Wat is angular en wat is het niet (angular.io)

- Bibs en frameworks:
- Dom manipulatie (jquery)
- Routing (sammy.js)
- Data Binding (backbone,knockout.js)
- Datums (moments.js)

Traditionele webapp -> libs niet compatible

Angular is geen bibliotheek, maar een compleet framework voor client-sided SPA's
"One framework - mobile and desktop)

Het is relatief eenvoudig om complexe webapplicaties te schrijven omdat het framework een abstractielaag biedt tussen browser, logica van de app en data! Dit kennen we als MVC.

Geen MVC: MVC wordt tegenwoordig minder gebruikt, MVVM (model-view view-model)

Angular 2+ en hoger vs AngularJS

AngularJS (oude versie, ook wel gekend als angular 1.x) gestart in 2009 als intern project bij Google. Angular is totaal anders met toevallig dezelfde naam.(sept 2016 angular 2+, elke 6 maanden 4,5,6,7…)

changelog lezen is ook belangrijk. voornamelijk naar upgrades toe :)

Angular concepten

Modulair programmeren en componenten

kernwoord = componenten.

Een angular app wordt volledig opgebouwd uit componenten en bestaat uit volgende onderdelen:

- Typescript componenten notatie geeft aan hoe component werkt (@Component)
- en HTML-sjabloon met interface (de view)
- Een javascript-klasse met logica van de componenten (de controller)
- Javascript-statements import en export dewelke aangegeven welke afhankelijkheden (dependencies) het component heeft en welke onderdelen herbruikbaar zijn in andere componenten (geëxporteerd worden)

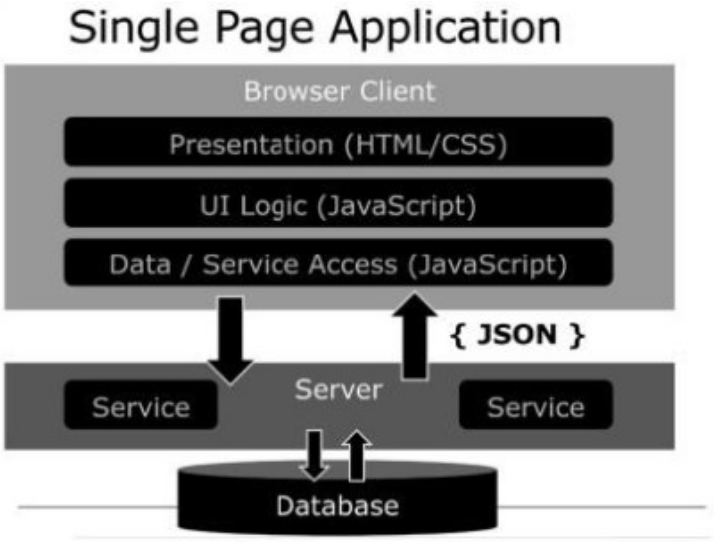
nog meer dan angularJS is een app in angular opgebouwd uit componenten. Componenten zijn containers die alle logica en informatie over de interface bevatten om de componenten te laten werken.

Architectuur van de 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 welke gemaakt worden met Electron (Desktop) of Ionic (Mobile)

Single page application

index.html is de default ingeladen pagina die de javascript aanspreekt. Vandaar de SPA of single page application.



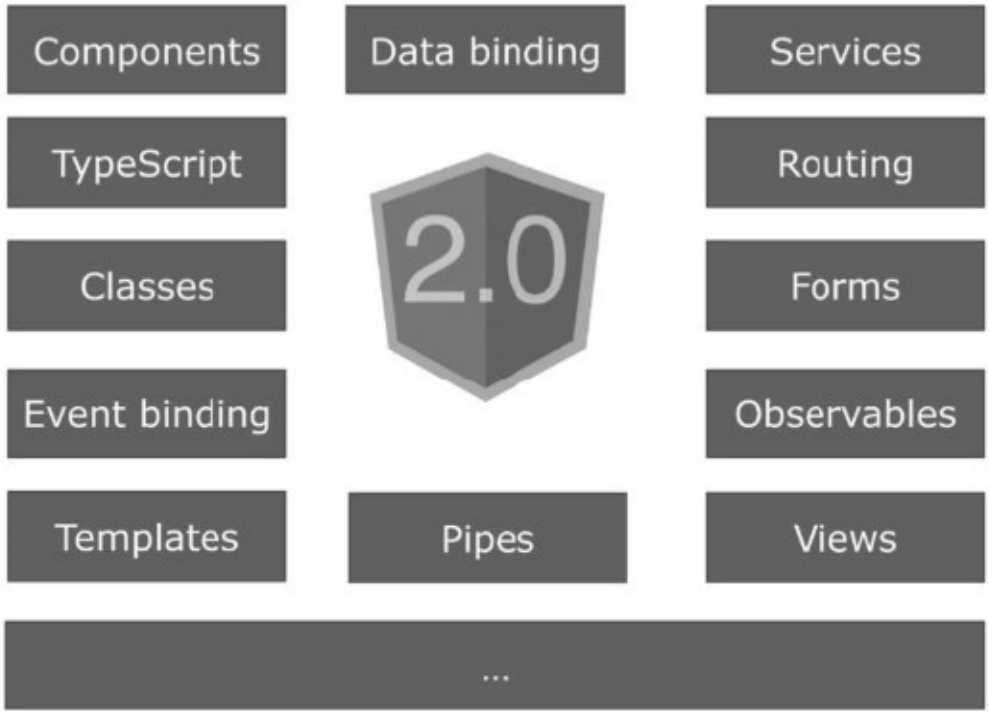
De architectuur die we nastreven in Angular-applicaties: de logica- en presentatielaag draaien in de browser, datatoegang en authenticatie draaien op de server.

De app zelf bestaat natuurlijk uit veel meer dan één bestand. Elk component staat in zijn eigen map, er zijn css-bestanden, afbeeldingen enz…

- De rol van de browser:
    - Presentatielaag tonen aan de gebruiker, via de view componenten. Dit is gewoon HTML en CSS zoals we die kennen met Angular-specifieke toevoegingen.
    - Gebruikersinterface Logica via de controller (klasse) van een view. Hierin staat logica om de gebruikersinterface samen te stellen, gebeurtenissen zoals muisklikken te verwerken of het verwerken van Ajax-aanroepen en meer.
    - Data- en service toegang, oftewel communiceren met één of meer RESTful API's op de server. De API praat vervolgens met de database en retourneert gegevens, bij voorkeur in het bestandsformaat JSON.
    - Authenticatie, op het terrein van toegang is de server nog steeds onontbeerlijk. De complete Angular-app draait immers in de browser en is daarmee onveilig. Authenticatie en autorisatie zijn nog steeds het domein van de server. De server moet vervolgens een token of auth cookie uitreiken (afhankelijk van de wijze van beveiliging welke is gekozen). De Angular-app is er verantwoordelijk voor dat het token of de cookie met een volgend verzoek wordt meegezonden.

Angular-begrippen

Om onderstaande architectuur te realiseren, moeten algemene termen als presentation, ui logic en data/service-access natuurlijk concreet worden gemaakt. Dit gebeurt in Angular-apps met begrippen als observables, routing, services en meer.



Applicatie als boomstructuur van componenten

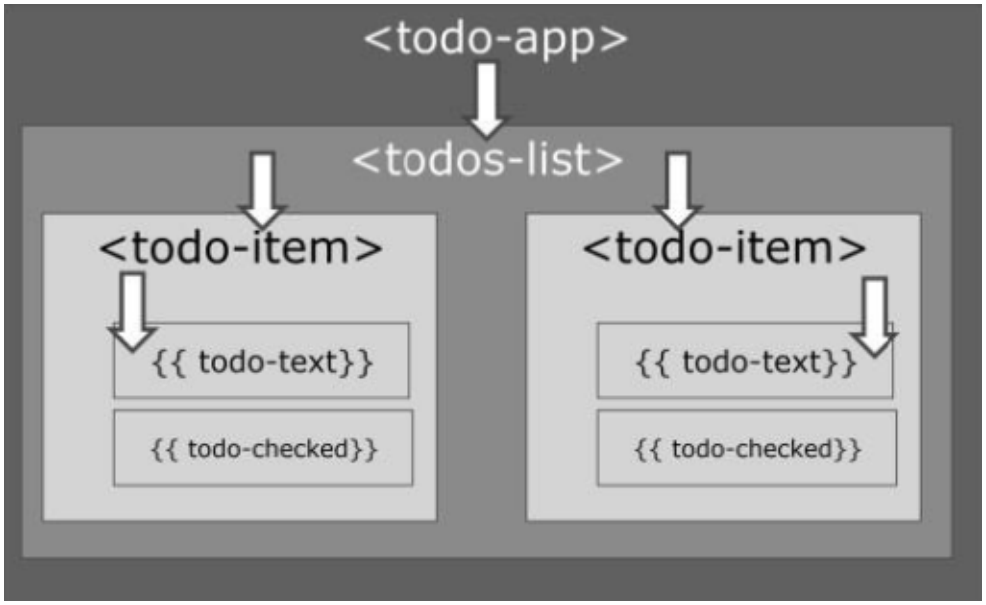
We hebben de term al een aantal keer genoemd. In de Angular wereld staat het begrip component centraal. Elke applicatie heeft exact één hoofdmodule en één hoofdcomponent (de root component)

- Binnen die root component worden deelcomponenten geladen
- Elke deelcomponent heeft een eigen verantwoordelijkheid, en met een eigen gebruikersinterface en eigen logica.
- Deelcomponenten kunnen op hun beurt weer andere componenten bevatten. Deze worden met HTML-tags ingesloten in de sjabloon (template) van de bovenliggende component.
- Het maken van een Angular-applicatie bestaat dus vaak uit het bouwen van erg veel kleine, zelfstandige componenten die met elkaar kunnen samenwerken. Legoblokjes… die onderling een logische samenhang hebben en de app vormen.

Schematisch kan dit er bijvoorbeeld uitzien zoals de afbeelding hieronder:

Een todo-applicatie voorstelling…

- De applicatie wordt in de index.html geladen met de selector <todo-app>. Hoe we zo een selector maken leren we nog.
- De todo-applicatie bestaat op zijn beurt weer uit een lijst. Deze wordt ingesloten via <todo-list>
- Eelke 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, eventuele knoppen, andere gebruikersinterface-elementen enz…



We onthouden: Elke Angular-applicatie bestaat uit een boomstructuur van componenten!!!

Boomstructuur visueel maken

Er is dus altijd één hoofdcomponent (root). Meer meer, niet minder. Hoe die boomstructuur vervolgens in de pagina wordt getoond, is volledig afhankelijk van de CSS. De app hoeft er uiteraard niet uit te zien als een boomstructuur, en creativiteit en UX/UI zijn volledig vrij!

Met CSS kunnen we bv een header component boven in de pagina tonen, en menu component aan de linkerkant, de todo component centraal en daaronder de footer component. De interne structuur heeft aldus niets te maken met de zichtbare weergave.

Ontwikkelomgeving

- IDE zoals bv visual studio code
- NODE js installed (compileren en package manager)
- Angular CLI (command line interface) cli.angular.io
- Augury (Debug tool) augury.angular.io

Allemaal open source hulpmiddelen, dus behalve tijd zijn er geen kosten verbonden aan het ontwikkelen van angular apps!

Hello World in Angular

- Angular CLI installeren
- Project generatie
- Package.json, tsconfig.json en angular-cli.json
- Component schrijven en inladen in de index.html
- Een nieuw component schrijven en weergeven
- Typescript

Angular CLI installeren

npm install -g @angular/cli

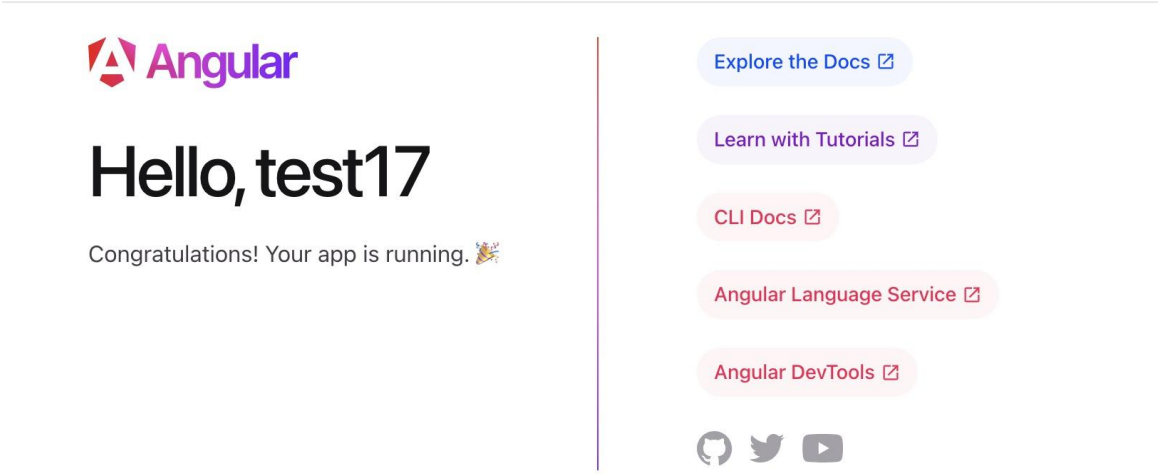
-g staat voor “global” zodoende de CLI permanent en in alle mappen/lagen beschikbaar is!

Project generatie (NO SSR - ServerSideRendering)

ng new angular-app
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? CSS

cd angular-app
ng serve --open

--open zorg voor het openen van de standaard browser. Zonder de --open flag moeten we de omgeving aanspreken via de http://localhost:4200 url.



Project openen en aanpassen

- Laat het terminalvenster gewoon open staan.
- Open de visual studio code en laad de projectfolder in
- Bestudeer de bestanden en structuur na de cli generatie
- Open het bestand src/app/app.component.html en verwijder de standaard code door:

```
<div style="text-align:center">
        <H1>Hello world</H1>
</div>
```

Zodra we het bestand opslaan, zal de browser in real-time de modificaties weergeven!

Foutieve HTML, div tagjes niet correct afgesloten bv, wordt afgestraft door een blanco pagina… De angular parser moet 100% geldige HTML hebben om deze weer te kunnen geven.

Mappenstructuur

We bespreken in kort de belangrijkste mappen dewelke de CLI heeft ingericht op onze machine:

e2e: Map met end to end test. Map niet noodzakelijk voor de werking van onze app
node_modules: Een map met alle afhankelijkheden van ons project, zoals de angular bibliotheken etc…Deze map wordt gevuld met de **npm install** opdracht. Niet in rommelen is de boodschap :)
src: De map met de bronbestanden, hier gaan we effectief onze app in schrijven!
- **app:** de map met de componenten en modules. Meest actieve map ever :)
- **assets:** de map met de zogenoemde statische bestanden (images, pdf, etc..)
- **environments:** Map waar we meerdere omgevingen in angular kunnen definiëren.

Belangrijk: Package.json, tsconfig.json en angular.json

Behalve de mappenstructuur heeft angular CLI ook een heel deel bestanden geschreven. Ze zijn uiteraard lang niet allemaal noodzakelijk, maar de volgende bestanden mag u niet verwijderen!

package.json (dependencies via npm)
tsconfig.json (Typescript = abstratielaag over js - config mbt de vertaalslag ES6 naar ES5)
angular.json (Angular CLI moet zelf weten hoe de structuur in elkaar steekt)

Een nieuw component genereren

We gaan een nieuw component aanmaken waarbij we al onze klanten kunnen oplijsten. Deze krijgt de naam CustomerComponent en gaan als volgt te werk:

```
ng g c customer
CREATE src/app/customer/customer.component.css (0 bytes)
```

CREATE src/app/customer/customer.component.html (23 bytes)
CREATE src/app/customer/customer.component.spec.ts (610 bytes)
CREATE src/app/customer/customer.component.ts (242 bytes)

Controleer als alle bestanden worden aangemaakt. Open het bestand customer.component.html. Dit is de view van de nieuwe component. Vervang de html door volgende inhoud:

```
<h2>Onze klanten</h2>
<ul>
        <li>Nasa</li>
        <li>SpaceX</li>
        <li>ESA</li>
        <li>Jaxa</li>
</ul>
```

Open de hoofdcomponent app.component.html
Plaats hier in de verwijzing naar het nieuwe component door zijn selector te gebruiken

<app-customer></app-customer>

Meer over componenten

Een component is de fundamentele bouwsteen van elke angular-app, Een component bevat een view en een Javascript klasse controller. Volgens de conventie moeten we elk component in haar eigen mapje plaatsen.

Open het bestand customer.component.ts.

```
import { Component } from '@angular/core';

@Component({
selector: 'app-customer',
standalone: true,
imports: [],
templateUrl: './customer.component.html',
styleUrl: './customer.component.css'
})
export class CustomerComponent {

}
```

Nieuw in ES6 en typescript? Dan zijn sleutelwoorden zoals import, export, @component en class waarschijnlijk nieuw voor u…

Een korte toelichting: Utsplitsing van de metadata:

- selector: app-customer Dit is de CSS-selector voor de component. Deze wordt gebruikt om het element in de HTML-template te identificeren dat wordt vervangen door de gegenereerde uitvoer van de component.
- standalone: true Dit geeft aan dat de component een standalone-component is. Dit betekent dat deze gebruikt kan worden zonder deel uit te maken van een NgModule.
- imports: [] Dit is een array van imports voor de component. Imports worden gebruikt om andere componenten, directives of modules te importeren die de component nodig heeft.
- templateUrl: './customer.component.html' Dit is de URL van de HTML-template van de component. De template definieert de HTML-inhoud van de component.
- styleUrl: './customer.component.css' Dit is de URL van de CSS-stijlsheet van de component. De stijlsheet definieert de CSS-stijlen voor de component.

Het export-sleutelwoord aan het einde van de code maakt de CustomerComponent-klasse beschikbaar voor import in andere modules.

Wat met de OPTIONELE modules?

We hebben nu 2 componenten en die worden keurig op het scherm getoond. Maar hoe weet Angular hoe dit moet gebeuren? Daarvoor moeten we nog een aantal extra onderdelen kennen, namelijk modules, de bootstrapper main.ts en het standaard webbestand index.html

- Een Angular app kan een module bevatten. (Sinds v17 -> Optioneel)
- Elke module kan nul, een of meer componenten bevatten;
- Modules zijn net als componenten, “js classes” welke met een TypeScript decorator van speciale functionaliteiten worden voorzien.
- de decorator voor een module is @ngModule();
- modules hebben zelf geen functionaliteit. Het zijn containers voor componenten & services

Tot slot de index.html

Hier is de enige opvallende verwijzing de <app-root></app-root>
Zijnde een selector van de component AppComponent. Verder zien we geen enkele verwijzing naar angular in de HTML. Hoe weet de browser dan dat het om een Angular applicatie gaat? Wel daarvoor hebben we de **ng serve**

Eigenaardigheid: klik op bron weergeven…

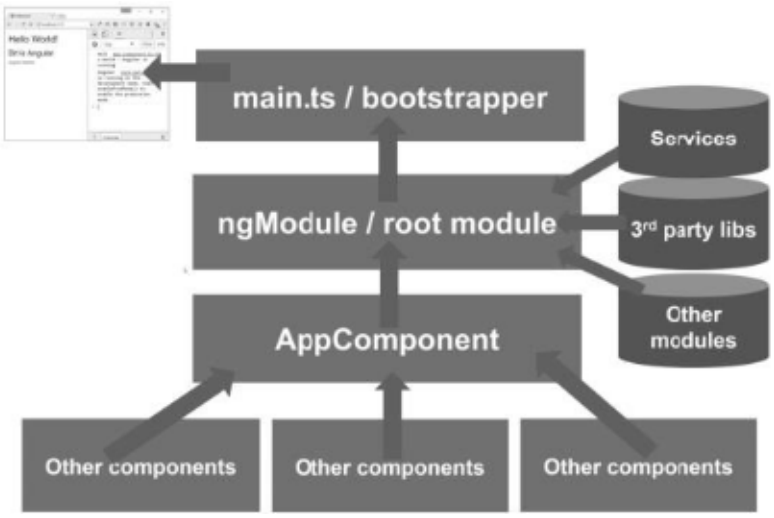
ng serve

Voert volgende taken uit:

- TypeScript compileren naar javascript
- Sass en Less compileren naar css (indien aanwezig)
- Webserver starten op localhost:4200
- Live reload starten van de browser
- Bundels samenstellen en serveren via webpack (automatisch)

Bootstrap 5 laden?
https://ng-bootstrap.github.io/#/home of preferred via de CDN
https://getbootstrap.com/docs/5.0/getting-started/introduction/

De architectuur van elke Angular applicatie:



Praktijkoefeningen:

Maak een eigen hello world app, gestyled met bootstrap en include een nieuwe component naar keuze!

Databinding en modellen

We hebben alvast 2 componenten gemaakt, maar ze doen nog niet veel meer dan statische HTML weergeven. Angular kent verschillende manieren van databinding. Eerst maken we kennis van éénvoudige databinding directives als *ngFor en *ngIf (@for @if
). Zo komt onze applicatie eindelijk tot leven.

- Eenvoudige databinding met {{ … }}
- De directives *ngFor en *ngIf maar ook sinds v17 @for @if
- Een datamodel maken en gebruiken
- Werken met class

Wat is databinding?

Databinding is het proces van gegevens binden aan placeholders in een HTML sjabloon. Data kan op verschillende plekken staan en op diverse manieren worden getoond.

Opgeslagen in de applicatie: Denk bijvoorbeeld aan taalselectie
Opgeslagen in externe systemen: Via een API data uit een DB halen

Javascript is van oudsher niet erg geschikt voor databinding. Zelf met jQuery was databinding vaak een moeizaam proces. In de loop der jaren zijn er wel enkele libs ontwikkeld om dit te vergemakkelijken. Knockout.js en Backbone bv…

Declaratieve syntax

Data toekennen aan plaatshouders in onze HTML. In tal van bib's en ook in Angular worden de dubbele accolades {{ … }} gebruikt voor éénvoudige databinding.

```
//Voorbeeld
<ul>
<li>{{ product.id }} <b> {{ product.name }} </b></li>
</ul>
```

Dit levert veel leesbare code op dan bv met jQuery het geval is. We moeten hier uitvoerig bij stilstaan. Data behoort nl. tot de kern van onze applicatie.

Eenvoudige databinding met {{ … }}

We illustreren dit met een eenvoudig voorbeeld.
Onze app.components.ts component krijg 2 eigenschappen of properties: title en name.
Ze worden gedefinieerd in de klasse AppComponent.

```ts
// app.component.ts
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterOutlet } from '@angular/router';

@Component({
selector: 'app-root',
standalone: true,
imports: [CommonModule, RouterOutlet],
templateUrl: './app.component.html',
styleUrl: './app.component.css'
})
export class AppComponent {
title = 'Nasa fake client database';
name = 'Massimo';
}
```

```
<!-- app.component.html -->

<h1>{{ title }}</h1>

<p>{{ name }}</p>
```

Nasa fake client database

Welkom, Massimo

Onze klanten

- Nasa
- SpaceX
- ESA
- Jaxa

TypeScript benutten

Een krachtige eigenschap van TypeScript is dat we het type van een variabele kunnen aangeven. (vandaar ook de naam :) In dit geval 2x string! De Typescript compiler zal een foutmelding geven in geval van onjuistheden, maar voorkomt de weergave niet.

```
title:string = 'Nasa fake client database';
name:string = 'Massimo';
aantal:number = 12;
```

https://www.typescriptlang.org/docs/handbook/basic-types.html

Geen output bij fouten? Zet dan de noEmitOnError op true in de tsconfig.json

Databinding in de constructor

Een andere mogelijkheid is om gegevens in een competent in de constructor van een klasse te initialiseren. Dit ziet er zo uit

```
// app.component.ts
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterOutlet } from '@angular/router';

@Component({
```