

# Inhoud

<b>1</b>	<b>Kennismaken met Angular</b>	<b>1</b>
	<b>Wat is Angular?</b>	<b>2</b>
	Bibliotheken en raamwerken	2
	Angular is geen bibliotheek	3
	Wat is het verschil?	3
	Wat is een SPA-framework?	4
	Omschrijving van Angular	4
	Angular op internet	7
	<b>Enkele woorden over de voorganger, AngularJS</b>	<b>7</b>
	Angular	8
	Changelog lezen	9
	Angular Update Guide	10
	<b>Angular-concepten</b>	<b>11</b>
	Modulair programmeren en componenten	11
	Modules	12
	Dependency injection	13
	Consistentie	14
	Programmeertalen	15
	Webstandaarden	15
	Snelheid	16
	<b>Architectuur van Angular-applicaties</b>	<b>17</b>
	Single page application	18
	Angular-begrippen	19
	<b>Applicatie als boomstructuur van componenten</b>	<b>19</b>
	De boomstructuur visueel maken	21
	<b>Enkele woorden over React</b>	<b>22</b>
	Overige boeken	23
	<b>Benodigde voorkennis</b>	<b>24</b>
	Tips voor meer lezen	25
	Waarom een boek?	25
	<b>De ontwikkelomgeving inrichten</b>	<b>26</b>
	Editor en browser	26
	Node.js	27
	Angular CLI	29
	Debugging en Augury	30

	<b>Oefenbestanden downloaden</b>	<b>31</b>
	<b>Samenvatting</b>	<b>32</b>
	<b>Praktijkoefeningen</b>	<b>33</b>
<b>2</b>	<b>Hello World in Angular</b>	<b>35</b>
	<b>Mogelijkheden voor Angular-projecten</b>	<b>36</b>
	<b>Angular CLI installeren</b>	<b>36</b>
	<b>Nieuw project starten en draaien</b>	<b>38</b>
	<b>Project openen en aanpassen</b>	<b>40</b>
	<b>Theorie – de bestandsstructuur verkennen</b>	<b>42</b>
	Mappen	42
	<b>Belangrijke bestanden</b>	<b>44</b>
	Package.json	44
	Tsconfig.json	46
	Angular.json	48
	<b>Overige bestanden</b>	<b>49</b>
	<b>Praktijk – een nieuwe component genereren</b>	<b>50</b>
	<b>Theorie – meer over componenten</b>	<b>53</b>
	<b>Theorie – modules bekijken</b>	<b>56</b>
	De bootstrapper main.ts	58
	De startpagina index.html	60
	Selector app-root	61
	De rol van ng serve	61
	<b>Praktijk – CSS-bibliotheek Bootstrap toevoegen</b>	<b>63</b>
	<b>Architectuur van Angular-applicaties</b>	<b>66</b>
	<b>Samenvatting</b>	<b>68</b>
	<b>Praktijkoefeningen</b>	<b>70</b>
<b>3</b>	<b>Databinding en modellen</b>	<b>73</b>
	<b>Wat is databinding?</b>	<b>74</b>
	Declaratieve syntaxis	75
	Vier typen databinding	75
	<b>Eenvoudige databinding met {{ ... }}</b>	<b>76</b>
	Voorbeeldbestanden downloaden en gebruiken	77
	TypeScript benutten	78
	TypeScript is optioneel	80
	<b>Databinding in de constructor</b>	<b>81</b>
	<b>Databinding in ngOnInit()</b>	<b>82</b>
	<b>De directive *ngFor</b>	<b>84</b>
	Wat is een directive?	84
	De component uitbreiden met een array	85
	Data – tijdelijk – in de component	87

Stap 1 – een model maken	88
Analyse	89
Stap 2 – model gebruiken in de applicatie	90
Stap 3 – view aanpassen	90
Interface gebruiken	91
<b>De directive *ngIf</b>	<b>92</b>
Booleaanse waarde in de klasse	93
<b>Werken met inline en externe sjablonen</b>	<b>94</b>
TemplateUrl	94
Inline sjablonen en backticks; alles in één bestand	95
<b>Samenvatting</b>	<b>96</b>
<b>Praktijkoefeningen</b>	<b>97</b>
<b>4 Meer over databinding</b>	<b>99</b>
<b>Gegevens binden aan gebeurtenissen</b>	<b>100</b>
Er zijn erg veel gebeurtenissen	100
Notatie voor gebeurtenisbinding	101
Andere gebeurtenissen verwerken	102
<b>Parameters meegeven aan de gebeurtenisverwerker</b>	<b>104</b>
<b>Werken met lokale sjabloonvariabelen</b>	<b>106</b>
Nieuwe stad toevoegen via tekstvak	106
Steden toevoegen aan de array	108
<b>Gegevens binden aan HTML-attributen</b>	<b>110</b>
Voorbeeld attribuutbinding	111
Afbelingen tonen via attribuutbinding	112
Analyse	113
Vraag	115
<b>Tweerichtingdatabinding met [(ngModel)]</b>	<b>115</b>
[(ngModel)] gebruiken	116
Formuliermodule importeren	117
Geen parameter meer bij [(ngModel)]	118
Wanneer [(ngModel)] gebruiken?	119
<b>Meer opties voor binding</b>	<b>119</b>
<b>Samenvatting</b>	<b>120</b>
<b>Praktijkoefeningen</b>	<b>121</b>
<b>5 Werken met services</b>	<b>123</b>
<b>Wat zijn services?</b>	<b>124</b>
Services in AngularJS tegenover Angular	125
Datastream via services	125
De rol van Injectable()	126

<b>Stap 1 – service met statische data</b>	<b>127</b>
Wat betekent getCities(): City[]?	129
Public en private in TypeScript	129
<b>Stap 2 – service gebruiken in de component</b>	<b>130</b>
Service instantiëren in de constructor	131
Het werkt niet... de array providers [] in de module	131
<b>Stap 3a – dependency injection in app.module.ts</b>	<b>132</b>
<b>Stap 3b – gebruik de notatie providedIn</b>	<b>133</b>
Services via Angular CLI	134
<b>Stad toevoegen via de service</b>	<b>134</b>
View aanpassen	135
Klasse aanpassen	135
Service aanpassen	136
Resultaat	136
<b>Samenvatting</b>	<b>137</b>
<b>Praktijkoefeningen</b>	<b>138</b>
<b>6 Asynchrone services</b>	<b>139</b>
<b>Wat zijn asynchrone services?</b>	<b>140</b>
Synchroon en asynchroon	140
HttpClientModule en de service HttpClient	141
<b>Theorie – meer over reactive programming</b>	<b>142</b>
De observable gegevensstroom (stream)	142
De rol van ReactiveX	145
<b>Theorie – de werking van ReactiveX</b>	<b>146</b>
De observable sandwich	147
Promises vergeleken met observables	148
Cities.json	149
<b>Praktijk – gegevens uit bestand lezen en verwerken</b>	<b>150</b>
Stap 1 – module aanpassen	150
Stap 2 – AppComponent aanpassen	151
Analyse	152
<b>Meer RxJS-methoden</b>	<b>153</b>
Importeren	154
De observable sandwich in code	155
<b>Verbetering – gegevens via de service</b>	<b>156</b>
<b>Automatisch abonnement met de pipe async</b>	<b>158</b>
De notatie   async	158
Extra pipe()	160
<b>Live API's op internet gebruiken</b>	<b>161</b>
Kenmerken van de applicatie	161
MovieService	164
Waarom res['Search']?	165

<b>Een API maken met json-server</b>	<b>166</b>
Json-server	166
Stap 1 – eisen aan de applicatie	167
Stap 2 – json-server toevoegen en testen	167
Stap 3 – de service aanpassen	169
Stap 4 – component aanpassen om nieuwe city toe te voegen	171
Stap 5 – stad toevoegen via service	173
Stap 6 – detailgegevens per stad tonen	174
Asynchroon *ngIf	176
<b>Meer API's om mee te experimenteren</b>	<b>177</b>
Registreren voor API-sleutel	178
Open API's	178
<b>Samenvatting</b>	<b>179</b>
<b>Praktijkoefeningen</b>	<b>180</b>
<b>7 Boomstructuur van componenten</b>	<b>183</b>
<b>Structuur van Angular-applicaties</b>	<b>184</b>
Componenten zijn niet verplicht	184
Componenten zijn wel handig	184
<b>Nieuwe componenten maken</b>	<b>186</b>
Werkwijze bij meerdere componenten	186
<b>Stap 1 – nieuwe component maken</b>	<b>187</b>
<b>Stap 2 – view aanpassen</b>	<b>188</b>
Optioneel: model uitbreiden	189
<b>Stap 3 – insluiten in HTML</b>	<b>189</b>
<b>Datastroom tussen componenten</b>	<b>191</b>
<b>Werken met @Input</b>	<b>192</b>
Oudercomponent aanpassen	193
Mogelijkheden	194
<b>Werken met @Output</b>	<b>195</b>
Werken met gebeurtenissen	195
Werkwijze bij @Output	196
Case – waardering geven aan steden	197
Stap 1 – Detailcomponent aanpassen	197
Stap 2 – oudercomponent voorbereiden op ontvangen van gebeurtenis	198
Stap 3 - waardering tonen in de HTML	199
<b>Samenvatting @Input en @Output</b>	<b>200</b>
<b>Communicatie tussen componenten op gelijk niveau</b>	<b>201</b>
Directe communicatie tussen siblings	201
Case – communicatie via event bus	202
Stap 1 – Model aanpassen en nieuw model maken	203
Stap 2 – OrderService schrijven met observers en observables	205
Stap 3 – OrderService importeren en gebruiken	206

Stap 4 – Component voor orders maken	208
Stap 5 – alles samenvoegen (let op!)	211
Het oog wil ook wat	211
De ordercomponent testen	212
Meer lezen over observables	213
<b>Samenvatting</b>	<b>214</b>
<b>Praktijkoefeningen</b>	<b>215</b>
<b>8 Routing en bronnen voor meer informatie</b>	<b>217</b>
<b>Kennismaken met routing</b>	<b>218</b>
Single page application of SPA	218
Architectuur bij routing	220
De rol van router-outlet	220
<b>Routing bij gebruik van Angular CLI</b>	<b>221</b>
<b>Stappenplan bij routing</b>	<b>223</b>
Stap 1 – base href controleren	223
Stap 2 – RouterModule importeren en routes maken	224
Stap 3 – nieuwe componenten maken	225
Stap 4 – AppComponent aanpassen	226
Stap 5 – testen	228
<b>Programmatisch een andere route selecteren</b>	<b>229</b>
<b>Dynamische routes met routeparameters</b>	<b>230</b>
Stap 1 – routetabel uitbreiden	230
Stap 2 – HomeComponent voorbereiden	231
Stap 3 – DetailComponent maken	231
Zoeken in backend	233
<b>Meer over routing</b>	<b>234</b>
<b>Meer over Angular</b>	<b>236</b>
Formulieren	236
Testen	237
Modules en lazy loading	238
Angular Elements	239
State management met @ngrx/store	240
PWA's	240
Tot slot	242
<b>Samenvatting</b>	<b>243</b>
<b>Praktijkoefeningen</b>	<b>244</b>
<b>Index</b>	<b>247</b>

# Kennismaken met Angular

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

## In dit hoofdstuk:

*Wat Angular is, en wat het niet is.*

*Enkele woorden over AngularJS.*

*Concepten en kenmerken van Angular.*

*Benodigde voorkennis en software.*

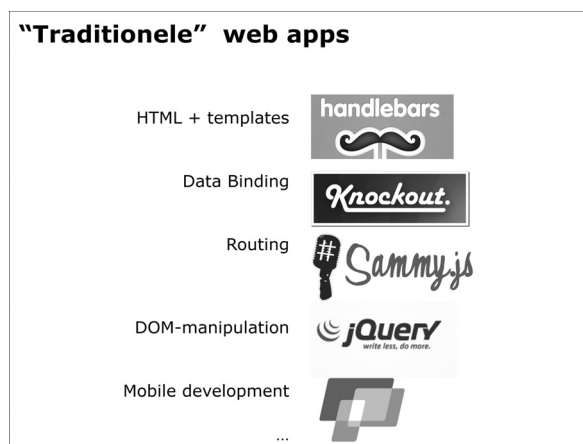
*Wat hebt u nodig? De werkomgeving inrichten.*

## Wat is Angular?

Het aloude HTML is prima om 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.

### Bibliotheken en raamwerken

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



**Afbeelding 1.1** In traditionele webapplicaties neemt elke bibliotheek een van de eisen die aan de applicatie wordt gesteld voor zijn rekening. De kans bestaat dat bibliotheken niet compatibel zijn.



## Angular is geen bibliotheek

Angular is geen bibliotheek zoals de hiervoor genoemde. Angular is een compleet raamwerk (*framework*) voor het realiseren van client-sided webapplicaties.

### Wat is het verschil?

- Als we de zaken erg vereenvoudigd voorstellen, kan worden gezegd dat bibliotheken in het algemeen één ding heel goed doen. Voorbeelden van bibliotheken voor webontwikkeling zijn jQuery, React en underscore.
- Een raamwerk zoals Angular biedt oplossingen voor alle niveaus van applicatieontwikkeling. Van het structureren en binden van data tot Ajax-communicatie met webserver, het verwerken van geretourneerde gegevens in een client-sided datamodel en het maken van herbruikbare componenten. Voorbeelden van andere frameworks dan Angular zijn Vue.js, Aurelia en Ember.

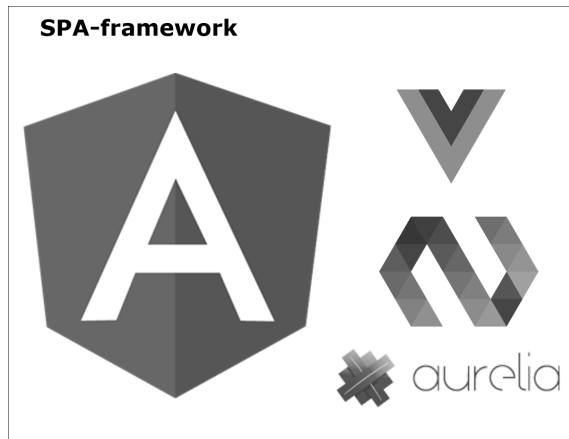
Bibliotheken kunnen over het algemeen gecombineerd worden in een project om gezamenlijk het beste resultaat te bereiken. Bij raamwerken wordt daarentegen één keuze gemaakt en wordt de app gebouwd volgens de richtlijnen en kenmerken van het gekozen raamwerk.



### Geen combinaties

We zullen in de praktijk bijvoorbeeld nooit zien dat een app tegelijk Angular en Vue.js gebruikt. Ook combinaties van Angular met Polymer of Ember (andere alternatieven) komen niet voor. U bouwt de site ofwel in Angular, ofwel in Vue. Niet in beide.

---



**Afbeelding 1.2** *In een raamwerk zoals Angular, maar ook in alternatieven zoals Polymer, 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.*

## Wat is een SPA-framework?

Bij Angular (maar ook bij andere raamwerken) wordt vaak de afkorting SPA gebruikt. Dit staat voor *Single Page Application*. Een Angular-applicatie is namelijk niet meer opgebouwd uit allerlei verschillende webpagina's zoals een traditionele website.

In plaats daarvan bestaat de website uit één enkele pagina met de naam `index.html`. Binnen de indexpagina worden vervolgens allerlei Angular-componenten geladen die tezamen de pagina's van de website vormen. Verderop in dit hoofdstuk wordt nog dieper ingegaan op het begrip Single Page Application.

## Omschrijving van Angular

Het raamwerk Angular wordt op de officiële site omschreven als

*"One framework. Mobile & desktop."*

Dat geeft het doel voldoende aan, dachten wij zo. Met Angular is het relatief eenvoudig om complexe webapplicaties te schrijven, omdat het raamwerk als het ware een abstractielaag biedt tussen de browser, de logica van de app en de gegevens 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 maakt u JavaScript-klassen die het model representeren.
- **Controller** De logica in de applicatie bewerkt data in het model. Deze voegt bijvoorbeeld losse velden zoals `firstName` en `lastName` samen tot een veld `fullName` dat in de gebruikersinterface (*user interface*) wordt getoond.
  - In Angular is de controller ook weer een JavaScript-klasse. De controller hoort altijd bij een bepaalde component.
- **View** De gebruikersinterface bestaat uit HTML-sjablonen (*templates*) waarin de – eventueel bewerkte – gegevens worden getoond. De HTML-sjabloon is daarmee de view van de applicatie.
  - Ook nu weer behoort de view van een component tot een bepaalde JavaScript-klasse in een Angular-applicatie. Er zijn – met uitzondering van `index.html` – geen ‘losse’ HTML-pagina’s meer in de applicatie. Alles is gebonden aan een component.

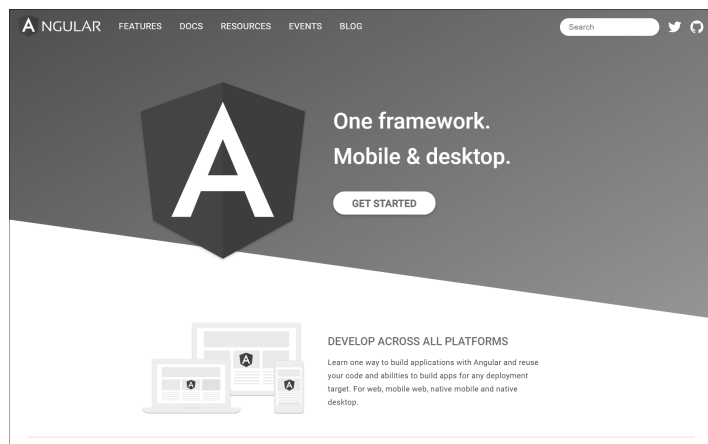
Angular is daarmee, net als zijn voorganger AngularJS, een compleet client-sided MVC-framework. Het is volledig in JavaScript en TypeScript geschreven en draait ook volledig in de browser.

Idealiter is de app compleet ontkoppeld van de server en database waar de gegevens vandaan komen. Alle communicatie vindt plaats via Ajax-aanroepen. Uiteraard gaan we ook hier later in dit boek nog dieper op in.



### Geen 'MVC' meer

Het begrip MVC om de architectuur voor Angular aan te duiden wordt langzamerhand minder gebruikt. De terminologie van MVC zou te strikt zijn en niet goed passen bij de flexibiliteit van Angular. Uit andere technologieën kent u misschien het begrip *MVVM (Model-View-View-Model)*. 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](https://angular.io). Begin hier voor officiële downloads, documentatie en meer.

## Angular op internet

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

Wilt u helemaal hardcore gaan, dan kunt u een eigen versie van Angular bouwen. Ga naar **github.com/angular/angular** om de broncode te bekijken, te builden en eventueel aan te passen voor eigen gebruik. In dit boek doen we dat niet.

## Enkele woorden over de voorganger, AngularJS

Voordat er Angular was, was er AngularJS (ja, we weten het. De naamgeving *is* verwarrend. Wij hebben het ook niet verzonnen). AngularJS is een oude versie en staat ook wel bekend als Angular 1.x. Deze 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 raamwerk dat ook door anderen gebruikt kon worden. Rond 2011 gaf Google het onder de MIT-licentie vrij als opensourcesoftware.

Daarna volgde AngularJS min of meer het gebruikelijke upgrade-pad. Er werden nieuwe mogelijkheden 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.

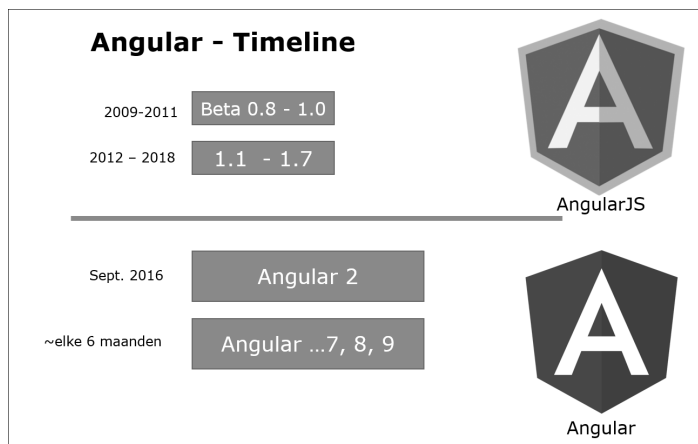


## Kennis van AngularJS niet belangrijk

Om met Angular aan het werk te gaan hoeft u niets te weten van eerdere versies. Kennis van AngularJS is dus onnodig.

## Angular

Heel anders is dat sinds de introductie van Angular 2. Er zijn eigenlijk precies zeven overeenkomsten tussen AngularJS en Angular 2+. Dat zijn de letters A, N, G, U, L, A, en R. Het is niet overdreven om te zeggen dat Angular een compleet nieuw en ander raamwerk is, met toevallig dezelfde naam. Daar zijn wel redenen voor, 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 overstapt naar Angular. Afhankelijk van de codeerstijl van een Angular 1.x-app zal het niet, of alleen met een aanzienlijke investering in tijd (en dus kosten) mogelijk zijn om een project te upgraden naar Angular of hoger (momenteel meestal minimaal Angular 8).

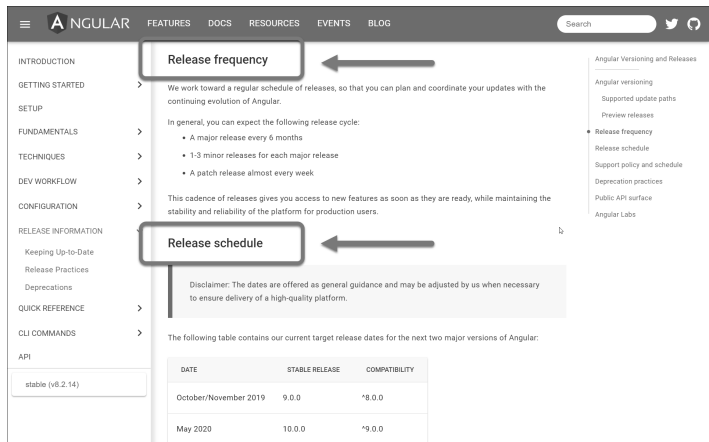


**Afbeelding 1.4** *Versies van AngularJS (Angular 1.x) en Angular. Het upgraden van AngularJS-toepassingen naar Angular 2 en hoger is lastig. Daarom staat er een streep tussen.*



## In dit boek: Angular 9

In dit boek gebruiken we Angular 9, uit voorjaar 2020. Dit was op het moment van schrijven de meest recente versie. Maar de voorbeelden in dit boek zijn geschikt voor elke applicatie die werkt met Angular 5 of hoger. Team Angular probeert elke zes maanden een nieuwe *major release* uit te brengen. De onderlinge verschillen zijn echter vaak klein en alleen in gespecialiseerde situaties belangrijk. Lees wel altijd goed de releaseopmerkingen om te zien wat er nieuw is, en of er *breaking changes* zijn met voorgaande versies.



The screenshot shows the Angular website's documentation for release frequency and schedule. The 'Release frequency' section is highlighted with a red box and an arrow. The 'Release schedule' section is also highlighted with a red box and an arrow. Below the schedule section is a table with columns for DATE, STABLE RELEASE, and COMPATIBILITY.

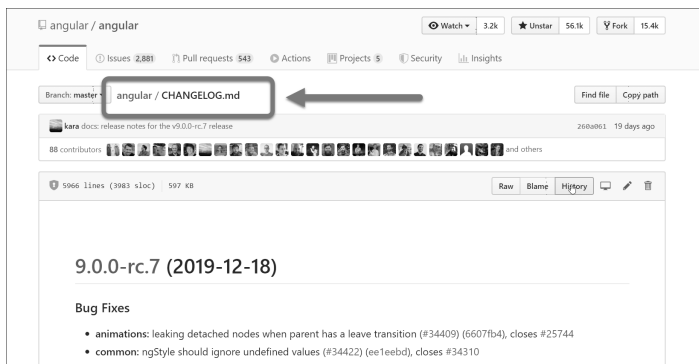
DATE	STABLE RELEASE	COMPATIBILITY
October/November 2019	9.0.0	*8.0.0
May 2020	10.0.0	*9.0.0

**Afbeelding 1.5** Versies volgen elkaar in hoog tempo op. Lees de laatste ontwikkelingen in het releaseschema. Googel eventueel naar Angular Release Schedule om dit makkelijk te vinden.

## 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

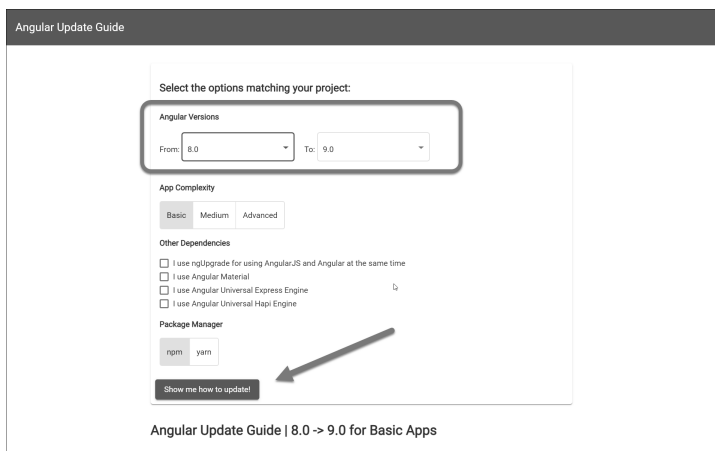
zijn gerepareerd. Pak deze pagina er daarom bij als u zelf uw versies van Angular wilt updaten. Vergeet niet om in dat geval ook de versienummers in package.json aan te passen (zie verderop).



**Afbeelding 1.6** *Saaï, maar noodzakelijk bij applicatiebeheer. Lees de wijzigingen en eventuele 'breaking changes'. Ze staan in changelog.md op [github.com/angular/angular](https://github.com/angular/angular).*

## Angular Update Guide

Als u googelt op Angular Update Guide vindt u bovendien een applicatie die aangeeft wat de beste manieren zijn om applicaties



**Afbeelding 1.7** *De Angular Update Guide kan handig zijn bij het opwaarderen van bestaande Angular-applicaties.*



te upgraden. Dit is uiteraard alleen van belang als er al applicaties zijn die moeten worden bijgewerkt naar een nieuwere versie. In dit boek gebruiken we dit niet. U leert nieuwe Angular-apps maken vanaf het nulpunt. We gaan er niet van uit dat er al code aanwezig is.

---



### Direct aan de slag

Wilt u direct met een eerste Angular-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 high-level achtergronden bij Angular.

---

## Angular-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 toegepast in het raamwerk. In de loop van het boek komen al deze concepten vanzelfsprekend aan de orde. Hier noemen we alvast kort de belangrijkste.

### Modulair programmeren en componenten

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

- JavaScript-statements `import` en `export` die aangeven welke afhankelijkheden (*dependencies*) de component heeft en welke onderdelen herbruikbaar zijn in andere componenten (geëxporteerd worden);

- TypeScript-componentannotatie die aangeeft hoe de component werkt (`@Component`). Dit heet ook wel een *decorator*;
- een HTML-sjabloon met de gebruikersinterface (de *view*) van de component;
- een JavaScript-klasse met de logica van de component (de *controller*).

```
// Opbouw van een component, app.component.ts
// 1. imports
import {Component, OnInit} from '@angular/core';

// 2. Decorator
@Component({
  ...
})

// 3. Klasse
export class AppComponent implements OnInit {
  ...
}
```

Componenten zijn in principe containers 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-app.

## Modules

Een component is altijd opgenomen in een module. Componenten draaien niet uit zichzelf. Hoewel er stemmen zijn om modules uit toekomstige versies van Angular te verwijderen (of in ieder geval optioneel te maken), bestaan Angular-toepassingen op dit moment nog uit minimaal de volgende onderdelen:

- Elke applicatie heeft minimaal één module (maar vaak meer).
  - De standaardnaam hiervoor is de klasse `AppModule`, in het bestand `app.module.ts`. We adviseren om deze naam-

geving aan te houden, maar technisch gezien mag u elke naam kiezen die u wilt.

- Elke module bevat minimaal één component (maar vaak zijn dit er – veel – meer).
  - De standaardnaam hiervoor is de klasse `AppComponent`, in het bestand `app.component.ts`. Ook hiervoor mag u in principe elke gewenste naam kiezen. Voor de duidelijkheid heet de startcomponent echter meestal `AppComponent`.

Modules kunt u hiermee beschouwen als een soort mini-applicaties. Modules kunnen in andere modules worden opgenomen om zo de functionaliteit van de hoofdmodule uit te breiden. Dit zullen we later in dit boek zien als we bijvoorbeeld gebruikmaken van de `HttpClientModule` (om http-mogelijkheden aan de app toe te voegen) of de `FormsModule` importeren (om formulieren in de app te kunnen gebruiken).



### Java en .NET-achtergrond

Veel meer dan andere raamwerken voor web-development is Angular een raamwerk voor programmeurs. Vooral degenen met een achtergrond in Java of .NET kunnen hun hart ophalen. Eindelijk is het mogelijk om ook op het web met klassen, constructors, getters en setters te werken. Voor die programmeurs is de overstap naar Angular eenvoudiger dan bijvoorbeeld naar React, Aurelia of AngularJS 1.x. Wie gewend is aan traditioneel webdevelopment waarbij voornamelijk wordt gewerkt in HTML-pagina's en JavaScript, is Angular 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 AngularJS zijn componenten zelf verantwoordelijk voor het opvragen van afhankelijkheden. In AngularJS moeten afhankelijke modules op het moment van instantiëren van de applicatie ingevoegd worden. U krijgt in dat geval code als:

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

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

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

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

## Consistentie

AngularJS 1.x was het resultaat van een gestage, jarenlange ontwikkeling, waarbij telkens ‘nieuwe ballen in de kerstboom werden gehangen’. Het was destijds een revolutionair raamwerk en bood mogelijkheden die geen enkel ander raamwerk had. Maar achteraf gezien zou het in sommige gevallen beter zijn geweest om

eerst wat langer na te denken over het implementeren van bepaalde concepten.

In Angular zijn beslissingen beter doordacht. Het voordeel van Angular is dat er vooraf lang is nagedacht over hoe het raamwerk moet functioneren en hoe componenten moeten worden genoemd. Het is daarmee veel consistent en eenduidiger dan AngularJS 1.x. Ook de werking van databinding, attribuutbinding en gebeurtenisbinding is veel consistent dan in Angular 1.

## Programmeertalen

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



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

## Webstandaarden

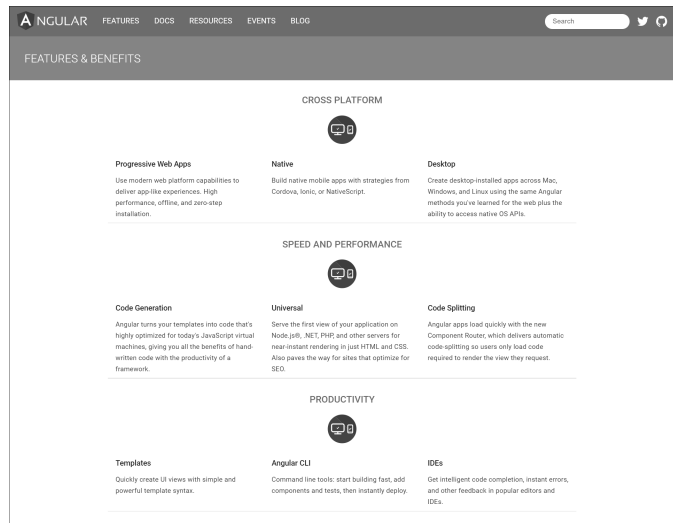
Veel beter dan AngularJS is Angular voorbereid op het gebruik van nieuwe webstandaarden. Ook dit heeft weer met de geschiedenis te maken. De basis van AngularJS stamt al uit 2009. Toen hadden we juist de iPhone 3GS in Nederland. Android 2.x (!) en het mobiele web stonden nog in de kinderschoenen. Nu, in 2020, hebben we TypeScript, ECMAScript 2015, standaarden voor websockets, webworkers, serviceworkers, lokale opslag, nieuwe HTML5-API's en nog veel meer. Hier is Angular veel beter op voorbereid.

## Snelheid

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

- **Aan de gebruikerskant** Angular-apps voelen snel aan in gebruik, door het DOM alvast in het geheugen aan te passen op basis van nieuwe data (schaduw-DOM) en door minimaal geheugengebruik en waar mogelijk eigen (*native*) technieken te gebruiken op mobiele apparaten. Angular 9 kent een nieuwe *rendering engine* (met de naam Ivy) die HTML nog sneller kan renderen, kleinere applicaties oplevert en de gebruiker sneller met de applicatie laat werken.
- **Aan de netwerkkant** Team Angular streeft er naar de voetafdruk van een minimale Hello World Angular-applicatie (bibliotheek plus uitvoerbare code) zo klein mogelijk maken, zodat de code snel geladen wordt en snel verwerkt kan worden door de pc, smartphone of tablet. Door *ahead of time*-compilatie te gebruiken, worden Angular-applicaties al op de server gecompileerd, zodat de browser direct aan de slag kan met de code die hij ontvangt.
- **Aan de ontwikkelaarskant** Omdat Angular een veel consistentere gedrag vertoont, kunnen ook de hulpmiddelen zoals compilers en editors daar op inspringen. Door TypeScript te gebruiken, kunt u bijvoorbeeld als ontwikkelaar al tijdens het compileren (*compile-time*) zien dat er nog fouten in staan, in plaats van af te wachten totdat een fout tijdens het uitvoeren (*run-time*) optreedt. Goede editors zoals Visual Studio Code en JetBrains WebStorm geven op tal van plekken tips voor automatisch aanvullen, inzicht in de werking, parameters van klassen en nog veel meer.

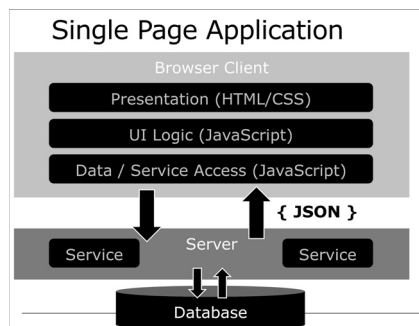
Meer kenmerken en eigenschappen van Angular leest u bijvoorbeeld op de pagina Features & Benefits, op [angular.io/features](https://angular.io/features).



**Afbeelding 1.9** Lees op internet meer over wat Angular allemaal in huis heeft.

## Architectuur van Angular-applicaties

In principe zijn Angular-applicaties toepassingen die volledig zelfstandig in de browser draaien. Er zijn ook varianten, zoals Angular in een Node.js-applicatie op de server, of in een zelfstandige app die is gemaakt met Electron, Ionic of NativeScript. Daar gaan we in dit boek echter niet op in. We concentreren ons op webapplicaties 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, datatoegang en authenticatie draaien op de server.