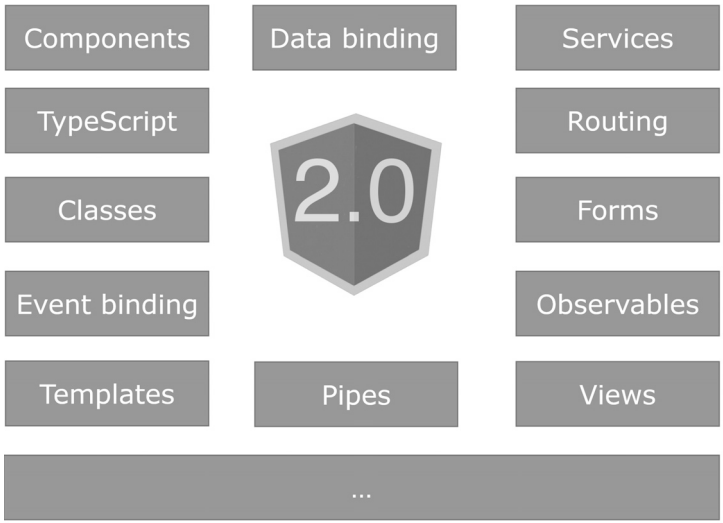
    - *data- en service-access,* oftewel communiceren met een of meerdere RESTful API's op de server. De API praat vervolgens met de database en retourneert gegevens, het liefst in JSON-formaat.

- De rol van de server is:
    - *database-toegang* via een API. Oneerbiedig gezegd wordt de rol van de server in moderne SPA-applicaties steeds verder teruggedrongen. Een server is niets meer dan een JSON-fabriek die data serveert;
    - *authenticatie*. Op het terrein van toegangsrechten is de server nog steeds onontbeerlijk. Immers; de complete Angular-app draait in de browser en is daarmee onveilig. Authenticatie en autorisatie zijn nog steeds het domein van de server. Hier moet worden ingelogd met gebruikersnaam en wachtwoord, via sociale netwerken of anderszins. De server moet vervolgens een token of authenticatiecookie uitreiken (afhankelijk van de wijze van beveiliging die door de serverbeheerder is gekozen). De Angular-app is er verantwoordelijk voor dat dit token of cookie met elke volgende request wordt meegezonden.

In dit boek gaan we niet verder in op het inrichten of beheren van een webserver. Vanaf hoofdstuk 6 gaat u wel aan de slag met het communiceren met (bestaande) databases vanuit Angular 2.

Angular 2-begrippen

Om de architectuur uit afbeelding 1.10 te realiseren, moeten algemene termen als *presentation*, *ui logic* en *data/service-access* natuurlijk concreet worden gemaakt. Dit gebeurt in Angular 2-apps met begrippen als observables, routing, services en meer. U ziet ze in afbeelding 1.11.



Afbeelding 1.11 *De algemene architectuur uit de vorige afbeelding wordt in Angular 2 geconcretiseerd via dit soort concepten.*

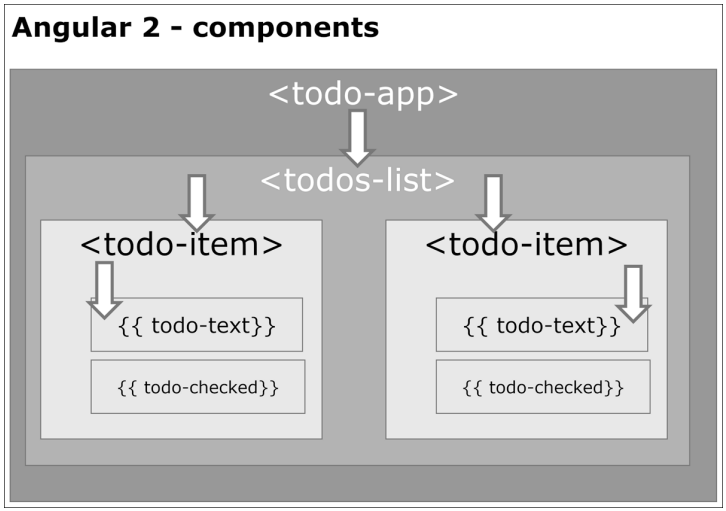
In de loop van dit boek zullen we steeds meer van de blokjes uit afbeelding 1.11 gaan invullen. We beginnen straks in hoofdstuk 2 met het concept componenten, en daarna komen ook databinding, services en alle andere onderdelen aan de orde.

Applicatie als boomstructuur van componenten

We hebben de term al een aantal keer genoemd. In de Angular 2-wereld staat het begrip *component* centraal. Elke applicatie heeft exact één hoofdcomponent (de *root component*).

- Binnen die hoofdcomponent worden deelcomponenten geladen.
- Elke deelcomponent heeft een eigen verantwoordelijkheid. Ze hebben een eigen gebruikersinterface en eigen logica.
- Deelcomponenten kunnen op hun beurt weer andere componenten bevatten. Deze worden met HTML-tags ingesloten in de template van de bovenliggende component.

- Het maken van een Angular 2-applicatie bestaat dus vaak uit het bouwen van erg veel kleine, zelfstandige componenten die met elkaar kunnen samenwerken. U bouwt aan een blokkendoos van componenten die onderling logische samenhang hebben en de app vormen.



Afbeelding 1.12 *Een Angular 2-app is een boomstructuur van componenten.*

Schematisch kan dit er bijvoorbeeld uitzien zoals in de afbeelding. Hierin is een Todo-applicatie voorgesteld.

- De applicatie wordt in index.html geladen met de selector `<todo-app>`. Hoe u zo'n selector maakt, leert u in het volgende hoofdstuk.
- De Todo-applicatie bestaat op zijn beurt weer uit een lijst. Deze wordt ingesloten via `<todo-list>`.
- Elke lijst bestaat weer uit afzonderlijke Todo-items (aangegeven met `<todo-item>`).
- Elke Todo-item kan weer bestaan uit velden met `todo-text`, de waarde `todo-checked` die aangeeft of het item is afgehandeld, eventueel knoppen, andere gebruikersinterface-elementen enzovoort.

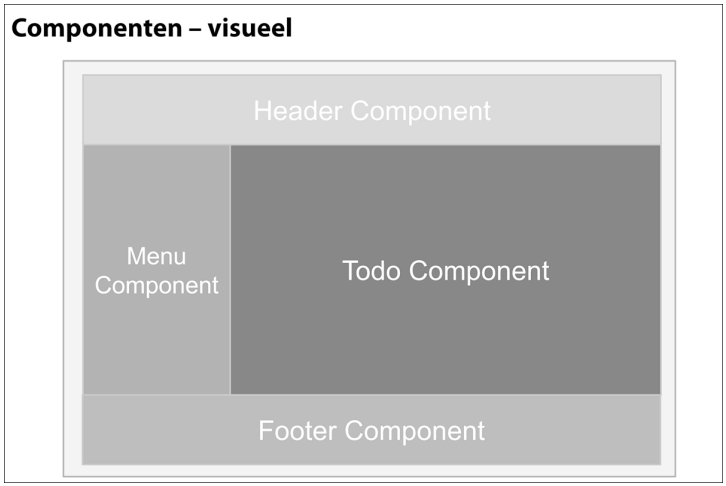
Op deze manier worden Angular 2-applicaties opgebouwd. Onthoud daarom altijd de volgende uitspraak:

"Elke Angular 2-applicatie bestaat uit een boomstructuur van componenten."

De boomstructuur visueel maken

Er is dus altijd één hoofdcomponent (*root*). Niet meer, niet minder. Hoe die boomstructuur vervolgens visueel in de pagina wordt getoond, is volledig afhankelijk van CSS. U hebt hier dezelfde vrijheid die elke webdesigner heeft. De applicatie hoeft er niet als een boomstructuur uit te zien.

Met CSS kan bijvoorbeeld een component header boven in de pagina worden getoond, een component menu aan de linkerkant, de component todo centraal in de pagina en daaronder een component footer. Kortom, de interne structuur van de applicatie heeft niks te maken met de visuele weergave.



Afbeelding 1.13 *CSS is nog steeds verantwoordelijk voor positionering en weergave van de componenten.*

Tot zover een korte inleiding in de praktijk en architectuur van Angular-applicaties. Zodra u in hoofdstuk 2 met code aan de slag gaat, wordt het allemaal een stuk duidelijker.

Benodigde voorkennis

Dit boek maakt deel uit van de *Web Development Library* (**www.webdevelopmentlibrary.nl**). In elk deel wordt een op zichzelf staande techniek besproken die te maken heeft met webdevelopment. Andere, gerelateerde technieken worden bekend verondersteld. Zo betaalt u alleen voor wat u echt nodig hebt.

In dit boek gaan we in op Angular 2. Het is een client-sided JavaScript-framework. We gaan er dan ook van uit dat u voldoende ervaring hebt met JavaScript. Variabelen, statements, functies en objecten worden niet apart besproken. Hoewel Angular 2 wel erg *logisch* is, is het niet het makkelijkste framework om te leren. Er wordt redelijk diepgaande kennis van JavaScript-concepten bekend verondersteld.

Kennis van ECMAScript 2015 en TypeScript is zeker handig, maar niet beslist noodzakelijk. Nieuwe onderdelen uit deze programmeertalen worden kort besproken.

Wat hoeft u niet te weten?

Kennis van andere JavaScript-bibliotheken (zoals jQuery) of JavaScript-frameworks is niet nodig. Ook hoeft u niks te weten van server-sided talen als PHP, Python, Ruby, Java of C#. Het werken met databases doen we alleen in de context van een Ajax-call. We gaan niet zelf aan de slag met het selecteren en retourneren van gegevens uit databases via SQL of NoSQL.

Bovendien wordt Angular 2, zoals u inmiddels hebt begrepen, gebruikt in de context van een website. U moet daarom kennis hebben van HTML en CSS. Tags, attributen, CSS-selectors, klassen en id's worden bekend verondersteld.

Kennis van Angular 1

We hebben al uitgelegd dat de frameworks erg van elkaar verschillen. Kennis van eerdere versies van Angular is dan ook niet noodzakelijk. Sterker nog, het kan zelfs in de weg zitten. Als u eerder al met Angular 1.x hebt gewerkt, zult u veel dingen moeten afleren. In Angular 2 gaat alles op een andere manier en bekende concepten (`$scope`, `angular.module()`, `ng-controller`, `directives` en meer) bestaan niet meer. Angular 2 is waarschijnlijk makkelijker te leren voor een nieuwkomer dan voor een Angular-veteraan die al jaren ervaring heeft met het framework.

Tips voor meer lezen

Kan uw HTML-, of JavaScript -voorkennis wel een opfrisbeurt gebruiken? Lees dan eerst de volgende titels (van dezelfde auteur):

- *Web Development Library – JavaScript*, ISBN 9789059407589
- *Web Development Library – Node.js*, ISBN 9789059408371