

3AON WebExpert



Intro OLOD WebExpert

- Lectoren
 - Dries Swinnen
- Mail
 - Onderwerp: [WebExpert]
 - Aansprekking & slot

Intro OLOD WebExpert

- Cursusmateriaal
 - Blackboard.pxl.be
 - Slides
 - Labs
 - Pluralsight
 - Angular courses
 - NodeJS
 - Github classroom
 - Oefeningen

Intro OLOD WebExpert

- 3 studiepunten
- 28 uren practicum
 - 2u per week
- 56 uren zelfstudie

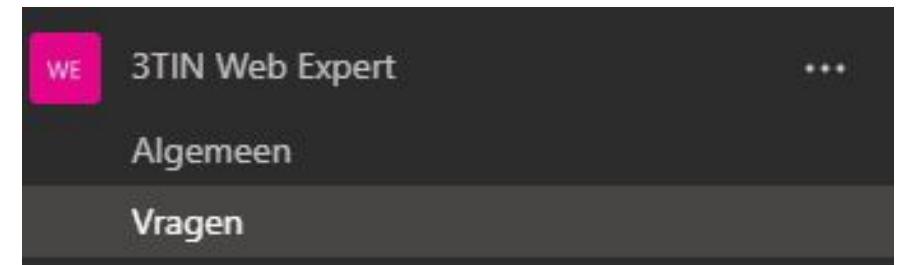
Examen (100%)

- Schriftelijk open boek laptop

Permanente evaluatie (0%)

Afstandsonderwijs

- MS Teams
 - Contactmomenten / Lesmomenten
 - Meeting op het moment van de les in het kanaal “Algemeen”
 - Communicatie tussen Studenten & docenten tijdens de lessen
 - Blackboard
 - aanbieden van lesmateriaal
 - Mededelingen
- Github (classroom)
 - opvolging oefeningen / code



Benodigdheden afstandsonderwijs

- MS Teams Desktop app
 - Zorg voor een headset / oortjes!
- Git & Github account
 - Github education for extra goodies!

<https://education.github.com/pack#offers>

Leerinhouden

- Angular
- NodeJs
- CSS-preprocessors (SASS & LESS)



Leerinhouden

Lesweek	Onderwerp
Week 1 & 2	Introductie + opzetten omgeving + inleiding Angular - Typescript - Components
Week 3 & 4	Communicatie tussen components – Angular directives & databinding
Week 5 & 6	Form validatie & functions – Services
Week 7 & 8	Data ophalen met http - pipes
Week 9 & 10	Routing - Routeguards & resolvers - Kennismaking nodeJS
Week 11 & 12	NodeJS modules & packages - WebApps met Express
Week 13 & 14	MongoDB & Mongoose – CSS Preprocessors

Lesmateriaal

(2020-21 - PBTIN)

ALGEMEEN

- Studiewijzer
- Afstandsonderwijs
- Mededelingen
- Studiegidsinfo

Overzicht

- Leerpad

Leerinhouden

Angular

- Inleiding - Typescript - Components (week 1 & 2)
- Data- en eventbinding - directives (week 3 & 4)
- Forms - Services - Dependency injection (week 5 & 6)

Data- en eventbinding - directives (week 3 & 4) ↗ CH5 Attribute directives & Property bindings ↗

Inhoud bouwen ▾ Beoordelingen ▾ Tools ▾ Inhoud van Inhoud bouwen ▾ Beoordelingen ▾ Tools ▾ Inhoud van partner ▾

 CH4 Databinding & Eventbinding ↗	 CH5 Attribute directives & Property bindings - Slides ↗
 CH5 Attribute directives & Property bindings ↗	 CH5 Attribute directives & Property bindings - Lab ↗
 CH4 5 Data & eventbinding - directives - Oefeningen ↗	
 Resources ↗	

[Overeenstemmingen](#)[week 1 - Angular](#) [week 2 - Angular](#) [week 3 - Angular](#) [week 4 - Angular](#) [week 5 - Angular](#) [week 6 - Angular](#) [week 7 - Angular](#) [week 8 - Angular](#)[week 9 & 10 - Angular](#)[week 11 - NodeJS](#)[week 12 - NodeJS](#)[week 13 - NodeJS](#)[week 14 - SaSS](#)[What's next?](#)[week 1 - Angular](#)

H0 Intro web expert

Dit hoofdstuk geeft een algemeen beeld van het vak Web Expert.

slides[Intro web expert](#)**studiewijzer**[Moduleboek](#)**studiewijzer**[ECTS Fiche](#)**studiewijzer**[Examenrichtlijnen](#)

H1 Inleiding Angular

In dit hoofdstuk krijg je een algemeen beeld van het Angular framework.

slides[Inleiding Angular](#)**pluralsight**[Angular Getting started \(D. Kurata\)](#)**pluralsight**[Introduction \(J. Cooper\)](#)**extern**[Angular getting started guide](#)[Top](#)

Lesmateriaal



3AON-WebExpert-2021

Accept the assignment —
CH2_3 oefeningen

Once you accept this assignment, you will be granted access to the
`ch2-3-oefeningen-dries` repository in the [3AON-WebExpert](#)
organization on GitHub.

[Accept this assignment](#)

[master](#) 2 branches 0 tags[Go to file](#)[Add file](#) [Code](#)

About

ch2-3-oefeningen-d-ries created by
GitHub Classroom[Readme](#)

Releases

No releases published
[Create a new release](#)

Packages

No packages published
[Publish your first package](#)

Contributors 2



niekvandael



d-ries Dries Swinnen

d-ries aanpassing footer app component

047fc6d 18 seconds ago 4 commits

.github

GitHub Classroom Feedback

1 minute ago

CH3-oefening-debug

aanpassing footer app component

18 seconds ago

README.md

Initial commit

1 minute ago

README.md

Oefeningen: Components

Oefening 1

Maak zelf een nieuw project in de root directory van deze repository met als naam "chapter3-oefening1" via de Angular CLI. Voorzie volgende componenten:

1. Maak een logoComponent. Voorzie een div en zorg ervoor dat de width overeenkomt met de schermbreedte en dat de hoogte overeenkomt met 140px. Zorg ervoor dat in deze component een logo naar keuze getoond wordt met een groene achtergrond. De HTML en CSS staat in de file `logo.component.ts`
2. Maak een nieuwsbriefComponent aan. Hierin heb je één input veld met label en een submit knop. Het label krijgt de waarde "Schrijf je in voor onze nieuwsbrief...". Er hoeft geen verwerking te zitten achter de knop / input

[3AON-WebExpert / ch2-3-oefeningen-d-ries](#) Private

generated from 3AON-WebExpert/CH2_3_Oefeningen

[Watch](#) 0 [Star](#) 0 [Fork](#) 0

[Code](#) [Issues](#) [Pull requests 1](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

[Filters](#) [Labels 9](#) [Milestones 0](#) [New pull request](#)

[1 Open](#) [0 Closed](#)

[Author](#) [Label](#) [Projects](#) [Milestones](#) [Reviews](#) [Assignee](#) [Sort](#)

[Feedback](#)
#1 opened 2 minutes ago by niekvandael

💡 ProTip! Ears burning? Get @d-ries mentions with [mentions:d-ries](#).



d-ries reviewed now

[View changes](#)

d-ries left a comment



Nog enkele zaken niet correct opgenomen

CH3-oefening-debug/src/app/app.component.html

```
... ... @@ -5,5 +5,5 @@
 5   5      <app-contact></app-contact>
 6   6      </main>
 7   7      <footer>
 8 - Copyright de inspiratie is op 2018.
```

d-ries now

Aanpassing Niet oké!



d-ries now

Reply...

[Resolve conversation](#)

CH3-oefening-debug/src/app/app.component.html

```
... ... @@ -5,5 +5,5 @@
 5   5      <app-contact></app-contact>
```

d-ries now

Hier mist nog een stuk uit de opgave

d-ries now

Reply...

[Resolve conversation](#)

Kennismaking Angular

Angular



Inhoud

- Wat is Angular
- Angular Structuur / Concepten
- Voorkennis
- Opzetten development omgeving
- Eerste Angular app

Wat is Angular

- **Algemeen**

- Web uitgegroeid, sinds de jaren 90, tot complexe systemen

- **Angular**

- Framework voor het programmeren van complexe web apps
- Concepten die het mogelijk maken om code en structuur van elkaar te scheiden, modulair programmeren en testbare applicaties maken

Wat is Angular

- **Algemeen**

- HTML is een taal, oorspronkelijk ontwikkeld om statische webpagina's te ontwikkelen
- JavaScript & CSS is toegevoegd om dynamische webapplicaties te behandelen
- Javascript was in het begin moeilijk te leren
- De verschillende browsers hadden een verschillende implementatie van JavaScript

Wat is Angular

- **Libraries en Frameworks**

- JavaScript is pas echt doorgebroken sinds 2006
- Er zijn tal van andere bibliotheken ontwikkeld
(DOM-manipulatie, routing, data binding, ...)
- Angular is geen library, maar een compleet framework voor het realiseren van clientside WebApps

Wat is Angular

- Frontend/Client side javascript framework
- Cross platform
- Ontwikkeld door Google
- <http://angular.io>
 - Getting started
 - Documentation

AngularJS <> Angular

Wat is Angular

- Versies
 - AngularJS (a.k.a. Angular 1.x): <http://angularjs.org>
 - **Angular 2: Complete rewrite van AngularJS:** <http://angular.io>
 - Angular 3: overgeslagen
 - Angular 4: Backward compatible met angular 2 (Released: 04-2017)
 - Angular 5 (Released: 01-11-2017)
 - Angular 6 (Released: 04-05-2018)
 - Angular 7 (Released: 18-10-2018)
 - ...
 - Angular 10: Wijzigingen zie [blog](#) (Released: 25-06-2020)

Wat is Angular

- Abstractielag tussen browser, logica van de app en data (lijkt op MVC)
- Variant op MVC
 - MV*
- Front-end applicaties
- Gebruik van externe libraries (bv. Nativescript) mogelijk
- Typescript

<https://www.madewithangular.com/>

Wat is TypeScript

- Superset van JavaScript
- Ontwikkeld door Microsoft
- Ondersteuning van DataTypes
- Ondersteuning van OOP
- Compileert naar JS
 - JS code kan gebruikt worden in TS code

Angular opbouw

- Modules
- Components
- Databinding
- Services

Angular Opbouw: Modules

- Declareren en laden van gebruikte elementen zoals:
 - (Zelfgemaakte) Components
 - Gebruikte modules
 - Service providers
 - ...
- Bootstrappen van de App

Angular Opbouw: Components

- Combinatie van View en stuk Controller
 - View
 - Html/CSS met Angular directives (zie verder)
 - CSS heeft aparte scopes
 - Controller
 - Typescript klasse met logica
 - Directives binnen de View
 - ngIf, ngFor, ...

Angular Opbouw: Databinding

- Oneway binding (best practise)
- 2way binding
- Curly braces {{..}}
- (Custom) Pipes

```
<ul>
  <li>Email: {{contact.email}}</li>
  <li>Phone: {{contact.phone}}</li>
</ul>
```

Angular Opbouw: Services

- Aanbieden van data
- Aanspreken van API's

Angular Opbouw: Directives

- Angular logica in de view
- Link leggen met de component logica
- NgIf, NgFor, ngModel, ngClass, ...

Angular Opbouw: Dependency Injection

- Modules/Services hergebruiken in verschillende plaatsen van je App
- Injecteren van deze module/services
- Code wordt flexibel en moet niet geduplicateerd worden

Opzetten development omgeving



NodeJS & NPM

- NodeJS
 - Javascript runtime environment
 - Server-side
 - Async
- NPM
 - Javascript package manager
 - Default in NodeJS



HOME | ABOUT | DOWNLOADS | DOCS | GET INVOLVED | SECURITY | CERTIFICATION | NEWS

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

#BlackLivesMatter

New security releases are available

Download for Windows (x64)

12.18.4 LTS

Recommended For Most Users

14.11.0 Current

Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#) [Other Downloads](#) | [Changelog](#) | [API Docs](#)

Or have a look at the [Long Term Support \(LTS\) schedule](#).

NodeJS & NPM

- Controle of Node & NPM geïnstalleerd is:

```
C:\Users\Dries>node -v  
v12.18.3
```

```
C:\Users\Dries>npm -v  
6.14.6
```

```
C:\Users\Dries>
```

Angular 4

- Installatie van NPM package manager in command prompt:

```
C:\Users\Dries> npm install -g @angular/cli
```

- Controle van installatie:

```
C:\Users\Dries> ng --version
```

```
C:\Users\Dries>ng --version
```

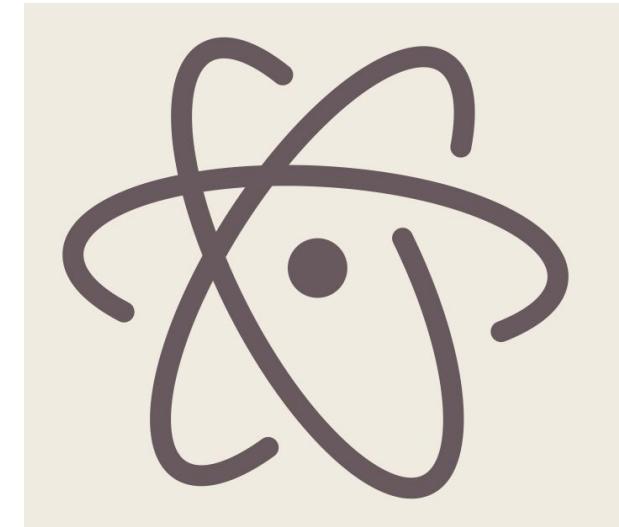
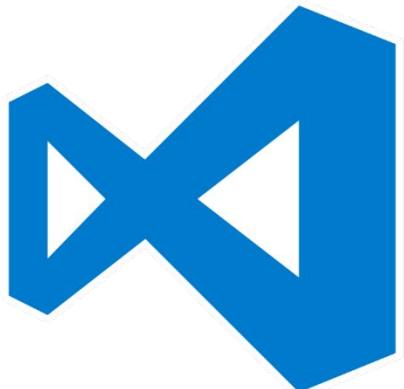
```
Angular CLI: 10.0.8  
Node: 12.18.3  
OS: win32 x64
```

```
Angular:  
...  
Ivy Workspace:
```

Package	Version
<hr/>	
@angular-devkit/architect	0.1000.8
@angular-devkit/core	10.0.8
@angular-devkit/schematics	10.0.8
@schematics/angular	10.0.8
@schematics/update	0.1000.8
rxjs	6.5.5

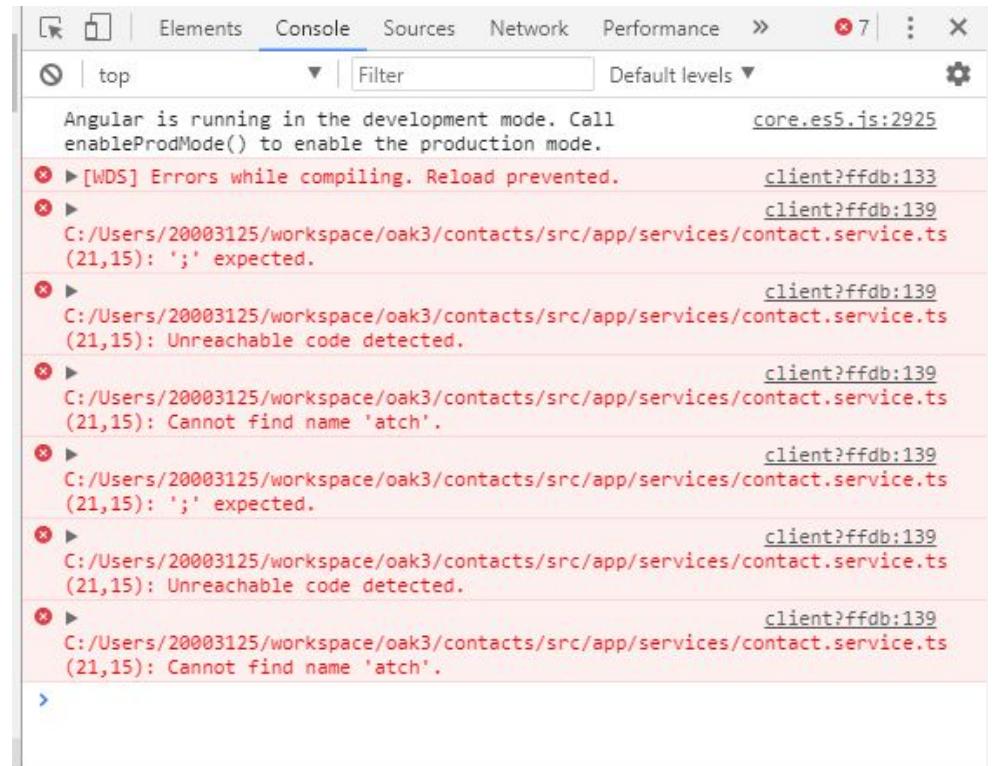
IDE

- Visual Studio Code (Microsoft)
- Webstorm (IntelliJ)
- Atom (GitHub)



Debugging

- Development console (F12 in browser)
- Commandprompt/Terminal
- IDE
- 3rd party plugins



The screenshot shows the VS Code terminal tab titled '1: node'. It displays the following output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: node + - x
trial]
chunk {styles} styles.bundle.js, styles.bundle.js.map (styles) 12.3 kB [inline] [initial]
chunk {vendor} vendor.bundle.js, vendor.bundle.js.map (vendor) 2.91 MB [initial] [rendered]

ERROR in C:/Users/20003125/workspace/oak3/contacts/src/app/services/contact.service.ts (21,15):
  ';' expected.
ERROR in C:/Users/20003125/workspace/oak3/contacts/src/app/services/contact.service.ts (21,15):
  Unreachable code detected.
ERROR in C:/Users/20003125/workspace/oak3/contacts/src/app/services/contact.service.ts (21,15):
```

At the bottom of the terminal, status information is shown: Ln 21, Col 17 Spaces: 4 UTF-8 CRLF TypeScript 2.5.2 😊

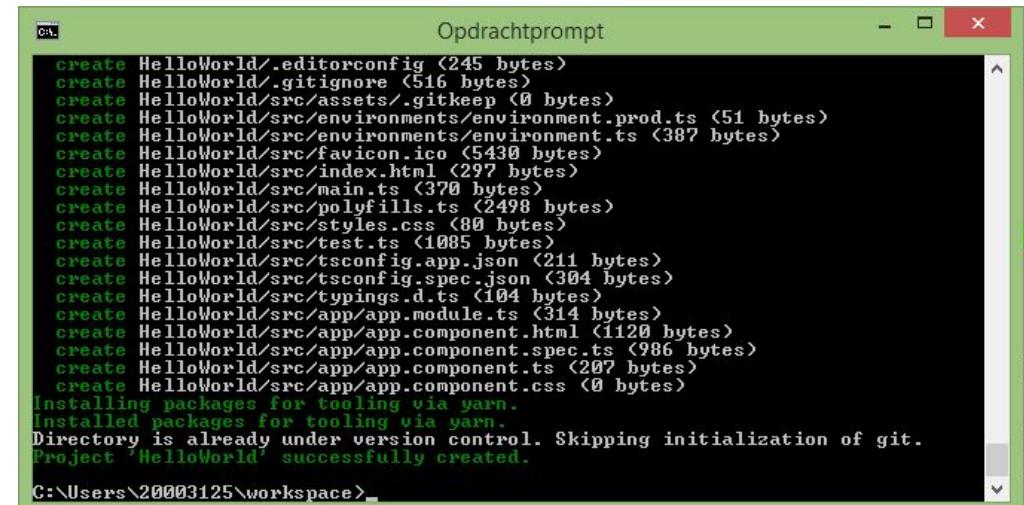
Eerste Angular app



Eerste Angular App

- Aanmaken via commandprompt
- Gebruik maken van de angularCLI
 - `ng help`
 - `ng new <projectnaam>`

```
C:\Users\Dries\workspace> ng new HelloWorld
```



The screenshot shows a Windows Command Prompt window with the title 'Opdrachtprompt'. The command `ng new HelloWorld` has been run, and the output shows the creation of various files and folders within the `HelloWorld` directory. The output includes:

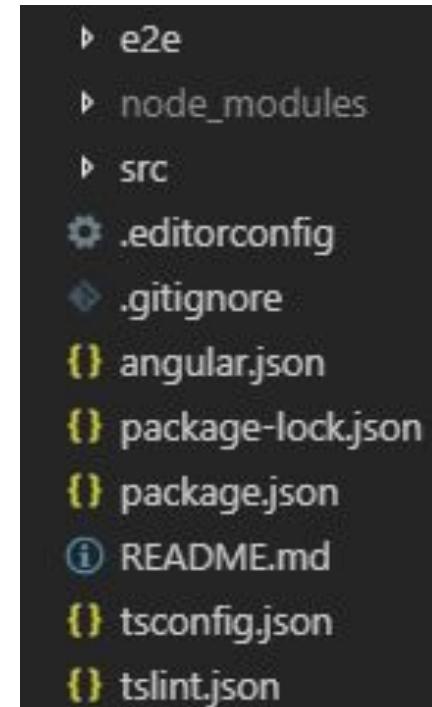
```
create HelloWorld/.editorconfig (245 bytes)
create HelloWorld/.gitignore (516 bytes)
create HelloWorld/src/assets/.gitkeep (0 bytes)
create HelloWorld/src/environments/environment.prod.ts (51 bytes)
create HelloWorld/src/favicon.ico (5430 bytes)
create HelloWorld/src/index.html (297 bytes)
create HelloWorld/src/main.ts (370 bytes)
create HelloWorld/src/polyfills.ts (2498 bytes)
create HelloWorld/src/styles.css (80 bytes)
create HelloWorld/src/test.ts (1005 bytes)
create HelloWorld/src/tsconfig.app.json (211 bytes)
create HelloWorld/src/tsconfig.spec.json (304 bytes)
create HelloWorld/src/typings.d.ts (104 bytes)
create HelloWorld/src/app/app.module.ts (314 bytes)
create HelloWorld/src/app/app.component.html (1120 bytes)
create HelloWorld/src/app/app.component.spec.ts (986 bytes)
create HelloWorld/src/app/app.component.ts (207 bytes)
create HelloWorld/src/app/app.component.css (0 bytes)
Installing packages for tooling via yarn.
Installed packages for tooling via yarn.
Directory is already under version control. Skipping initialization of git.
Project 'HelloWorld' successfully created.
```

At the bottom of the window, the path `C:\Users\20003125\workspace>` is visible.

- Voeg geen routing toe en selecteer “css” als opmaak

Eerste Angular App

- Project folder openen in IDE
- Verschillende zaken worden aangemaakt:
 - Node_modules
 - Src/index.html
 - Src/app
 - Src/app/app.module.ts
 - Src/app/app.component.ts
 - Package.json



Eerste Angular App

- Node_modules
 - Bevat modules & 3rd party libraries / dependencies die nodig zijn voor de werking van de app
 - Link wordt gelegd in de **package.json** file
 - Lijst van alle dependencies die nodig zijn
 - Zie slides nodejs
- Src/index.html
 - Eerste pagina die geladen wordt
 - Hierin wordt de component <app-root> geplaatst. Deze is gelinkt aan de app.component.ts

Eerste Angular App

- Src/app
 - De working directory van het project. Hierin worden zaken zoals components, services, ... aangemaakt.
- Src/app/app.module.ts
 - Bevat alle declarations , imports en providers die gebruikt worden binnen het project.
 - **Bij aanmaken van nieuwe components, services, ... altijd toevoegen aan app.module.ts !!**
- Src/app/app.component.ts
 - De default component die geladen wordt.

Eerste Angular App

- **App.component.ts**

Verwijst naar <app-root> in index.html

Verwijst naar de **HTML** opmaak van een component (zie file: app.component.html).

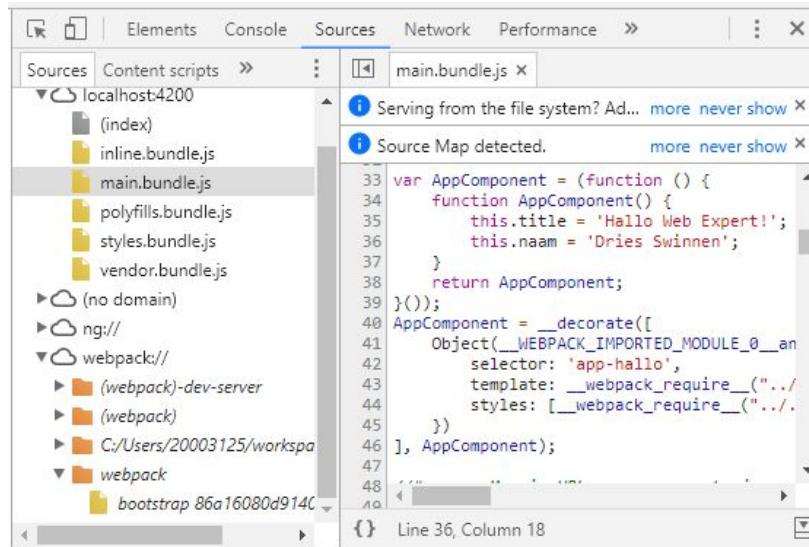
Verwijst naar de **CSS** opmaak van een component. CSS in dit bestand is enkel van toepassing op deze component (zie file: app.component.css).

Iedere component heeft een klasse waarin klassevariabelen, methodes, ... gedefinieerd kunnen worden.

```
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-root',
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9   title = 'app';
10 }
11 }
```

Eerste Angular App

- Webpack
 - Module bundler
 - Opgeroepen na het opslaan van files tijdens development, voor het laden van de webserver
 - Leest code en bundelt imports & code in één of meerdere JavaScript files



The screenshot shows the Chrome DevTools Sources tab. The left sidebar lists files: (index), inline.bundle.js, main.bundle.js (which is selected), polyfills.bundle.js, styles.bundle.js, and vendor.bundle.js. The right pane displays the code for the AppComponent. The code defines the AppComponent as a function that returns the component definition. It includes a title and a name, and uses __decorate to add a selector and template.

```
var AppComponent = (function () {
  function AppComponent() {
    this.title = 'Hallo Web Expert!';
    this.naam = 'Dries Swinnen';
  }
  return AppComponent;
}());
AppComponent = __decorate([
  Object(__WEBPACK_IMPORTED_MODULE_0__an
    selector: 'app-hallo',
    template: __webpack_require__(../../
      styles: [__webpack_require__(../../
        })
    ],
    AppComponent);
  
```

Eerste Angular App

- Je eerste app openen in de browser via de Angular CLI of via NodeJS

```
C:\Users\Dries\workspace\HelloWorld> ng serve -o
```

- De optie `-o` opent de app meteen in de browser
- Aanpassingen aan de app worden automatisch gecompileerd en de browser wordt automatisch vernieuwd.

Eerste Angular App

The screenshot shows the Angular CLI's welcome screen. At the top, there's a blue header with the Angular logo and the word "Welcome". On the right side of the header is a Twitter icon. Below the header, a red circular icon contains a white rocket ship, with the text "testproject489 app is running!" to its right. The main content area has a light gray background with a large white cloud shape on the left. It features a section titled "Resources" with three cards: "Learn Angular", "CLI Documentation", and "Angular Blog". Below this is a section titled "Next Steps" with five buttons: "New Component", "Angular Material", "Add Dependency", "Run and Watch Tests", and "Build for Production". A black terminal window at the bottom displays the command "ng generate component xyz". At the very bottom, there's a row of small circular icons representing various Angular tools or services, followed by the text "Love Angular? Give our repo a star." and a "Star" button.

A

Welcome

testproject489 app is running!

Resources

Here are some links to help you get started:

Learn Angular >

CLI Documentation >

Angular Blog >

Next Steps

What do you want to do next with your app?

+ New Component

+ Angular Material

+ Add Dependency

+ Run and Watch Tests

+ Build for Production

...

ng generate component xyz

Love Angular? Give our repo a star. ★ Star >

Eerste Angular App: Opdrachten

- Pas de selector van de App.component.ts aan naar “app-hallo”. Zorg ervoor dat de applicatie nu werkt met deze selector.
- Verwijder het Angular logo van de app en pas de titel variabele aan naar “Hallo Web expert!”.
- Voeg een klassevariabele “naam” met als inhoud je volledige naam toe aan de appComponent. Zorg ervoor dat deze getoond wordt op de webpagina in een h2 tag.
- Voorzie css opmaak zodat de h1 van deze component een blauwe kleur krijgt. De h2 van deze component krijgt een rode kleur.
- Voeg aan index.html een h1 tag toe met als inhoud “dit is een test” net boven de <app-hallo> selector. Welke kleur krijgt deze header? Waarom?
- Zorg ervoor dat de opmaak van de h1 tag toegepast wordt op de hele app.

TypeScript

Angular



Introductie TypeScript

- Ontwikkeld door Microsoft
- Javascript-achtige syntax
- Voorzien van volledig type support
- Voorzien van volledig OOP support
- Compiled naar JavaScript

=> Wordt gebruikt doorheen Angular vanaf Angular 2

TypeScript Syntax

- Uitbreiding op bestaande JavaScript
- Datatypes en object ondersteuning toegevoegd
- Eenvoudig om te leren
- Kan legacy JavaScript code bevatten
- Kan gebruikt worden buiten Angular

Definitie van variabelen

- Standaard Javascript ondersteunt maar één variabele type:

```
let x = 'blabla'; // String data
let y = false; // boolean data
let z = 33; // numeric data
let a = 'whatever'; // any data
```

- TypeScript ondersteunt verschillende variabele types:

```
let x: string = 'blabla'; // String data
let y: boolean = false; // boolean data
let z: number = 33; // numeric data
let a: any = 'whatever'; // any data
```

Definitie van arrays

- Standaard in JavaScript:

```
let kleuren = ['rood', 'blauw', 'zwart', 'groen'];
let getallen = {1,2,3,4,20};
let personen = [{naam: 'Dries'}, {naam: 'Bob'}];
```

- In TypeScript

```
let kleuren: string[] = ['rood', 'blauw', 'zwart', 'groen'];
let getallen: number[] = {1,2,3,4,20};
let personen: Object[] = [{naam: 'Dries'}, {naam: 'Bob'}];
```

Definitie van arrays

- Datatypes bij arrays zorgen ervoor dat enkel objecten van dat datatype toegevoegd kunnen worden:
 - Enkel getallen kunnen toegevoegd worden:

```
let getallen: number[] = {10,20,30}
```

- Verschillende types van objecten kunnen toegevoegd worden:

```
let namen: Object[];
```

- Enkel Objecten van het type Person kunnen toegevoegd worden:

```
let people: Person[];
```

Classes en Objecten

- Definitie van een klasse met properties in TypeScript:

```
class Cat{  
    naam:string;  
    type:string;  
}
```

- Aanmaken van objecten van het type Cat:

```
let cats: Cat[];  
let cat1: Cat = {name: "Luna", type:"European Shorthair"};  
let cat2: Cat = {name: "Kiara", type:"Maine Coon"};  
  
cats[0] = cat1;  
cats[1] = cat2;
```

Classes en Objecten

- Gebruik van constructors:

```
class Cat{  
    naam:string;  
    type:string;  
  
    constructor(naam: string, type:string){  
        this.naam = naam;  
        this.type = type;  
    }  
}  
  
let cat1 = new Cat("Luna", "European Shorthair");  
let cat2 = new Cat("Kiara", "Maine Coon");  
  
let cats: Cat[] = [cat1, cat2];
```

Constructors alternatief

- Argumenten in de constructor worden automatisch toegevoegd als klasse properties:
 - Scope moet meegegeven worden (private / public)
 - Wordt vaak gebruikt bij dependency injection (zie later)

```
class Building{  
    constructor(private address: string, private units:  
number){}  
  
let bld1 = new Building("1 main street", 4);
```

Interfaces

- Interfaces kunnen gebruikt worden als “template” voor een klasse.
- Eén van de meest gebruikte interfaces bij Angular is “OnInit”

```
interface Itrip{  
    destination: string;  
    days: number;  
    display();  
}  
  
class BizTrip implements Itrip{  
    constructor(private destination: string, private days: number){}  
  
    display(){  
        console.log(this.destination + ", " + this.days);  
    }  
}
```

Functies & methodes

- Bij functies met een return value wordt het datatype meegegeven in de declaratie van de functie:

```
function getString(): string {  
    let str: string = "My string";  
    return str  
}
```

- Meegeven van argumenten binnen een functie zonder return value:

```
function logMessage(message: string) {  
    console.log(message);  
}  
  
logMessage(getString());
```

Functies & methodes

- Het gebruik van een array als parameter van een functie:

```
let strings:string[] = ["abc","def","ghi"];  
  
function displayStr(string_array: string[]){  
    for(let idx in string_array){  
        console.log(string_array[idx]);  
    }  
}  
  
displayStr(strings);
```

Functies en methodes

- Bij methodes binnen een klasse gebruik je **geen** keyword “function”:

```
class BizTrip implements Itrip{
    constructor(private destination: string, private days: number){}

    display(){
        console.log(this.destination + ", " + this.days);
    }
}
```

Werken met modules

- Voor het gebruik van verschillende .ts bestanden moeten classes geëxporteerd worden om ze te kunnen gebruiken.
- Bijvoorbeeld:
 - person.ts = Bestand met de klasse Person
 - app.ts = de logica van een applicatie waarin de klasse Person gebruikt moet worden.

Werken met modules

- Gebruik van het keyword **export** in person.ts
- **Import** toevoegen aan app.ts

```
//person.ts
export class Person {
    constructor(private fname, private lname){}
    display(){console.log(fname + " " + lname);
}

//app.ts
import {Person} from './person';

let p1 = new Person("John", "Doe");
p1.display();
```

Var, let & const

- const variabelen:

- Variabelen waarvan de waarden slechts één maal gedefinieerd wordt
- Best practise : uppercase declaratie

```
for(let i=0; i < 5;i++){  
    ...  
}  
  
if(a === b){  
    let x = 5;  
}  
  
const PI: number = 3.14159;  
const URL: string = "http://pxl.be/products";
```

Arrow functies

- “Arrow functies” wordt gebruikt als verkorte manier voor het definiëren van functies
- Linkse gedeelte van de pijl = de parameters van de functie
- Rechtse gedeelte van de pijl: implementatie van de functie
- Geen return keyword nodig

```
let adder1 = function(a,b){ return a + b; } //klassieke methode
let adder2 = (a,b)=>a+b; //arrow functie

console.log(adder1(1,2));
console.log(adder2(1,2));

let data1 = function(result) { return result.data; }
let data2 = result=>result.data;
```

Template strings

- Nieuwe syntax beschikbaar in ES6 & TypeScript
- Gebruik van back-ticks (`) om string literals te definiëren
- Expressies en variabelen inline vervangen
- Multiline ondersteuning

```
let name = `Dries`;
let x = `Hallo ${name}`;

let y = `${name} heeft ${2 * 3} euro
        gewonnen!`;

function getName(){return "Joske";}
let z = `Goedemorgen ${getName()}`;
```

Components

Angular



Wat is een component

- Modules / custom objecten
- Implementatie van views met behulp van:
 - HTML templates
 - CSS stijlen
- Koppeling tussen view elementen en code

Wat is een component

- Components kunnen:
 - Andere components gebruiken
 - Code bevatten
 - Directives gebruiken
 - Services gebruiken

Wat is een component

- De voorbeeldcomponent heeft:
 - 2 input velden
 - Een output bericht
 - Een submit knop
- Om een component te gebruiken gebruiken we custom HTML tags:

```
<body>
<app-login>Loading...</app-login>
</body>
```

Inloggen

Naam:

Wachtwoord:

Welkom Dries!

Wat is een component

- Componenten in een groter geheel

The screenshot shows a web application interface titled "Book Collection". At the top, there is a purple header bar with the title "Book Collection". Below it, a search bar contains the text "angular". The main content area is titled "Find a Book" and displays a list of six book cards:

- Ng-Book 2**: "The Complete Book on Angular 2". Cover image shows a red book with "Ng-Book 2" on the cover. Description: "Ready to master Angular 2? What if you could master the entire framework - with solid foundations - in less time without beating your head against a wall? Imagine how quickly you could work if you knew the best practices and the best tools? Stop wastin..."
Written By: Nate Murray, Ari Lerner, Felipe Coury, and Carlos Taborda
- AngularJS**: "Develop smaller, lighter web apps that are simple to create and easy to test, extend, and maintain as they grow. This hands-on guide introduces you to AngularJS, the open source JavaScript framework that uses Model-view-controller (MVC) architecture,..."
Written By: Brad Green and Shyam Seshadri
- Professional AngularJS**: "A comprehensive guide to AngularJS, Google's open-source client-side framework for app development. Most of the existing guides to AngularJS struggle to provide simple and understandable explanations for more advanced concepts. As a result, some deve..."
Written By: Valeri Karpov and Diego Netto
- Mastering Web Application Development with AngularJS**: "The book will be a step-by-step guide showing the readers how to build a web application using AngularJS. It covers the basics of AngularJS and provides examples of how to use it in real-world scenarios. The book also covers advanced topics such as routing, services, and directives. By the end of the book, readers will have a solid understanding of AngularJS and be able to build complex web applications."
Written By: [unclear]
- Eloquent JavaScript**: "A Modern Introduction to Programming". Cover image shows a yellow book with "Eloquent JavaScript" on the cover. Description: "Provides information and examples on writing JavaScript code, covering such topics as syntax, control, data, regular expressions, and functions. It also covers object-oriented programming and functional programming, as well as the Node.js environment and the browser API. By the end of the book, readers will have a solid understanding of the language and be able to write effective and efficient JavaScript code."
Written By: [unclear]
- Pro AngularJS**: "AngularJS is the leading framework for building dynamic JavaScript applications that take advantage of the capabilities of modern browsers and devices. AngularJS, which is..."
Written By: [unclear]

Wat is een component

- Componenten in een groter geheel

The screenshot shows a web application interface. At the top is a purple header bar with the text "Book Collection". Below it is a white main area with a title "Find a Book" and a search input field containing "angular". The main content area displays a grid of book cards:

- Ng-Book 2**: "The Complete Book on Angular 2". Description: "Ready to master Angular 2? What if you could master the entire framework - with solid foundations - in less time without beating your head against a wall? Imagine how quickly you could work if you knew the best practices and the best tools? Stop wastin...".
- AngularJS**: "Develop smaller, lighter web apps that are simple to create and easy to test, extend, and maintain as they grow. This hands-on guide introduces you to AngularJS, the open source JavaScript framework that uses Model-view-controller (MVC) architecture,...".
- Professional AngularJS**: "A comprehensive guide to AngularJS, Google's open-source client-side framework for app development. Most of the existing guides to AngularJS struggle to provide simple and understandable explanations for more advanced concepts. As a result, some deve...".
- Mastering Web Application Development with AngularJS**: "The book will be a step-by-step guide showing the readers how to build web applications with AngularJS".
- Eloquent JavaScript**: "A Modern Introduction to Programming". Description: "Provides information and examples on writing JavaScript code, covering such topics as syntax, control, data, regular...
- Pro AngularJS**: "AngularJS is the leading framework for building dynamic JavaScript applications that take advantage of the capabilities of modern browsers and devices. AngularJS, which is...

Nav component

Search component

Booklist component

- > BookItem Component
- > BookItem Component
- > BookItem Component
- ...

Opbouw van een component

- Componenten hebben 3 secties:
 - Imports:
 - altijd minstens één import aanwezig.
 - Hier worden er toegevoegd als andere components / klasses / services gebruikt worden.
 - Annotatie: @Component
 - Geassocieerd met de klasse die eronder staat
 - Data in de annotatie wordt gebruikt om de view op te bouwen
 - JSON formaat
 - Component klasse
 - Export keyword zodat deze ook geimporteerd kan worden in andere components

Opbouw van een component

```
TS app.component.ts x  TS app.module.ts      # app.component.css      # styles.css
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-hallo',
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9   title = 'Hallo Web Expert!';
10  naam = 'Dries Swinnen';
11 }
```

@Component

- selector
- template
- styles
- templateUrl
- styleUrls
- input
- output
- providers
- ...

Components maken

- Handmatig aanmaken van de file
 - hallo.component.ts
- Voorzien van component import

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

- Voorzien van annotatie

```
@Component({  
  selector: 'app-hallo',  
  template: `<p>Hallo! dit is een test</p>`,  
  styles:[`p {color: blue;} `],  
})
```

Components maken

- Voorzien van component klasse

```
Export class HalloComponent {  
}
```

- Toevoegen aan de declarations in de app.module.ts file

```
import { HalloComponent } from './hallo.component';  
  
@NgModule({  
  declarations: [  
    AppComponent,  
    HalloComponent  
  ],
```

- Vervolgens kan je de selector <app-hallo> gebruiken in je project.

Components maken

- hallo.component.ts

```
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-hallo',
5   template: `<p>Hallo! Dit is een test.</p>`,
6   styles: [`p {color: blue;}`],
7 })
8 export class HalloComponent {
9
10 }
11
```

Components maken

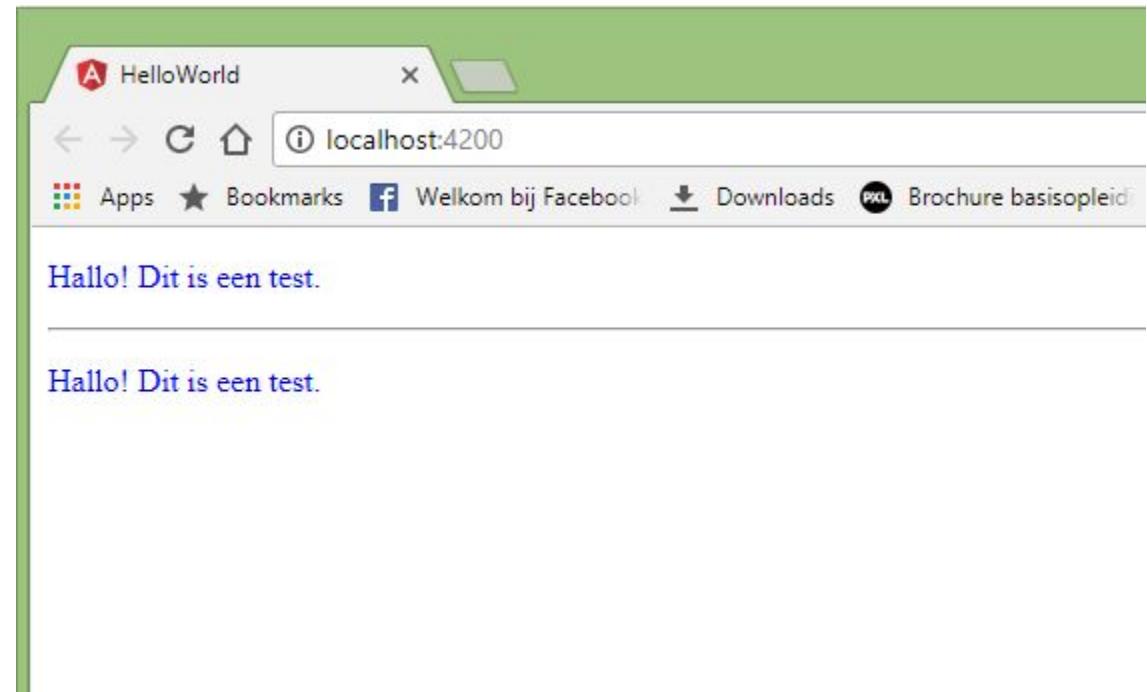
- app.module.ts

```
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3 import { AppComponent } from './app.component.';
4 import { HalloComponent } from './hallo.component';
5
6
7 @NgModule({
8   declarations: [
9     AppComponent,
10    HalloComponent
11   ],
12   imports: [
13     BrowserModule
14   ],
15   providers: [],
16   bootstrap: [AppComponent]
17 })
18 export class AppModule { }
```

Components maken

- app.component.html

```
1  <app-hallo></app-hallo>
2  <hr>
3  <app-hallo></app-hallo>
4
5
```



Components maken

- AngularCLI

```
ng generate component path/to/component
```

Lifecycle hooks

- Components worden gemanaged door Angular zelf.
- Ze worden automatisch aangemaakt, gerenderd en vernietigd.
- Lifecycle hooks worden gebruikt om code uit te voeren op bepaalde tijdstippen:
 - `ngOnInit()`: Na het uitvoeren van de constructor van een component
 - `ngOnChanges()`: Na het aanpassen van een data-gebonden input veld
 - `ngOnDestroy()`: Bij het vernietigen van de component
 - ...

ngOnInit

- Meest gebruikte lifecycle hook
- Meestal om data op te halen die nodig is in de component
- Importeren uit @angular/core:

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

- OnInit interface implementeren op de component klasse:

```
export class halloComponent implements OnInit { ... }
```

- Methode met de naam ngOnInit() toevoegen aan de klasse

```
ngOnInit(){  
    ...  
}
```

ngOnInit

```
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-hallo',
5   template: `<p>Hallo! Dit is een test.</p>`,
6   styles: [`p {color: blue;} `],
7 })
8 export class HalloComponent implements OnInit {
9   constructor() {
10     console.log('uitvoeren constructor');
11   }
12   ngOnInit(): void {
13     console.log('initiatie van de component, na uitvoeren constructor');
14   }
15 }
16 }
```

Uitgewerkt voorbeeld: Login-component

- [CH3-voorbeeld1](#)
- Maak een nieuwe file aan login.component.ts met volgende inhoud:

```
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-login',
5   templateUrl: './login.component.html',
6   styles:[ ],
7 })
8 export class LoginComponent {
9   name: string;
10  password: string;
11  message: string = 'Please enter login details!';
12 }
13
```

Uitgewerkt voorbeeld: Login-component

- Maak een nieuw bestand login.comonent.html en voorzie volgende html code:

```
1 <h2>Please log-in</h2>
2 <table>
3   <tr>
4     <td>Name</td>
5     <td><input type="text" [(ngModel)]="name"/></td>
6   </tr>
7   <tr>
8     <td>Password</td>
9     <td><input type="password" [(ngModel)]="password"/></td>
10   </tr>
11 </table>
12 <p>{{ message }}</p>
13 <button (click)="verwerk()">Submit</button>
```

Uitgewerkt voorbeeld: Login-component

- Voeg de LoginComponent toe aan de declarations in de app.module.ts file:

```
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3 import { AppComponent } from './app.component.';
4 import { LoginComponent } from './login.component';

5

6

7 @NgModule({
8   declarations: [
9     AppComponent,
10    LoginComponent
11   ],
12 })

13 
```

- Voeg de <app-login></app-login> tag toe aan de app.component.html file

[(ngModel)]

- 2 way databinding
- Zorgt voor een verbinding tussen de inhoud van het input veld en een variabele in de component klasse
- Aanpassingen gebeuren automatisch in beide richtingen!
- **Inladen van de FormsModule in de app.module.ts file**

```
5 import { FormsModule } from '@angular/forms'; //voor gebruik ngModel
6
7 @NgModule({
8   declarations: [
9     AppComponent,
10    LoginComponent
11   ],
12   imports: [
13     BrowserModule,
14     FormsModule, //voor gebruik ngModel
15   ],
16 })
17 export class AppModule { }
```

[(ngModel)]

```
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-login',
5   templateUrl: './login.component.html',
6   styles: [ ],
7 })
8 export class LoginComponent {
9   name: string;
10  password: string;
11  message: string = 'Please enter login details!';
12 }
13
```

```
1 <h2>Please log-in</h2>
2 <table>
3   <tr>
4     <td>Name</td>
5     <td><input type="text" [(ngModel)]="name"/></td>
6   </tr>
7   <tr>
8     <td>Password</td>
9     <td><input type="password" [(ngModel)]="password"/></td>
10    </tr>
11 </table>
12 <p>{{ message }}</p>
13 <button (click)="verwerk()">Submit</button>
```

- {{ ... }} wordt gebruikt als one way binding (model to view)

Event handling

- One way databinding van view naar model
- Events worden gekoppeld in de html code naar een methode in de component klasse

```
13  <button (click)="verwerk()">Submit</button>
```

- Verschillende events zoals: submit, click, dblclick, dragover, focus, blur, keydown, ...

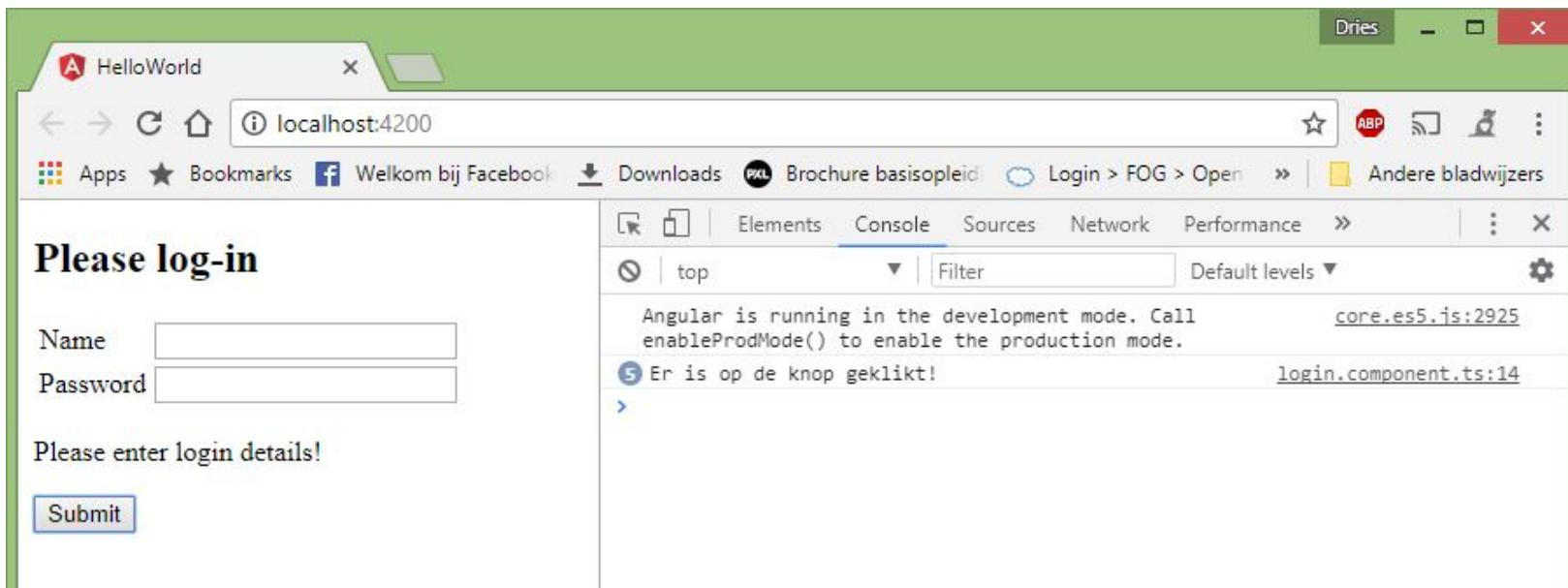
Event handling

- De event wordt altijd gekoppeld aan een methode in de component klasse:

```
8  export class LoginComponent {  
9    name: string;  
10   password: string;  
11   message: string = 'Please enter login details!';  
12  
13   verwerk(){  
14     console.log('Er is op de knop geklikt!');  
15   }  
16 }
```

- Verwerk() wordt opgeroepen bij het klikken op de button.

Event handling



Event handling & databinding

- In onderstaand voorbeeld wordt message aangepast.
- Name heeft automatisch de waarde van het input veld door **[(ngModel)]** (**2 way databinding**)
- Message wordt automatisch aangepast op de view door de verwijzing **{{ message }}** (**1 way databinding – model to view**)

```
8  export class LoginComponent {  
9    name: string;  
10   password: string;  
11   message: string = 'Please enter login details!';  
12  
13   verwerk(){  
14     this.message = this.name + ' has logged in!';  
15   }  
16 }
```

Please log-in

Name

Password

Dries has logged in!

Databinding & eventbinding

Angular



Databinding

- Elementen en events kunnen gekoppeld worden aan:
 - Code in de component klasse
 - Properties (in de component klasse)
- Meerdere syntaxen mogelijk, uitwerking is anders!
 - One way binding
 - Two way binding

Databinding

Type binding	Omschrijving	Syntax
One-way binding Controller to view	Output van een variabele (property) in de component klasse (of andere elementen in de view) tonen in de view	<code>{{ variable_name }}</code>
Two-way binding	Waardes uit een input veld koppelen aan een variabele (property) in de component klasse	<code>[ngModel]="variable_name"</code>
One-way binding View to controller	View events koppelen aan een functie uit de component klasse	<code>(click)='voorbeeldmethode()'</code>
One-way binding View to controller	Eigenschappen van een element koppelen aan een variabele (property) van de component klasse (of andere waardes in de view).	<code>[disabled] = "isDisabled"</code>

One-way output binding

- Interpolatie
- Output data van de component klasse naar de view
- Double brackets syntax

```
{{ expressie }}
```

One way output binding

- De expressie binnen de accolades kan een aantal zaken bevatten:

- Variabele

```
 {{ contact.name }}
```

- Expressie

```
 {{ a + b }}
```

- Functie

```
 {{ getMessage() }}
```

- String

```
 {{ 'hallo wereld' }}
```

One way output binding

- Variabele staan meestal in de component klasse
- Kunnen soms ook verwijzen naar andere elementen in de view

```
1 <form #f="ngForm">
2   <input name="first" ngModel required #first="ngModel"/>
3 </form>
4 <p>First name value: {{ first.value }}</p>
5 <p>Form valid: {{ f.valid }}</p>
```

One way output binding: voorbeelden

```
template: `<h4>String</h4>
<p>Er zijn {{days}} {{unit}} in een jaar!</p>
<h4>Datum</h4>
<p>{{today|date: 'dd/MM/yy'}}</p>
<h4>Tekst</h4>
<p>{{text}}</p>`,
styleUrls: ['./output.component.css']
})
export class OutputComponent {
  text: string = 'Lorem ipsum dolor sit amet';
  days: number = 365;
  unit: string = 'dagen';
  today: Date = new Date();
}
```

String

Er zijn 365 dagen in een jaar!

Datum

01/12/17

Tekst

Lorem ipsum dolor sit amet

One way output binding: voorbeelden

- Bij de eerste reeks worden de variabelen in een zin geplaatst

```
<p> Er zijn {{days}} {{unit}} in een jaar!</p>
```

- Bij het 2^{de} voorbeeld wordt er een pipe gebruikt om de datum property een format mee te geven:

- Pipes worden later verder behandeld

```
{{today|date: 'dd/MM/yy' }}
```

- Alle variabelen worden gedefinieerd in de outputComponent klasse

```
text: string = 'Lorem ipsum ...';
days: number = 365;
unit: string = 'dagen';
today: Date = new Date(); //Let op de hoofdletter in het datatype
```

Two-way input binding

- In 2 richtingen
- Composite Bracket syntax
 - [(ngModel)] = “variabeleNaam”;
- Wordt gebruikt bij input velden
- Voorbeeld:

```
<input type="tekst [(ngModel)]="first_name">
```

```
export class exampleComponent {  
  first_name: string = "Dries Swinnen";  
}  
c
```

Two-way input binding

- Om two-way binding te gebruiken, moet je de **FormsModule** toevoegen aan de `app.module.ts` file!!

```
import { FormsModule } from '@angular/forms';
@NgModule({
  declarations: [
    AppComponent,
    OutputComponent,
  ],
  imports: [
    BrowserModule,
    FormsModule,
  ],
  providers: [],
  bootstrap: [AppComponent])
})
```

Two-way input binding: voorbeelden

```
template: `<h4>Tekst input</h4>
Voornaam: <input type="text"
[(ngModel)]="voornaam"/><br/>
Je voornaam is: {{voornaam}}
<h4>Datum input</h4>
Je verjaardag: <input type="date"
[(ngModel)]="verjaardag"/><br/>
Je verjaardag is: {{verjaardag}}` ,
styleUrls: ['./input.component.css']
})
export class InputComponent {
  voornaam: string;
  verjaardag: Date;
}
```

Tekst input

Voornaam:

Je voornaam is: Dries

Datum input

Je verjaardag:

Je verjaardag is: 1990-08-04

Event binding

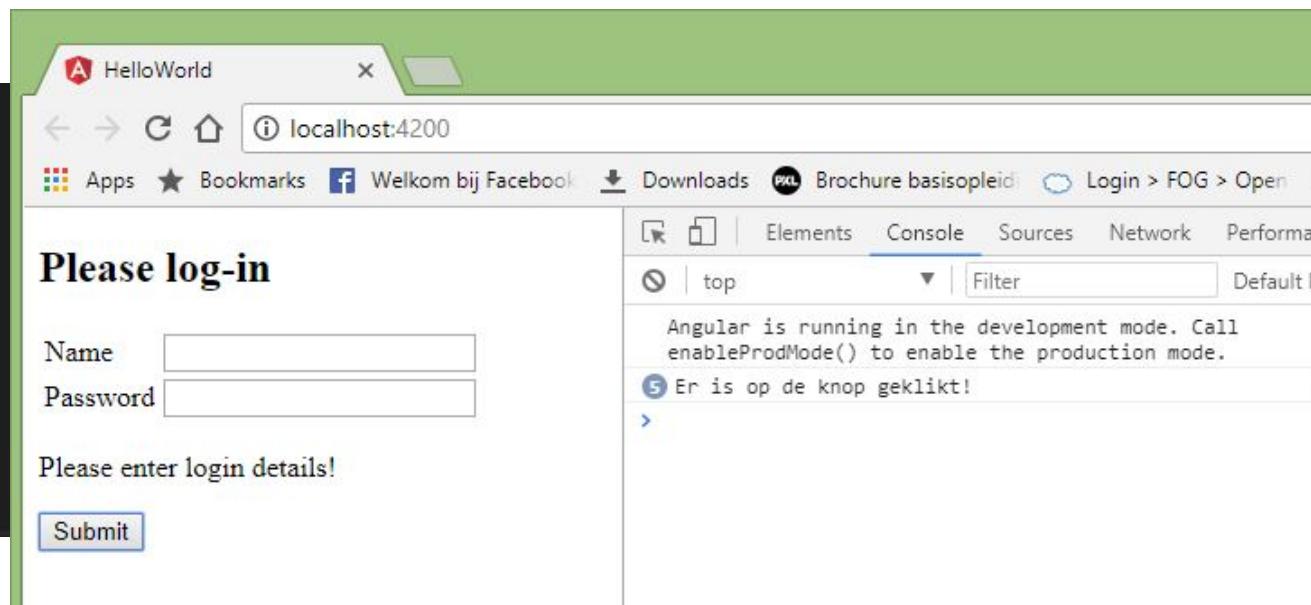
- Functies uit de Component klasse kunnen gekoppeld worden aan DOM events
 - Verschillende events zoals: submit, click, dblclick, dragover, focus, blur, keydown, keyup ...
- One way databinding van view naar model

```
13  <button (click)="verwerk()">Submit</button>
```

Event binding

- Koppeling kan op eender welk element
 - Ook op components!

```
8 export class LoginComponent {  
9   name: string;  
10  password: string;  
11  message: string = 'Please enter login details!';  
12  
13  verwerk(){  
14    console.log('Er is op de knop geklikt!');  
15  }  
16 }
```



Event handling & databinding

- In onderstaand voorbeeld wordt message aangepast.
- Name heeft automatisch de waarde van het input veld door **[(ngModel)]** (**2 way databinding**)
- Message wordt automatisch aangepast op de view door de verwijzing **{{ message }}** (**1 way databinding – model to view**)

```
8  export class LoginComponent {  
9    name: string;  
10   password: string;  
11   message: string = 'Please enter login details!';  
12  
13   verwerk(){  
14     this.message = this.name + ' has logged in!';  
15   }  
16 }
```

Please log-in

Name

Password

Dries has logged in!

Event handling: voorbeelden

```
template: `<input id="txt1" type="text"
(keydown)="toggle('txt1') [(ngModel)]="voornaam">
<button id="btn1" (click)="toggle('btn1')">
Button</button>
<div id="div1" (mouseover)="toggle('div1')"
(mouseleave)="toggle('div1')">Mouse over me!</div>
`,
styleUrls: ['./event.component.css']
})
export class EventComponent {
toggle(id) {
let e = document.getElementById(id);
let bc = e.style.backgroundColor;
bc = (bc != 'lightblue') ? 'lightblue' : 'yellow';
e.style.backgroundColor = bc;
}
}
```

Change Event

Click event



Mouse event

Mouse over me!

Data doorgeven met @Input()

- Bij geneste components kan er data doorgegeven worden **van parent naar child component**
- Onderstaand voorbeeld: Parent en Child zijn onafhankelijk van elkaar

```
// parent.component.ts
@Component({
  selector: 'app-parent',
  template: `
    <h1>Parent</h1>
    <app-child></app-child>
  `,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent {
  msgText: string='dit is een string uit de parent';
}
```

```
// child.component.ts
@Component({
  selector: 'app-child',
  template: `
    <h2>Child component</h2>
    <p>{{msg}}</p>
  `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {
  msg: string = 'child string';
}
```

Data doorgeven met @Input()

- @Input() annotatie wordt toegevoegd aan een property in de component Klasse van de child component
- Zorgt ervoor dat de variabele msg gevuld wordt via property binding vanuit de parent
- Input import voorzien

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

// child.component.ts
@Component({
  selector: 'app-child',
  template: `
    <h2>Child component</h2>
    <p>{{msg}}</p>
  `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {
  @Input() msg: string;
}
```

Data doorgeven met @Input()

- In de parent component voorzien we in de declaratie van de child component:
 - Koppeling naar de input variabele in de child -- [msg]
 - Koppeling naar de variabele in de parent -- msgText

```
// parent.component.ts
@Component({
  selector: 'app-parent',
  template: `
    <h1>Parent</h1>
    <app-child [msg]="msgText"></app-child>
  `,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent {
  msgText: string='dit is een string uit de parent';
}
```

Parent

Child component

dit is een string uit de parent

Data doorgeven met @Input()

```
// parent.component.ts
@Component({
  selector: 'app-parent',
  template: `
    <h1>Parent</h1>
    <app-child [msg]="msgText"></app-child>
  `,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent {
  msgText: string='dit is een string uit de parent';
}
```

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

// child.component.ts
@Component({
  selector: 'app-child',
  template: `
    <h2>Child component</h2>
    <p>{{msg}}</p>
  `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {
  @Input() msg: string;
}
```

Data doorgeven met @Output()

- Bij geneste components kan er data doorgegeven worden **van child naar parent** component
- Onderstaand voorbeeld: Parent en Child zijn onafhankelijk van elkaar

```
// parent.component.ts
@Component({
  selector: 'app-parent',
  template: `
    <h1>Parent</h1>
    <app-child></app-child>
  `,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent {
  msgText: string='dit is een string uit de parent';
}
```

```
// child.component.ts
@Component({
  selector: 'app-child',
  template: `
    <h2>Child component</h2>
    <button (click)="onChange()">Click me</button>
  `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {
  msg: string = 'message from the child';

  onChange() {
    console.log('Button clicked');
  }
}
```

Date doorgeven met @Output()

- Binnen de child component wordt er een eventEmitter aangemaakt.
- Deze property krijgt de @Output() annotatie.
- Import voor eventEmitter en @Output() voorzien

```
import { Component, Output, EventEmitter } from '@angular/core'

// child.component.ts
@Component({
  selector: 'app-child',
  template:
    `<h2>Child component</h2>
     <button (click)="onChange()">Click me</button>
    `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {
  msg: string = 'Message from the child';
  @Output() childValueChange = new EventEmitter();

  onChange() {
    this.childValueChange.emit(this.msg);
  }
}
```

Data doorgeven met @Output()

- In de Parent Component wordt de childValueChange event gekoppeld aan een methode in de parent component klasse
- Methode krijgt \$event als argument

```
// parent.component.ts
@Component({
  selector: 'app-parent',
  template: `
    <h1>Parent</h1>
    <app-child (childValueChange)="eventParent($event)">
    </app-child>
  `,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent {

  eventParent(event: string){
    console.log('event from child:' + event);
  }
}
```

Data doorgeven met @Output()

```
// parent.component.ts
@Component({
  selector: 'app-parent',
  template: `
    <h1>Parent</h1>
    <app-child (childValueChange)="eventParent($event)">
    </app-child>
  `,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent {
  eventParent(event: string){
    console.log('event from child:' + event);
  }
}
```

```
// child.component.ts
@Component({
  selector: 'app-child',
  template: `
    <h2>Child component</h2>
    <button (click)="onChange()">Click me</button>
  `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {
  msg: string = 'Message from the child';
  @Output() childValueChange = new EventEmitter();

  onChange() {
    this.childValueChange.emit(this.msg);
  }
}
```



Data doorgeven met `@Output()`

- `$event`
 - parameter wordt altijd meegegeven aan de callback methode
 - Kan eender welk datatype bevatten
 - String, number, Date, Object, Person, Contact, ...

Attribute directives property bindings



Wat zijn directives

- Stukken HTML code met Angular logica
 - Koppelen van specifiek gedrag
 - Transformatie van een element
- Directives bestaan in 2 vormen in HTML templates:
 - Als attribuut van een element:

```
<div *ngIf="someVar">...</div>
```
 - Als elementnaam:

```
<app-myDirective></app-myDirective>
```
- Een deel directives zijn builtIn, andere directives zelf aanmaken

Soorten directives

Type	Omchrijving	Voorbeeld
Component	Een klasse met variabelen, methodes, een HTML template en een directive naam (selector tag).	<app-login></app-login> <app-contacts></app-contacts>
Attribuut directives	Wijzigt het element waarin de directive staat op een bepaalde manier.	ngStyle, ngClass, ...
Structurele directives	Aanbrengen van wijzigingen aan de DOM door elementen toe te voegen of te verwijderen.	ngIf, ngFor, ...

Stijlen toepassen m.b.v. [ngClass]

- ngClass: dynamisch toevoegen of verwijderen van een klasse aan een element

```
<div [ngClass] = "conditie">...</div>
```

- De conditie kan een value, expressie of functie zijn.

Stijlen toepassen m.b.v. [ngClass]

- De conditie van ngClass kan verschillende waardes krijgen:

Datatype	Voorbeeld	Omschrijving
String	“active bordered”	Alle klassen in de string worden gekoppeld aan het element. In dit voorbeeld zijn dit de klassen ‘active’ en ‘bordered’.
Array	[“active”, “bordered”]	Alle klassen in de array worden gekoppeld aan het element. In dit voorbeeld zijn dit de klassen ‘active’ en ‘bordered’.
Object	{active: isActive, bordered: hasBorder}	Voegt elke klasse toe, waarvan de waarde true is. In dit voorbeeld wordt de klasse ‘active’ toegevoegd als de variabele ‘isActive’ de waarde true heeft.

Stijlen toepassen m.b.v. [ngClass]

- Praktisch voorbeeld waarbij we starten van onderstaande component:

```
@Component({
  selector: 'app-styles',
  template: `<span>Test tekst</span>
    <br/> Voorzie rand: <input type="checkbox" [(ngModel)]="hasBorder"/><br/>
    Actief: <input type="checkbox" [(ngModel)]="isActive"/>`,
  styles: [`.active{color: red;}` .bordered{border: solid 1px black;}`]
})
export class StylesComponent {
  hasBorder: boolean = false;
  isActive: boolean = false;
}
```

- 2 CSS klasses voorzien: .active en .bordered
- De variabelen uit de StylesComponent klasse zijn gekoppeld aan de checkboxen via 2-way binding

Stijlen toepassen m.b.v. [ngClass]

- ngClass import toevoegen
- krijgt de ngClass directive met een object als value:
 - .bordered wordt toegevoegd als hasBordered = true
 - .active wordt toegevoegd als isActive = true;

```
<span [ngClass]="{bordered: hasBorder, active: isActive}">Test tekst</span>
<br/>
Voorzie rand: <input type="checkbox" [(ngModel)]="hasBorder"/><br/>
Actief: <input type="checkbox" [(ngModel)]="isActive"/>
```

Stijlen toepassen m.b.v. [ngClass]

```
import { Component, OnInit } from '@angular/core';
import { NgClass } from '@angular/common';

@Component({
  selector: 'app-styles',
  template: `<span
[ngClass]="{bordered: hasBorder, active: isActive}">Test tekst</span>
<br/> Voorzie rand: <input type="checkbox" [(ngModel)]="hasBorder"/><br/>
Actief: <input type="checkbox" [(ngModel)]="isActive"/>`,
  styles: [`.active{color: red;}` `.bordered{border: solid 1px black;}`]
})
export class StylesComponent {
  hasBorder: boolean = false;
  isActive: boolean = false;
}
```

Test tekst
Voorzie rand:
Actief:

Test tekst
Voorzie rand:
Actief:

Stijlen toepassen m.b.v. [ngStyle]

- **ngStyle**: Hiermee kunnen we CSS stijlen direct toevoegen aan een HTML element:

```
<div [ngStyle] = "styleExpressie">...</div>
```

- De conditie kan een Object, variabele of functie zijn.

Stijlen toepassen m.b.v. [ngStyle]

- De expressie van ngStyle kan verschillende waardes krijgen:

Type	Voorbeeld
Object	[ngStyle] = “{color: ‘blue’}”;
Variabele (uit de component)	[ngStyle] = “styles” // In de HTML styles: Object = { color: ‘blue’ } // In component klasse
Functie	[ngStyle] = “getStyles()” // In de HTML getStyles(){ // In de component klasse return {color: ‘blue’}; }

Stijlen toepassen m.b.v. [ngStyle]

- Praktisch voorbeeld waarbij we starten van onderstaande component:

```
@Component({
  selector: 'app-styles2',
  template: `<span>Tekst 1</span><br/>
    <span (click)="toggleStyles()">Klik mij</span>`,
})
export class Styles2Component {
  styles1: Object = { border: 'solid 1px black', color: 'red' };
  styles2: Object = { border: 'none', color: 'black' };
  styles: Object = this.styles1;

  toggleStyles() {
    if (this.styles = this.styles1) {
      this.styles = this.styles2;
    } else {
      this.styles = this.styles1;
    }
  }
}
```

- Styles zijn reeds voorzien als objecten in de component klasse
- toggleStyles() zorgt ervoor dat de variabele styles aangepast wordt

Stijlen toepassen m.b.v. [ngStyle]

- ngStyle import toevoegen
- krijgt de ngStyle directive met een object als value
- krijgt de ngStyle directive met een variable als value

```
<span [ngStyle]="{fontStyle:'italic', fontFamily:'sans-serif'}">Tekst 1</span>
<br/><span [ngStyle]="styles" (click)="toggleStyles()">Klik mij</span>
```

Stijlen toepassen m.b.v. [ngStyle]

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

@Component({
  selector: 'app-styles2',
  template: `<span>
    [ngStyle]="{fontStyle:'italic', fontFamily:'sans-serif'}">Tekst 1</span>
    <br/><span [ngStyle]="styles" (click)="toggleStyles()">Klik mij</span>`,
})
export class Styles2Component {
  styles1: Object = { border: 'solid 1px black', color: 'red' };
  styles2: Object = { border: 'none', color: 'black' };
  styles: Object = this.styles1;

  toggleStyles() {
    if (this.styles == this.styles1) {
      this.styles = this.styles2;
    } else {
      this.styles = this.styles1;
    }
  }
}
```

Tekst 1

Klik mij

Tekst 1

Klik mij

Element properties in Angular

- Binnen angular kan je eender welke DOM properties van elementen aanspreken

```
[prop_naam] = "variabele|expressie"
```

- De value kan volgende zaken bevatten:
 - Statische waardes
 - Variabelen
 - Expressies
 - Functies

Element properties in Angular

- Binding is dynamisch – properties veranderen als de variabele / statement verandert
- Voorbeeld:

```
[style.color] = “isRed ? ‘red’ : ‘blue’;”
[disabled] = “isDisabled”
[style.width] = “50px”
```

Zichtbaarheid van een element

- Gebruik maken van [hidden] property
- Syntax:

```
<div [hidden] = "conditie">...</div>
```

- De conditie kan een variabele of expressie zijn
- Voorbeeld:

```
<div [hidden] = "!showDiv">...</div>
```

Dynamische source van een afbeelding

- Gebruik maken van de [src] property
- Syntax:

```
<img [src] = "imageUrl" />
```

- De conditie kan een variabele of functie zijn
- [Src] kan hier dus ook variabel zijn, wat met “src=“ niet kan
- Voorbeeld:

```
<img [src] = "getImageUrl()" > ... </img>
```

Source van een afbeelding

```
@Component({
  selector: 'app-image',
  template: `<h3>Scr voorbeeld</h3>
<img [src]="imageUrl"/><br/>
<input type="button" value="wijzig" (click)="toggleImage()">`,
  styleUrls: ['./image.component.css']
})
export class ImageComponent {
  imageUrl = 'assets/badge1.png';
  index: number = 1;

  toggleImage() {
    this.index = this.index + 1;
    this.index = (this.index < 5) ? this.index : 1;
    this.imageUrl = 'assets/badge' + this.index + '.png';
  }
}
```

Scr voorbeeld



wijzig

Scr voorbeeld



wijzig

Dynamische hyperlinks

- Gebruik maken van de [href] property
- Syntax:

```
<a [href] = "hrefUrl" />
```

- De conditie kan een variabele of functie zijn
- Href kan hier dus ook variabel zijn, wat met “href=” niet kan
- Voorbeeld:

```
<a [href] = "getHrefUrl()" >...</div>
```

Structurele directives

- Structurele directives passen de DOM aan door:
 - Elementen toe te voegen
 - Elementen te verwijderen
- *ngIf
- *ngFor
- *ngSwitch

Structurele directives: *ngIf

- *ngIf
- Bepaalde DOM elementen toevoegen of verwijderen
- Meestal gekoppeld aan een boolean variabele, expressie of functie
- Syntax:

```
<div *ngIf="showMessage"> ... </div>
```

- Als showMessage true is, wordt de div toegevoegd aan de DOM
- Het *-teken geeft aan dat dit een verkorte schrijfwijze is

Structurele directives: *ngIf

- Enkele voorbeelden:

- //gekoppeld aan de variable isVisible van het type boolean in de component klasse

```
*ngIf="isVisible"
```

- Gekoppeld aan de variabele myVar van het type string.
ngIf verwacht een boolean

```
*ngIf="myVar == 'showit'"
```

- Gekoppeld aan een functie in de component klasse die een boolean retourneert:

```
*ngIf="showIt()"
```

Structurele directives: *ngFor

- *ngFor is een repeater directive
- Loopt door JavaScript arrays
- Wordt voornamelijk gebruikt om lijsten en tabellen op te stellen.
- Kan toegepast worden op elk element:
 - <div>
 - <p>
 - <app-contact>

Structurele directives: *ngFor

- Basis Syntax:

```
<li *ngFor="let pet of pets">  
{{pet}}  
</li>
```

- Data van de array uit de component klasse:

```
Pets: string[] = ["Cat", "Dog", "Turtle"];
```

- pets verwijst naar de array uit de klasse. Daarnaast wordt er een lokale variabele pet aangemaakt.

Structurele directives *ngFor

```
@Component({
  selector: 'app-ngfor',
  template: `
    <table>
      <tr *ngFor="let person of people">
        <td>{{person.name}}</td>
        <td>{{person.firstName}}</td>
        <td>{{person.email}}</td>
      </tr>
    </table>
  `,
  styles: [`table, td{ border: 1px solid black`]
})
export class NgforComponent {
  people: object = [
    {name: 'Swinnen', firstName: 'Dries', email: 'dries.swinnen@pxl.be'},
    {name: 'Doe', firstName: 'John', email: 'john.doe@gmail.com'},
    {name: 'Doe', firstName: 'Jane', email: 'jane.doe@gmail.com'},
    {name: 'Vader', firstName: 'Darth', email: 'anakin@theEmpire.com'}
  ];
}
```

Structurele directives: *ngFor

- *ngFor voorziet enkele loop-gerelateerde waardes die je kan gebruiken binnen de loop:
 - Index: array index van het huidige item
 - First: boolean value die true is als het element het eerste element uit de array is
 - Last: boolean value die true is als het element het laatste element uit de array is
 - Even: boolean value die true is al het element zijn positie in de array even is
 - Odd: boolean value die true is als het element zijn positie in de array odd is

Structurele directives: *ngFor

```
template: `<table>
  <tr *ngFor="let person of people; let i=index; let e=even;" [ngClass]="{evenrow:e,oddrow:!e}">
    <td>{{i}}</td>
    <td>{{person.name}}</td>
    <td>{{person.firstName}}</td>
    <td>{{person.email}}</td>
  </tr>
</table>
`,
styles: [`.evenrow{ background-color: yellow; }
         .oddrow{ background-color: orange;}`]
})
export class NgforComponent {
  people: object = [
    {name: 'Swinnen', firstName: 'Dries', email: 'dries.swinnen@pxl.be'},
    {name: 'Doe', firstName: 'John', email: 'john.doe@gmail.com'},
    {name: 'Doe', firstName: 'Jane', email: 'jane.doe@gmail.com'},
    {name: 'Vader', firstName: 'Darth', email: 'anakin@theEmpire.com'}
  ];
}
```

Structurele directives: *ngFor

- Angular bekijkt de achterliggende array om de view te updaten.
- Data die toegevoegd wordt aan de array, wordt ook automatisch aangepast in de view
- Data die verwijderd wordt uit de array, wordt ook automatisch aangepast in de view
- Items toevoegen en verwijderen aan een array, gaat met eenvoudige JavaScript methodes:

```
pets.push(petToAdd);  
pets.splice(indexToRemove,1);
```

*ngFor en @Input()

- Het is mogelijk om het element uit de array door te geven naar een @Input() variabele
- Dit is nodig als je bijvoorbeeld een *ngFor in een child component plaatst

```
<!--  
  parent html  
  contactList is an array of contacts  
-->  
<app-contact *ngFor="let contactItem of contactList;"  
             [contactChild]="contactItem">  
</app-contact>
```

```
// <app-contact> component.  
// Gets input from ngFor  
export class ContactComponent implements OnInit {  
  @Input() contactChild: Contact;  
  
  ngOnInit() { }  
}
```

Structurele directives: *ngSwitch

- *ngSwitch is een structurele directive
- Wordt gebruikt naast ngSwitchCase en ngSwitchDefault
- Toont één element uit een bepaalde set, en verbergt de andere
- Kan gebruikt worden om bijvoorbeeld tabs te genereren.

Structurele directives: *ngSwitch

```
@Component({
  selector: 'app-ngswitch',
  template: `
    <div [ngSwitch]="types[index]">
      <p *ngSwitchCase="'one'">Page 1</p>
      <p *ngSwitchCase="'two'">Page 2</p>
      <p *ngSwitchCase="'three'">Page 3</p>
      <p *ngSwitchDefault>Default</p>
    </div>
    <button (click)="next()">Volgende</button>`,
  styleUrls: ['./ngswitch.component.css']
})
export class NgswitchComponent {
  types: string[] = ['one', 'two', 'three'];
  index: number = 0;

  next(){
    this.index++;
    this.index = (this.index < 4) ? this.index : 0;
  }
}
```

Forms, validation & functions

Angular



Types of forms in Angular

- Databinding + formulieren was het USP van de eerste versies van Angular
- Template driven forms
- Model driven forms

Template driven forms

- Gebruiken ngModel bindings
 - 2 way binding
- Ondersteunen validatie vanuit de HTML
 - Angular validation / HTML5 validation
- End-to-end testing methodes
- Voornamelijk gebruikt bij formulieren van medium grootte waar weinig interactie nodig is

Template driven forms

- Om template driven forms te gebruiken, moet de `FormsModule` toegevoegd worden aan de `app.module.ts` file:

```
import { FormsModule } from '@angular/forms';

@NgModule({
  ...
  imports: [
    BrowserModule,
    FormsModule
  ],
  ...
})
```

Template driven forms - voorbeeld

- In volgend voorbeeld gebruiken we het form object
- Input velden worden gekoppeld aan het formulier
- Een functie wordt gekoppeld aan de submit button
- Data van de velden wordt meegegeven aan de submit button

Template driven forms - voorbeeld

- Starten van volgende code in een default component:

```
<form>
  Name:<br/>
  <input type="text" name="name" [(ngModel)]="name" /> <br/>
  Email:<br/>
  <input type="email" name="email" [(ngModel)]="email" /><br/>
  <button type="submit">Submit</button>
</form>
```

The screenshot shows a user interface with a light beige background. It contains two text input fields labeled "Name:" and "Email:", each with a corresponding empty input box below it. Below these fields is a "Submit" button.

Template driven forms - voorbeeld

- De koppeling van de input velden gebeurt aan de hand van 2-way binding:
 - [(ngModel)]

```
<input type="text" name="name" [(ngModel)]="name" />
```

- De 2-way binding zorgt er ook voor dat de input velden binnen een <form> tag gekoppeld worden aan een form object.
- Dit form object bevat data over:
 - Input van de velden
 - Informatie over de validatie van het formulier

Template driven form - voorbeeld

- Het aangemaakte Angular form object kan je gebruiken door een referentie te voorzien als volgt:

```
<form #mijnForm="ngForm">
```

- Variable ‘mijnForm’ wordt aangemaakt met het form object. Dit object bevat:
 - mijnForm.form.value: object met de waardes van input velden
 - mijnForm.form.valid: boolean of alle validatieregels valid zijn
 - mijnForm.form.submitted: boolean of het form gesubmit is.

Template driven form - voorbeeld

- Koppelen van het submitten van het formulier aan een methode in de klasse:

```
<form #mijnForm="ngForm" (ngSubmit)="onSubmit(mijnForm)">
```

- Het object mijnForm wordt meegegeven in de methode
 - Op deze manier krijgt de methode toegang tot de values & status van het formulier
- Een element van het type submit zorgt voor het triggeren van de onSubmit methode
 - “action” attribuut is niet nodig!

Template driven form - voorbeeld

- De onSubmit methode wordt voorzien in de component klasse:

```
onSubmit(form) {  
  console.log('Submitted:' +  
            JSON.stringify(form.value, null, 2));  
}
```

- De form.value waardes zijn gekoppeld aan het “name” attribuut van de input velden.

Template driven form - voorbeeld

- Templateform.component.ts:

```
export class TemplateformComponent {  
  onSubmit(form) {  
    console.log('Submitted: ' + JSON.stringify(form.value, null, 2));  
  }  
}
```

- Templateform.component.html:

```
<form #mijnForm="ngForm"  
      (ngSubmit)="onSubmit(mijnForm)">  
  Name:<br/>  
  <input type="text" name="name" [(ngModel)]="name" /> <br/>  
  Email:<br/>  
  <input type="email" name="email" [(ngModel)]="email" /><br/>  
  <button type="submit">Submit</button>  
</form>
```

Template driven form - voorbeeld

The screenshot displays a web browser interface. On the left, a simple form is shown with two input fields and a submit button. The first field is labeled "Name:" and contains the value "Dries Swinnen". The second field is labeled "Email:" and contains the value "dries.swinnen@pxl.be". Below these fields is a "Submit" button. On the right, the browser's developer tools are open, specifically the "Console" tab. The console output shows two messages: the first message is a general runtime notice about the development mode; the second message is the JSON representation of the submitted form data, indicating successful data binding.

```
Angular is running in the core.es5.js:2925
development mode. Call enableProdMode() to
enable the production mode.

Submitted:{ templateform.component.ts:13
  "name": "Dries Swinnen",
  "email": "dries.swinnen@pxl.be"
}
```

Template driven forms – HTML Validatie

- Standaard Angular validation actief
- Validatie kan aan de hand van HTML5 validatie:
 - HTML5 attribuut 'required'
 - Input types zoals date, email, url, ...
- Wil je gewone HTML5 validatie gebruiken moet je **ngNativeValidate**

```
<form #mijnForm="ngForm" ngNativeValidate  
(ngSubmit)="onSubmit(mijnForm)">  
Name:<br/>  
<input type="text" name="name" [(ngModel)]="name" required />
```

Template driven forms – Angular validatie

- Naast HTML5 validatie kan je ook Angular validatie gebruiken
 - Angular validatie staat standaard actief
- Meer mogelijkheden
- Betere integratie met je component & het Angular framework:
 - Error message na het verlaten van een inputveld (dynamisch)
 - Aangepaste error messages
 - Programmatorisch testen van form validatie
 - Zelf schrijven van validatieregels

Template driven forms – Angular validatie

- Angular heeft volgende validators die toegevoegd kunnen worden aan input elementen binnen template driven forms:
 - Required
 - minLength
 - maxLength
 - Pattern
- Combinatie van verschillende validators is mogelijk

Template driven forms – Angular validatie

```
<form>
  <input type="text" required name="name" [(ngModel)]="name">
  <input type="text" minlength="3" name="street" [(ngModel)]="street">
  <input type="text" maxlength="8" name="city" [(ngModel)]="city">
  <input type="text" pattern="[A-Za-z]{5}" name="zip" [(ngModel)]="zip">
</form>
```

Template driven forms – Angular validatie

- Angular validatie heeft een deel dynamische booleans die gebruikt kunnen worden om de status van het formulier te raadplegen:
 - Valid & invalid: voldoet aan validatie
 - Touched & untouched: is de gebruiker in het veld geweest
 - Dirty & pristine: heeft de gebruiker wijzigingen aangebracht aan het veld
- Deze booleans kunnen geraadpleegd worden via het form object:

```
<p>
Form valid: {{mijnForm.form.valid}} <br/>
Form touched: {{mijnForm.form.touched}}
</p>
```

Template driven forms – Angular validatie

- De status kan ook geraadpleegd worden op veld niveau in het formulier.
 - Template variabele voorzien:

```
<input type="text" name="name" [(ngModel)]="name" required  
#firstname="ngModel"/> <br/>
```

- Template variabele raadplegen in view:

```
Name veld valid: {{firstname.valid}} <br/>  
Name veld touched: {{firstname.touched}} <br/>  
Name veld dirty: {{firstname.dirty}}
```

Template driven forms – Angular validatie

- Er worden dynamisch css classes gekoppeld aan velden op basis van de status. Onderstaande classes kunnen gebruikt worden:

Control status	Class als waar	Class als onwaar
Input veld is bezocht (focus)	ng-touched	ng-untouched
Waarde is gewijzigd	ng-dirty	ng-pristine
Waarde is 'valid'	ng-valid	ng-invalid

- CSS classes kunnen op basis van deze zaken aangepast worden:

```
input.ng-invalid{ border: 2px solid red}  
input.ng-valid{ border: 2px solid green}
```

Template driven forms – Form submit

- mijnForm.form.valid kan ook gebruikt worden in combinatie met bv. [disabled]

```
<button type="submit"  
[disabled]="!mijnForm.form.valid">Submit</button>
```

- De submit knop wordt zichtbaar als de boolean valid true is
- Bij grote formulieren kan een klasse object gekoppeld worden aan een formulier voor een beter overzicht van data.

Template driven forms – Classes

```
// registration.class.ts
export class Registration {
    name: string;
    email: string;
}
```

```
// form2.component.ts
export class Form2Component {
    constructor(private reg: Registration) { }
    onSubmit(data){
        console.log("Submitted:" + JSON.stringify(reg));
    }
}
```

```
<!-- form2.component.html -->
<input type="text" name="name" [(ngModel)]="registration.name" />
<input type="email" name="email" [(ngModel)]="registration.email" />
```

Template driven forms – Input types

- Verschillende input types kunnen gebruikt worden:
 - Input velden:
 - Tekst
 - Email
 - Checkbox
 - ...
 - Dropdown menu's:

```
<select name="product" [(ngModel)]=“product”>  
  <option *ngFor=“let item of products” [value]=“item”>{{item}}</option>  
</select>
```

```
//in component klasse:  
products: string[] = [“Laptop”, “Tablet”, “Phone”, “Watch”];
```

Template driven forms – Input types

- Verschillende input types kunnen gebruikt worden:
 - Datum velden:

```
<input type="date" [(ngModel)] = "dateStr" name="dateStr"
```

- Radio buttons:

```
<input id="one" type="radio" value="one" name="choice"
[(ngModel)]=“choice”><label for="one">One</label>
<input id="two" type="radio" value="two" name="choice"
[(ngModel)]=“choice”><label for="two">Two</label>
<input id="three" type="radio" value="three" name="choice"
[(ngModel)]=“choice”><label for="three">Three</label>
```

Model driven forms

- Ook wel reactive forms genoemd
- Form control objecten worden gemanipuleerd vanuit de component klasse
- De vormgeving / validatie gebeurt niet meer volledig in de HTML, maar meer in de component klasse,
- De component klasse kan kijken naar veranderingen in het formulier om vervolgens erop te “reageren” (=reactive).

Model driven forms

- Om modeldriven forms te gebruiken, moet de ReactiveFormsModuleModule toegevoegd worden aan de app.module.ts file:

```
import { FormsModule, ReactiveFormsModule } from '@angular/forms';

@NgModule({
  ...
  imports: [
    BrowserModule,
    FormsModule,
    ReactiveFormsModule
  ],
  ...
})
```

Model driven forms - voorbeeld

- In volgend voorbeeld gebruiken we een FormGroup
- Input velden worden gekoppeld aan de FormGroup in de component klasse
- Een functie wordt gekoppeld aan de submit button
- Data van de velden wordt meegegeven aan de FormGroup

Model driven forms - voorbeeld

- In de component.ts file worden 3 imports voorzien:

```
import { FormGroup, Validators, FormControl } from '@angular/forms';
```

- In de component klasse wordt een FormGroup object aangemaakt en gedefinieerd in de ngOnInit methode:

- De namen van de FormControls worden gebruikt in het formulier
- De FormControl krijgt 2 argumenten:
 - Een default value: waarde of null
 - Eén validator of een array met validators

Model driven forms - voorbeeld

```
export class ModelformComponent implements OnInit {
  myForm: FormGroup;

  ngOnInit(): void {
    this.myForm = new FormGroup({
      first: new FormControl('Dries', []),
      last: new FormControl('Swinnen', [Validators.required])
    });
  }
}
```

Model driven forms - voorbeeld

- In de component.html file voorzien we het formulier met koppeling naar de FormGroup
- Elk element krijgt een verwijzing naar de FormControl in de FormGroup

```
<form [formGroup]="myForm">
First name: <input formControlName="first"> <br/>
Last name: <input formControlName="last"> <br/>
<input type="button" value="Submit" (click)=onSubmit() />
</form>
```

Model driven form - voorbeeld

- De onSubmit methode wordt voorzien in de component klasse:

```
onSubmit() {  
    console.log('Submitted: ' +  
              JSON.stringify(this.myForm.value, null, 2));  
}
```

- myForm verwijst naar de FormGroup. Je kan FormControl's ook afzonderlijk opvragen:

```
console.log('Submitted: ' + this.myForm.get('first').value);  
console.log('Submitted: ' + this.myForm.get('last').value);
```

Model driven form - validatie

- Validatie wordt niet in de HTML code voorzien maar in de declaratie van de FormControl objecten
- Eén of meerdere validators kunnen gekoppeld worden
- Volgende build-in validators zijn beschikbaar:
 - required
 - minLength
 - maxLength
 - Pattern
- Zelf validators schrijven is mogelijk

Model driven form - validatie

- Validators worden toegevoegd aan de FormControl:

```
this.myForm = new FormGroup({
  first: new FormControl('Dries', [Validators.required,
Validators.minLength(3)],
  last: new FormControl('Swinnen',
[Validators.pattern('[A-Z][a-z]*')]),
```

- Pattern kan zowel een regex zijn of een string met een regex (zie voorbeeld lab)
- Referentie in de view:

```
<input type="text" name="name" placeholder="Name" formControlName="name">
<p *ngIf="form.get('name').hasError('minlength')" class="error">Minimum 3 characters.</p>
<p *ngIf="form.get('name').hasError('required')" class="error">Required.</p>
```

Model driven form – Custom validator

- Het is ook mogelijk om custom validators te maken:

```
//File: email.validator.ts

import { FormControl } from '@angular/forms';

export function validateEmail(control: FormControl) {
    if (typeof (control.value) === 'string') {
        if (control.value.includes('@')) {
            return null;
        } else {
            return { validateEmail: { valid: false } };
        }
    }
}
```

Model driven form – Custom validator

- Email validator gebruiken in de FormGroup:

```
//File: modelform.component.ts
import { validateEmail } from './email.validator';

this.myForm = new FormGroup({
  first: new FormControl('Dries', []),
  last: new FormControl('Swinnen', []),
  email: new FormControl(null, [ validateEmail ]),
})
```

Model driven form – SubFormGroups

- Het is mogelijk om een formulier op te splitsen in verschillende SubFormGroups.
- Hierdoor krijgt je this.myForm.value aparte objecten voor de subFormGroup:

```
<form [formGroup]="myForm">
  First name: <input type="text" formControlName="first"> <br/>
  Last name: <input type="text" formControlName="last"> <br/>
  Email: <input formControlName="email"> <br/>
  <div formGroupName="address">
    Address: <input formControlName="street"> <br/>
    City: <input formControlName="city"> <br/>
  </div>
  <input type="button" value="Submit" (click)=onSubmit() [disabled]=!"myForm.valid" />
</form>
```

```
Submitted: {
  "first": "Dries",
  "last": "Swinnen",
  "email": "qzefoij@qzefoij",
  "address": {
    "street": "zqefoij",
    "city": "zebo"
  }
}
```

Model driven form – SubFormGroups

- Het is mogelijk om een formulier op te splitsen in verschillende SubFormGroups.
- Hierdoor krijgt je this.myForm.value aparte objecten voor de subFormGroup:

```
this.myForm = new FormGroup({  
  first: new FormControl('Dries', []),  
  last: new FormControl('Swinnen', []),  
  email: new FormControl(null, []),  
  address: new FormGroup({  
    street: new FormControl(null, []),  
    city: new FormControl(null, [])  
  })  
});
```

```
Submitted:  
  "first": "Dries",  
  "last": "Swinnen",  
  "email": "qzefoij@qzefoij",  
  "address": {  
    "street": "zqefoij",  
    "city": "zefo"  
  }  
}
```

Model driven forms - FormBuilder

- Een FormBuilder kan gebruikt worden om je code te vereenvoudigen
 - Aanspreken van constructor FormControl niet meer nodig.

```
export class FormBuilderComp {  
  form: FormGroup;  
  
  constructor(private fb: FormBuilder) {  
    this.form = fb.group({  
      name: fb.group({  
        first: ['Nancy', Validators.minLength(2)],  
        last: 'Drew',  
      }),  
      email: '',  
    });  
  }  
}
```

Services & Dependency injection (DI)



Service in Angular

- Implementatie gebruik makend van eenvoudige classes

```
Export class LogService{  
    log(message: string){  
        console.log(message);  
    }  
}
```

- Definitie in de module
- @Injectable annotatie voorzien

Services in Angular

- Worden gebruikt door Angular components:
 - Ophalen van data
 - Validatie
 - Logging
 - ...

Services in Angular

- Services zijn eenvoudige classes met een `@Injectable()` decorator
- Injectable importeren van `@angular/core`

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

@Injectable()
export class PetService {
  pets: string[] = ['Cat', 'Dog', 'Rabbit', 'Fish']

  getPets() {
    return this.pets;
  }
}
```

Services in Angular

- Central beheer van gegevens in een app
- Meerdere components kunnen met dezelfde services werken
- Singleton (op globaal niveau)
 - Services worden maar één keer geïnitialiseerd tijdens het opstarten van de app
 - De levensduur van de service is net zo lang als de levensduur van de app

Dependency Injection (DI) – Code Pattern

- Code pattern
- Gebruik van classes zonder hard-coded constructor calls:
 - Flexibele code

```
//Voorbeeld 1: klassieke method
export class Car {
    engine: Engine;
    tire: Tire;

    constructor(){
        this.engine = new Engine();
        this.tire = new Tire();
    }

}
```

```
//Voorbeeld 2: DI
export class Car {
    engine: Engine;
    tire: Tire;

    constructor(engine, tire){
        this.engine = engine;
        this.tire = tire;
    }

}
```

Dependency Injection (DI) – Design Pattern

- Stel dat bij het eerste voorbeeld de Engine constructor aangepast wordt (bv: meegeven parameters), gaat de Car klasse stuk. Bij voorbeeld 2 is dit niet het geval, omdat de Engine buiten de klasse aangemaakt wordt.

```
//Voorbeeld 1: klassieke method
export class Car {
    engine: Engine;
    tire: Tire;

    constructor(){
        this.engine = new Engine();
        this.tire = new Tire();
    }
}
```

```
//Voorbeeld 2: DI
export class Car {
    engine: Engine;
    tire: Tire;

    constructor(engine, tire){
        this.engine = engine;
        this.tire = tire;
    }
}
```

Dependency Injection (DI) – Design Pattern

```
//Voorbeeld 1: klassieke method  
export class Car {  
    engine: Engine;  
    tire: Tire;  
  
    constructor(){  
        this.engine = new Engine();  
        this.tire = new Tire();  
    }  
  
}  
// car is afhankelijk van de constructor  
// en opbouw van de classes Engine & Tire  
// er kunnen geen bestaande / oude objects  
// van car & engine gebruikt worden.  
  
car1: Car = new Car();
```

```
//Voorbeeld 2: DI  
export class Car {  
    engine: Engine;  
    tire: Tire;  
  
    constructor(engine, tire){  
        this.engine = engine;  
        this.tire = tire;  
    }  
  
}  
  
// car is onafhankelijk van het aanmaken  
// van engine & tire  
engine1: Engine = new Engine();  
tire1: Tire = new Tire();  
car1: Car = new Car(engine1, tire1);
```

Dependency Injection (DI) - Angular

- Angular heeft ingebouwde dependency Injection
- Services worden via de constructor meegegeven aan components
- Injectors detecteren de services en voorzien de nodige dependencies
- Injectors zorgen voor de initialisatie van de service.

Dependency Injections (DI) - Angular

- Injectors bestaan op globaal niveau
 - Worden voorzien in app.module.ts

```
@NgModule({  
    imports: [...],  
    declarations: [...],  
    providers: [PetService], //Kan ook toegevoegd worden aan @Component  
})
```

- Injectors bestaan op component niveau
 - Worden voorzien in de x.component.ts

```
@Component ({  
    selector: 'app-component',  
    ...  
    providers: [PetService], //Kan ook toegevoegd worden aan @NgModule  
})
```

Dependency Injection (DI) - Angular

- Het initialiseren van de dependencies (Engine & Tire) wordt afgehandeld door het Angular framework. (Injectors)
- Services worden geïnjecteerd via DI in de component constructor

```
import { Component, OnInit } from '@angular/core';
import { PetService } from './pet.service';
@Component({...})
export class AppComponent implements OnInit {
  pets: string[];
  constructor(private petsrv: PetService) { }
  ngOnInit() {
    this.pets = this.petsrv.getPets();
  }
}
```

Dependency Injection (DI) - Angular

- Indien een component dependency injection toepast en de service niet kan vinden, gaat hij kijken in de injectors van de parent
 - Indien deze de service ook niet kent -> parent
 - Tot aan de root injector (app.module.ts)
- Services globaal declareren: gedeeld over alle componenten
 - In app.module.ts
- Services lokaal declareren: één instantie voor die component
 - In de component

Dependency Injection (DI) - Samengevat

- Services zijn eenvoudige classes
 - met de `@Injectable()` decorator
- Services worden geregistreerd bij een Injector
 - Op `app.module.ts` niveau
 - Op component niveau

Dependency Injection (DI) - Samengevat

- Injectors kunnen enkel services injecteren die ze ‘kennen’. Deze worden toegevoegd aan de providers array in de app.module.ts file (globaal) of in de component.

```
@NgModule({  
    imports: [...],  
    declarations: [...],  
    providers: [PetService], //Kan ook toegevoegd worden aan @Component  
    bootstrap: [...]  
})
```

- Injectors werken in een hiërarchie. Als een injector een service niet kent, gaat hij kijken bij de parent.
 - Tot dat hij aan de root injector komt (app.module.ts)

Dependency Injection (DI) - Angular

- Meer info:
 - <https://angular.io/guide/dependency-injection>

(Reusable) forms & services - Voorbeeld

Add event

Name:

Speaker:

Location:

Level:

Name	Speaker	Location	Level	Update
Angular fundamentals	Dries Swinnen	Hasselt	1	<input type="button" value="Update"/>
Angular Advanced	Luc Doumen	Zonhoven	2	<input type="button" value="Update"/>
Angular Expert	Steve Jobs	Neerpelt	3	<input type="button" value="Update"/>

https://github.com/PXL-WebExpert-2018/CH7_Voorbeeld1

Data met HTTP



HTTP client

- De Angular HTTP client
 - Wordt gebruikt voor communicatie naar een webservice / API
 - Ondersteuning voor:
 - Maken van HTTP requests (GET, POST, PUT, PATCH, ...)
 - Werken met requests en response headers
 - Asynchroon programmeren
 - Maakt gebruik van [RxJS](#) async library

De HTTP client gebruiken

- De HttpClient is een service die geïnjecteerd wordt in classes
- Voorzien in de constructor van de services:

```
import { HttpClient } from '@angular/common/http';

@Injectable()
class myService{
  constructor(private http: HttpClient) {
  }
}
```

De HTTP client gebruiken

- Om gebruik te kunnen maken van HttpClient, moet de HttpClientModule geïmporteerd worden in de app.module.ts:

```
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  declarations: [
    ...
  ],
  imports: [
    BrowserModule,
    HttpClientModule
  ],
})
```

De HTTP client gebruiken

- Bijkomend worden verschillende imports uit de RxJS library geïmporteerd
 - In de service waar de http service geïnjecteerd wordt

```
import { HttpClient, HttpHeadersResponse } from '@angular/common/http';
import { Observable, throwError } from 'rxjs'
import { catchError, tap, map } from 'rxjs/operators'
```

RxJS



RxJS

- Reactive programming in JavaScript
- Asynchrone datastreams aan de hand van observables
 - UI Events
 - **HTTP requests**
 - File systems
 - **Array-achtige objecten**
 - Cache
- Wordt toegepast binnen de HttpClient module van Angular
- Voorkennis niet vereist, maar wel een meerwaarde

RxJS - ReactiveX

- Niet enkel beperkt tot JS:

Java: RxJava

Javascript: RxJS

C#(unity): UniRx

C++: RxCpp

RxJs - Datastreams

- Datastreams zijn lopende events op een tijdlijn met
 - Data
 - Errors
 - Complete signaal (optioneel)



- **Observables** worden gebruikt om deze streams te bekijken en om functies uit te voeren wanneer er een value / error of complete signal binnenkomt

RxJS - Observables

- Bij het uitvoeren van een HTTP request wordt een **Observable** object teruggegeven
- Observables kijken constant naar streams voor nieuwe data
- Een observer kan zich “**subscribe**n” op een Observable object
 - Interactie lijkt op die van een array
 - **Bij het subscribeen op een subscriber wordt de data pas binnengehaald**

RxJS - Observables

- Onderstaande functie returned een Observable
 - De map operator wordt verder uitgelegd

```
getContactList(): Observable<any> {  
    return this.http.get(BASEAPIURL);  
}
```

- De http request wordt pas uitgevoerd bij het subscriben op de observable ergens in onze code:

```
fetchContactList(): void {  
    this.service.getContactList().subscribe(data =>  
    {  
        this.contactList = data;  
    });  
}
```

RxJS – operators

- Operators zijn functies die bovenop observables kunnen draaien om collections te manipuleren
- Verschillende operators kunnen gelinkt worden met de pipe() functie

AREA	OPERATORS
Creation	from, fromPromise, fromEvent, of
Combination	combineLatest, concat, merge, startWith, withLatestFrom, zip
Filtering	debounceTime, distinctUntilChanged, filter, take, takeUntil
Transformation	bufferTime, concatMap, map, mergeMap, scan, switchMap
Utility	tap
Multicasting	share

RxJS – tap operator

- Wordt gebruikt om voor elke value uit de observable bepaalde functies uit te voeren
- Geeft op het einde de observable door naar de volgende functie
- **Manipuleert de data in de observable niet**

```
getMovies(): Observable<Movie[]>{
  return this.http.get<Movie[]>(this.moviesUrl).pipe(
    tap(obj => console.log('fetched movies')),
    catchError(this.handleError)
  );
}
```

RxJS – map operator

- De Map operator voert een functie uit op elk item dat binnenkomt van een observable.
- De Map operator geeft een observable terug met het resultaat van die functie
- **Manipuleert de data in de observable als deze in die functie aangepast wordt**

RxJS – map operator

- http.get geeft altijd een observable terug
- De map operator zet bij elke array die uit de observable komt de title property uit het eerste object om naar ‘test map’
- Geeft vervolgens de aangepaste array terug via een observable

```
getMovies(): Observable<Movie[]>{
    return this.http.get<Movie[]>(this.moviesUrl).pipe(
        tap(movieArr => console.log('fetched movies')),
        map(movieArr => {
            movieArr[0].title = 'test map';
            return movieArr;
        }),
        catchError(this.handleError)
    );
}
```

RxJS – map operator

- Meerdere maps kunnen elkaar opvolgen:

```
getNumbers(): Observable<any> {
    return this.http.get(BASEAPIURL).pipe(
        map(res => res * 2),
        map(res => res - 1)
    );
}
```

- Maps kunnen functies bevatten:

```
getContactList(): Observable<Contact[]> {
    return this.http.get(BASEAPIURL).pipe(
        map(this.parseContactData)
    );
}

parseContactData(rawContacts: any[]): Contact[] {
    return ...
}
```

RxJS – fouten opvangen

- Pipen de observable naar de RxJS catchError() operator
- Deze operator geeft een observable terug
- Voorzien van nodige imports

```
import { HttpClient, HttpErrorResponse } from '@angular/common/http';
import { Observable, throwError } from 'rxjs'
import { catchError } from 'rxjs/operators'
```

RxJS – fouten opvangen

- Bij het uitvoeren van een httpRequest Pipen we de observable door naar de catchError operator:

```
getMovies(): Observable<Movie[]>{
  return this.http.get<Movie[]>(this.moviesUrl).pipe(
    catchError(this.handleError)
  );
}
```

- Deze voert de functie handleError uit die een object van het type HttpErrorResponse meekrijgt
 - Deze functie voorzien we zelf in de service

RxJS – fouten opvangen

```
private handleError(error: HttpErrorResponse) {
  if (error.error instanceof ErrorEvent) {
    // A client-side or network error occurred.
    console.error('An error occurred:', error.error.message);
  } else {
    // The backend returned an unsuccessful response code.
    // The response body may also contain errors.
    console.error(
      `Backend returned code ${error.status}, ` +
      `body was: ${error.error}`);
  }
  return throwError(
    'Something bad happened; please try again later.');
};
```

RxJS – Gimme more please [extra]

- <https://angular.io/guide/rx-library>
- <https://gist.github.com/staltz/868e7e9bc2a7b8c1f754>
- https://www.youtube.com/watch?v=Tux1nhBPl_w
- <https://app.pluralsight.com/library/courses/rxjs-getting-started>

Data ophalen uit een lokaal JSON bestand



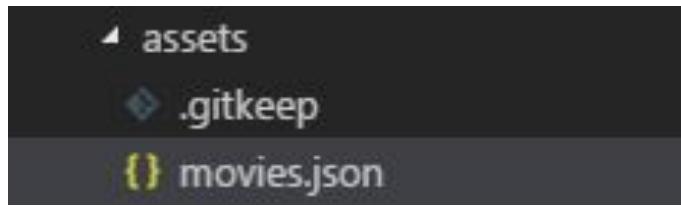
De HTTP client – lokale JSON

- Zie voorbeeld: Resources/Voorbeeld: lokale JSON
- In onze applicatie maken we gebruik van volgende model:

```
export class Movie {  
    title: string;  
    rating: number;  
    isReleased: boolean;  
  
    constructor(t: string, r: number, isR: boolean) {  
        this.title = t;  
        this.rating = r;  
        this.isReleased = isR;  
    }  
}
```

De HTTP client – lokale JSON

- In onze app plaatsen we in de assets folder een JSON file:



```
[  
  {  
    "title": "It",  
    "rating": 5,  
    "isReleased": true  
  },  
  {  
    "title": "Pitch perfect 3",  
    "rating": 7,  
    "isReleased": true  
  },  
  {  
    "title": "Star wars: The last jedi",  
    "rating": 8,  
    "isReleased": true  
  },  
  {  
    "title": "Aquaman",  
    "rating": 0,  
    "isReleased": false  
  },  
  {  
    "title": "Mary Poppins returns",  
    "rating": 0,  
    "isReleased": false  
}
```

De HTTP client – lokale JSON

- Importeren van de HttpClientModule in de app.module.ts file
- Injecteren van de HttpClient in de constructor van de service
- Voorzien van de nodige imports
- Voorzien van een link naar het JSON bestand

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpErrorResponse } from '@angular/common/http';
import { Observable, throwError } from 'rxjs'
import { catchError, tap, map } from 'rxjs/operators'
import { Movie } from '../shared/movie.model';

@Injectable()
export class MovieService {
    moviesUrl = 'assets/movies.json';

    constructor(private http: HttpClient) { }
```

De HTTP client – lokale JSON

- Methode getMovies wordt aangemaakt in de service:

```
getMovies(): Observable<Movie[]>{  
    return this.http.get<Movie[]>(this.moviesUrl);  
}
```

- De methode geeft een Observable terug met een Movie array.
- Omdat ons Model dezelfde structuur & properties heeft als het json object movies, kan er een automatische conversie gebeuren naar onze klasse
 - Indien dit niet is, kan je ook nog manipuleren via de map operator (zie lab)

De HTTP client – lokale JSON

- In de app.component.ts voorzien we een lokale variabele movieList van het type Movie[].
- In de ngOnInit methode subscriben we op de observable die teruggegeven wordt door de getMovies() methode uit de service:

```
export class AppComponent implements OnInit {
  movieList: Movie[];

  constructor(private serv: MovieService) { }

  ngOnInit() {
    this.serv.getMovies().subscribe(data => this.movieList = data);
  }
}
```

De HTTP client – lokale JSON

- In de HTML component gebruiken we een ngFor om te itereren over de lokale array.

```
<h1>Movie table from json file</h1>
<table>
  <tr>
    <th>Movie title</th><th>Rating</th><th>Released</th>
  </tr>
  <tr *ngFor="let movie of movieList">
    <td>{{movie.title}}</td><td>{{movie.rating}}</td><td>{{movie.isReleased}}</td>
  </tr>
</table>
```

Movie title	Rating	Released
It	5	true
Pitch perfect 3	7	false
Star wars: The last jedi	10	true

Data ophalen via een externe API



De HTTP client – Aanspreken van firebase

- Zie voorbeeld: Resources/Voorbeeld 2: Firebase
- Algemene documentatie is beschikbaar via onderstaande link
 - <https://firebase.google.com/docs/database/rest/start>
- API URL is beschikbaar in het databases venster:



De HTTP client – Aanspreken van firebase - GET

- In De service voorzien we volgende API url:

```
const BASE_API_URL = 'https://webexpert-games.firebaseio.com/';
```

- De get Request wordt toegevoegd in de methode getGames

```
getGames(): Observable<Game[]> {
  return this.http.get(BASE_API_URL + 'games.json').pipe(
    map(res => this.parseData(res)),
    catchError(this.handleError)
  );
}
```

- Deze request krijg alle games uit Firebase. De methode parseData zorgt voor de mapping naar het model Game.
 - Firebase geeft een deel JSON objecten terug, geen array

De HTTP client – Aanspreken van firebase - GET

- Object.keys haalt alle keys uit het JSON object in array vorm
- De map maakt voor elke key in de array een Game object aan
- De methode geeft een array van Games terug

```
parseData(json: any): Game[] {  
    return Object.keys(json).map(key => {  
        let game = new Game(json[key].title, json[key].publisher, json[key].rating, key);  
        return game;  
    });  
}
```

De HTTP client – Aanspreken van firebase – GET met parameters

- Het is mogelijk om requests options mee te geven aan requests
- Hiervoor gebruiken we de types Headers en RequestOptions als volgt:

```
getData(){
    let headers = new Headers();
    headers.append("Content-Type", "application/json");
    let reqOpts = new RequestOptions(headers: headers);
    return http.get('app/data.json', reqOpts);
}
```

De HTTP client – Aanspreken van firebase - POST

- Worden gebruikt om data naar de server te sturen (data toevoegen)
- Syntax:

```
post(url: string, body: any, options?: RequestOptionsArgs):  
Observable<Response>
```

- Voorbeeld:

```
addGame(game: Game) {  
    return this.http.post(BASE_API_URL + 'games.json', game).pipe(  
        catchError(this.handleError)  
    );  
}
```

- Game object wordt automatisch omgezet naar JSON

De HTTP client – Aanspreken van firebase - DELETE

- Wordt gebruikt om data van de server te verwijderen
- Syntax:

```
delete(url: string, body: any, options?: RequestOptionsArgs):  
Observable<Response>
```

- Voorbeeld:

```
deleteGame(game: Game) {  
    return this.http.delete(BASE_API_URL + 'games/' + game.id + '.json').pipe(  
        catchError(this.handleError)  
    );  
}
```

De HTTP client – Aanspreken van firebase - PUT

- Put wordt gebruikt om data aan te maken of aan te passen als geheel object
- Syntax:

```
put(url: string, body: any, options?: RequestOptionsArgs):  
Observable<Response>
```

- Voorbeeld:

```
editGamePut(id: string, newGame: Game) {  
    return this.http.patch(BASE_API_URL + 'games/' + id + '.json', newGame).pipe(  
        catchError(this.handleError)  
    );  
}
```

De HTTP client – Aanspreken van firebase - PATCH

- Patch wordt gebruikt om een gedeelte van een bestaand object te updaten
- Syntax:

```
patch(url: string, body: any, options?: RequestOptionsArgs):  
Observable<Response>
```

- Voorbeeld:

```
editGamePatch(id: string, property: any) {  
    return this.http.patch(BASE_API_URL + 'games/' + id + '.json', property).pipe(  
        catchError(this.handleError)  
    );  
}
```

Handige functies



De async pipe

- Om data asynchroon in de view in te laden kan je gebruik maken van de async pipe
- Handig bij grote hoeveelheden data, doorlopende streams
- De async pipe verwacht een Observable, dus je dient hier in de component niet op te subscriben!
 - De async pipe zorgt zelf voor het subscriben

De async pipe

```
@Component({
  selector: 'app-root',
  template: `
    <ul>
      <li *ngFor="let game of gamesObservable | async">{{ game.title }}</li>
    </ul>
  `,
})
export class AppComponent implements OnInit {
  gamesObservable: Observable<Game[]>

  constructor(private gameService: GameService) { }

  ngOnInit() {
    // Returned een observable, GEEN SUBSCRIBE!
    this.gamesObservable = this.gameService.getGames();
  }
}
```

Pipes

- Manier om data te manipuleren
- Pipe symbool: |
- Pipes combineren is mogelijk
- Built in pipes:
 - uppercase
 - lowercase
 - date
 - currency
 - Number
 - ...

```
 {{ name | lowercase}}  
 {{ birthday | date: 'MMMM dd,yyyy' | uppercase }}  
 {{ price | currency:'EUR':true }}  
 {{ 82.328|number: '2.0-2' }}
```

Pipes

- Zelf pipes maken is mogelijk
- Gebruik angularCLI

```
ng generate pipe capitalize
```

- Voorbeeld pipe om alles om te zetten naar hoofdletters

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'capitalize'
})
export class CapitalizePipe implements PipeTransform {

  transform(value: string, args?: any): any {
    return value.toUpperCase();
  }
}
```

Pipes

- Voeg de pipe toe als declaration in de app.module.ts file
 - AngularCLI doet dit automatisch
- Vervolgens kan je deze gebruiken in de template van je components

```
<ul>
| <li *ngFor="let game of gameList">{{game.title | capitalize }}</li>
</ul>
```

Http en filteren van data

- Filtering van data in frontend/backend afhankelijk van case
 - Bij voorkeur grote datasets in backend
- Filtering in de frontend bij kleine datasets / beperkingen van backend
 - **Geen gebruik maken van custom pipes (bad performance!)**
 - Filtering voorzien in service / component

```
getFilteredGames(searchStr: string){
    return this.http.get(BASE_API_URL + 'games.json').pipe(
        map(res => this.parseData(res)),
        map(res => {
            return res.filter(item => item.title.toLowerCase().includes(searchStr.toLowerCase()));
        }),
        catchError(this.handleError)
    );
}
```

Http en filteren van data

```
@Component({
  selector: 'app-root',
  template: `<input type="text" placeholder="search..." #term (keyup)="getGames(term.value)">
    <ul>
      <li *ngFor="let game of gameList$ | async">{{ game.title }}</li>
    </ul>`,
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  gameList$: Observable<Game[]>;

  constructor(private gs: GameService) { }

  ngOnInit(): void{
    this.getGames('');
  }

  getGames(term: string){
    this.gameList$ = this.gs.getFilteredGames(term);
  }
}
```

Http en filteren van data

```
export class AppComponent implements OnInit {
  searchString = new Subject<string>();
  gameList$: Observable<Game[]>;
  constructor(private gs: GameService) { }

  ngOnInit(): void{
    this.gameList$ = this.searchString.pipe(
      debounceTime(300),
      distinctUntilChanged(),
      switchMap(term => this.gs.getFilteredGames(term)),
    );
  }

  search(term): void{
    this.searchString.next(term);
  }
}
```

API's om mee te experimenteren

- [Riot Games](#)
- [import.io](#)
- [Spotify](#)
- [Telegram](#)
- [SoundCloud](#)
- [Reddit](#)
- [YouTube](#)
- [Wunderground](#)
- [Firebase](#)
- [Kandy](#)
- [Star Wars](#)
- [Marvel Comics](#)
- [Mashape](#)
- [Foaas](#)
- [BreweryDB](#)
- [Slack](#)
- [Geo Names](#)
- [Common Crawl](#)
- [Programmers API](#)
- [FitBit](#)
- [JawBone](#)
- [Steam](#)
- [Twilio](#)
- [IBM Watson](#)
- [Algolia](#)
- [Battle.net](#)
- [Free Geo IP](#)
- [The Counted](#)
- [Wolfram Alpha](#)
- [Github](#)
- [Twitter](#)
- [Nutritionix](#)
- [Pokémon API](#)
- [Open Weather Map](#)
- [Yodaspeak](#)
- [Recipe API](#)

In memory web API

Angular



Angular in memory web api

- In memory web api voor demo's & prototypes
- Nabootsen CRUD operaties via een Resty API
- Vangt Angular zijn Http & HttpClient requests op
- Controle over data die teruggegeven wordt
 - Handig voor testen

<https://github.com/angular/in-memory-web-api>

Installatie

- Vanuit je huidig project volgend commando uitvoeren:

```
npm install angular-in-memory-web-api --save
```

- Vervolgens toevoegen aan je app.module.ts file:

- InMemoryDataService is de naam van de mock service
- Delay is optioneel

```
import { HttpClientInMemoryWebApiModule } from  
'angular-in-memory-web-api';  
import { InMemoryDataService } from './in-memory-data.service';  
  
@NgModule({  
    imports: [  
        HttpClientModule,  
        HttpClientInMemoryWebApiModule.forRoot(InMemoryDataService, {  
            delay: 100 })  
    ]  
})
```

Gebruik in memory web API

- Zie [CH9 Voorbeeld1](#)
- Aanmaken mock service ‘InMemoryDataService’
 - Koppeling in app.module.ts

```
@Injectable()
export class InMemService implements InMemoryDbService {

    constructor() { }

    createDb() {
        const todo = [
            { id: 1, name: 'item 1' },
            { id: 2, name: 'item 2' },
            { id: 3, name: 'item 3' }
        ]

        return { todo };
    }

    genId(todos: any[]): number {
        return todos.length > 0 ? todos[todos.length - 1].id + 1 : 1;
    }
}
```

Gebruik in memory web API

- genId methode wordt overschreven
 - Mock service probeert zelf ID genereren
 - Dit lukt niet bij een lege array

```
genId(todos: any[]): number{  
    return todos.length > 0 ? todos[todos.length-1].id+1 : 1;  
}
```

- De mock service zal vervolgens HTTP requests opvangen en mock data gebruiken om deze te verwerken
 - Snel opzetten van applicaties zonder werkende backend
 - Handig bij het testen van services

Gebruik in memory web API

- Aanmaken van basis CRUD requests in onze TodoService
- Calls maken we naar de url ‘api/todo’
 - Service toevoegen aan serviceprovider

```
@Injectable()
export class TodoService {
  apiurl: string = 'api/todo';
  constructor(private http: HttpClient) { }
```

Gebruik in memory web API

- Mock API oproepen vanuit service

```
getTodos(): Observable<Object[]> {  
    return this.http.get<Object[]>('api/todo');  
}
```

- Verschillende endpoints beschikbaar:

GET	/api/todo
GET	/api/todo?name=j
GET	/api/todo/1
POST	/api/todo
PUT	/api/todo
DELETE	/api/todo/1

Gebruik in memory web API

The screenshot shows a user interface for a todo application. At the top, there is a green header bar containing a white input field labeled "todo item..." and a white button labeled "add". Below this, there is a list of three todo items, each represented by a light gray rectangular card. Each card contains the item number and text ("1. todo item 1", "2. todo item 2", "3. todo item 3"), followed by two small gray buttons labeled "edit" and "delete". At the bottom, there is another green header bar containing a white input field labeled "search...".

Index	Item	Actions
1	todo item 1	edit delete
2	todo item 2	edit delete
3	todo item 3	edit delete

Routing & Routeguards

Angular



Routing & Navigatie

- Applicaties met verschillende views hebben navigatie nodig
- Traditionele manier is met hyperlinks afgehandeld door de browser
- In web apps wordt de navigatie afgehandeld door de app zelf
 - Hangt niet af van de browser functionaliteit
- De component Router in Angular zorgt voor navigatie binnen een applicatie

Routing in Angular

- De Angular Component Router zorgt voor navigatie binnen de applicatie
- Navigatie kan op verschillende manieren:
 - Door het klikken op een (router)link
 - Via de code van een component / serviceprovider / ...

Router gebruiken

- Base href in index.html is verplicht

```
<head>
  <meta charset="utf-8">
  <title>RoutingExample</title>
  <base href="/">
```

- Aanmaken van een file voor het definiëren van routes
 - app.routes.ts
- Routes worden gedefinieerd in deze file als variabele
 - Variabele exporteren voor later gebruik

```
export const routes: Routes = [
  {path: 'page1', component: Page1Component}
];
```

Router gebruiken

- De volgende van de routes is belangrijk
- Redirects voorzien kan via de redirectTo property:

```
export const appRoutes: Routes = [
  {
    path: '',
    redirectTo: 'page1',
    pathMatch: 'full'
  },
]
```

- Opvangen van routes die niet bestaan kan als volgt
 - **Moet als laatste route gedefinieerd worden!**

```
{
  path: '**',
  component: Error404Component
}
```

Router gebruiken

- RouterModule inladen in app.module.ts
- Variabele uit app.route.ts inladen in app.module.ts

```
import { RouterModule } from '@angular/router';
import { appRoutes } from './app.routes';
```

- Routes variabele koppelen aan de RouterModule

```
imports: [
  BrowserModule,
  FormsModule,
  RouterModule.forRoot(appRoutes)
],
```

Router gebruiken

- Via de view:
 - Voorzien van URLs aan de hand van [routerLink]
 - routerLink bevat een array met properties van de route
 - Eerste item is de naam van de route
 - Deze zorgt voor de link naar de app.routes.ts file

```
<li><a [routerLink]=["/'"]>Home</a></li>
<li><a [routerLink]=["/about"]>About</a></li>
<li><a [routerLink]=["/contact"]>Contact</a></li>
```

- Het attribuut routerLinkActive="cssclass" kan gebruikt worden om een css klasse toe te voegen aan de hyperlink als de route actief is

Router gebruiken

- Via de code van een Component
 - Importeren van de Router uit @angular/router
 - Injecteren via de constructor
 - Oproepen in bijvoorbeeld een methode

```
constructor(private router: Router) { }

onClick() {
  this.router.navigate(['/contact']);
}
```

Router gebruiken

- Voorzien van een router-outlet
- De router-outlet zorgt voor het renderen van de components die overeenkomen met de route
- Meestal aanwezig in app.component.html

```
<nav>
  <ul>
    <li><a [routerLink]="/">Home</a></li>
    <li><a [routerLink]="/about">About</a></li>
    <li><a [routerLink]="/contact">Contact</a></li>
  </ul>
</nav>
<div class="content">
  <router-outlet></router-outlet>
</div>
```

Data doorgeven met Routes

- Werken met parameters in een route
 - Vaak wordt een id of index van een record doorgegeven in de URL
- Syntax:

```
{ path: 'person/:index',
  component: PersonDetailComponent}
```

- De route wordt in de browser vertaald naar
<http://localhost/person/3>

Data doorgeven met Routes

- Navigeren naar routes met parameters kan via:
 - Het [routerLink] attribuut vanuit de view

```
<ul>
  <li *ngFor="let x of list; let i=index">
    <a [routerLink]=["/person",i]">{{x.name}}</a>
  </li>
</ul>
```

- De router.navigate methode vanuit de controller

```
onClick() {
  this.router.navigate(['/person', this.index]);
}
```

Data doorgeven met Routes

- Data opvangen in de component
 - Injecteren van ActivatedRoute
 - Data ophalen:
 - Synchroon:

```
constructor(private route: ActivatedRoute, private peopleService: PeopleService) { }

ngOnInit() {
  let index = this.route.snapshot.params['index'];
  this.person = this.peopleService.getPerson(index);
}
```

Data doorgeven met Routes

- Data opvangen in de component
 - Injecteren van ActivatedRoute
 - Data ophalen:
 - Asynchroon:

```
ngOnInit() {  
    this.route.params.subscribe(params => {  
        let index = +params['index'];  
        this.person = this.peopleService.getPerson(index);  
    });  
}
```

Data doorgeven met Routes

- Meerdere parameters aan één route is ook mogelijk. In de route variabele:

```
{  
  path: 'page5/:par1/:par2',  
  component: Page5Component,  
  canDeactivate: [CanDeactivateConfirmGuard]  
},
```

- In de View om een link te leggen:

```
<li><a [routerLink]="/page4" routerLinkActive="active">Page 4 (activator)</a>  
<li><a [routerLink]="/page5,123,456" routerLinkActive="active">Page 5</a>
```

- De parameters opvragen kan terug met
router.snapshot.params['paramNaam']

Routeguards

- Routeguards zijn methodes die opgeroepen worden op bepaalde momenten in de levencyclus van de route
 - canActivate
 - canDeactivate
 - canLoad
 - canActivateChild
- Moeten gedefinieerd worden bij de providers in app.module.ts
- Moeten de `@Injectable()` decorator bevatten

Routeguards: canActivate

- Wordt geladen voor de activatie van de route (= voor het renderen van de component)
- Klasse die canActivate implementeert
- Bevat:
 - Een constructor: Dependency injection
 - Een methode canActivate:
 - Komt van de canActivate interface
 - Wordt uitgevoerd bij het uitvoeren van de routeguard
 - Geeft true of false terug

Routeguards: canActivate

- auth-activator.service.ts

```
import { Injectable } from '@angular/core';
import { CanActivate, Router } from '@angular/router';
import { AuthService } from '../services/auth.service';

@Injectable()
export class CanActivateViaAuthGuard implements CanActivate {

  constructor(private authService: AuthService, private router: Router) { }

  canActivate() {
    console.log(this.authService.isLoggedIn());
    if (!this.authService.isLoggedIn()) {
      this.router.navigate(['/login']);
    }
    return this.authService.isLoggedIn();
  }
}
```

Routeguards: canActivate

- Routeguard wordt toegevoegd aan de app.module.ts file:

```
@NgModule({
  declarations: [ ... ],
  imports: [ ... ],
  providers: [AuthService, CanActivateViaAuthGuard],
  bootstrap: [AppComponent]
})
```

- Routeguard wordt toegevoegd aan de app routes:

```
{
  path: 'page4',
  component: Page4Component,
  canActivate: [CanActivateViaAuthGuard]
},
```

- canActivate is een array, meerdere canActivate routeguards zijn dus mogelijk

Routeguards: canDeactivate

- Wordt geladen voor het deactiveren van de huidige route (= voor het verlaten van de component)
- Klasse die canDeactivate implementeert
- Bevat:
 - Een constructor: Dependency injection
 - Een methode canDeactivate:
 - Komt van de canDeactivate interface
 - Wordt uitgevoerd bij het uitvoeren van de routeguard
 - Geeft true of false terug

Routeguards: canActivate

- confirm-deactivator.service.ts

```
import { Injectable } from '@angular/core';
import { CanDeactivate, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';
import { Observable } from 'rxjs/Observable';
import { Page5Component } from '../page5/page5.component';

@Injectable()
export class CanDeactivateConfirmGuard implements CanDeactivate<Page5Component> {
  canDeactivate(component: Page5Component) {
    return window.confirm('Are you sure you want to leave?');
  }
}
```

Routeguards: canActivate

- Routeguard wordt toegevoegd aan de app.module.ts file
- Routeguard wordt toegevoegd aan de app routes:

```
{  
  path: 'page5',  
  component: Page5Component,  
  canActivate: [CanDeactivateConfirmGuard]  
},
```

- canActivate is een array, meerdere canActivate routeguards zijn dus mogelijk

Routeguards: canDeactivate - parameters

- Verschillende zaken kunnen meegegeven worden aan de canDeactivate methode:
 - De component
 - De route parameters
 - De route url
- Hierdoor is het mogelijk om bijvoorbeeld een deactivator te linken aan content in de component:

```
@Injectable()
export class CanDeactivateCustomGuard implements CanDeactivate<Page5Component> {
  canDeactivate(component: Page5Component, currentRoute: ActivatedRouteSnapshot,
    currentState: RouterStateSnapshot, nextState?: RouterStateSnapshot) {
    return component.compMethod();
  }
}
```



Resolvers

- Kunnen gebruikt worden om data op te halen bij het activeren van de route in plaats van na het renderen van de component (ngOnInit)
- Foutafhandeling voor het tonen van de component
 - Denk bv aan een foute id bij een detailcomponent

```
import { Injectable } from '@angular/core';
import { Resolve, ActivatedRouteSnapshot } from '@angular/router';
import { Contactsservice } from './contactsservice';

@Injectable()
export class ContactResolve implements Resolve<Contact> {

  constructor(private contactsservice: Contactsservice) {}

  resolve(route: ActivatedRouteSnapshot) {
    return this.contactsservice.getContact(route.paramMap.get('id'));
  }
}
```

Resolvers

- Worden ook toegevoegd aan de providers in app.module.ts
- In de components kan de data uit de activatedRoute gehaald worden

```
export const AppRoutes: Routes = [
  ...
  {
    path: 'contact/:id',
    component: ContactsDetailComponent,
    resolve: {
      contact: ContactResolve
    }
  }
];
```

Routing als aparte module

- Zie [CH9-Voorbeeld2](#)
- Best practise om routing te implementeren
 - Handig bij complexe routing bij applicaties die uit meerdere modules bestaan
- Zelf opbouwen ofwel gebruik maken van de AngularCLI

```
PS C:\Users\Dries\workspace> ng new testrouting  
? Would you like to add Angular routing? (y/N) |
```

Routing als aparte module

- Er wordt een aparte module ‘app-routing.module.ts’ gemaakt waarin routing gedefinieerd wordt:

```
const routes: Routes = [ ...  
  
@NgModule({  
    imports: [RouterModule.forRoot(routes)],  
    exports: [RouterModule],  
    providers: [CanActivateViaAuthGuard, CanDeactivateConfirmGuard]  
})  
export class AppRoutingModule { }  
|
```

- Koppeling doen we in de app.module.ts file:

```
@NgModule({  
    declarations: [ ...  
    imports: [  
        BrowserModule,  
        FormsModule,  
        AppRoutingModule  
    ],  
    providers: [AuthService],  
    bootstrap: [AppComponent]  
})  
export class AppModule { }
```

Kennismaking NodeJS



Kenmerken van NodeJS

- Javascript kan server-sided werken (= losgetrokken uit de browser)
- Maakt gebruik van event-driven architectuur (= code die reageert op events d.m.v. callback functies => gevolg: goede performance, uitvoering van een programma wordt nooit geblokkeerd omdat het programma wacht op een berekening. Callback functie wordt uitgevoerd als het resultaat van de berekening er is.
- Gemaakt voor het bouwen van snelle, schaalbare werkapplicaties
- Er is geen andere webserver nodig (apache, IIS, ...)

Kenmerken van NodeJS

- NodeJS is geschreven voor JavaScript
- NodeJS zelf is geschreven in C++
- Toegankelijk voor Windows, Linux en Mac
- Het moet apart worden geïnstalleerd
- Alle Node-toepassingen zelf zijn geschreven in JavaScript en hoeven dus maar één keer te worden geschreven. Nadien draaien ze op alle systemen



Kenmerken van NodeJS

- Geen DOM beschikbaar, wel requests & responses
- In plaats van DOM zal je vaak console.log()-meldingen schrijven
- Een applicatie die data retourneert, zal (bijna) altijd JSON-formaat gebruiken
- MEAN-stack:
 - M(mongoDB): database
 - A (Angular): framework voor de client-side toepassing
 - E (Express): Module die wordt ingezet om webserver, routing en API te maken (server-side)
 - N (NodeJS): De motor voor M & E (server-side)

Waarom NodeJS gebruiken

- Eén programmeertaal (JavaScript in de client en op de server)
- Code opnieuw gebruiken (server-side)
- Snellere ontwikkeltijd (NodeJS-apps moeten niet gecompileerd worden)
- Ondersteuning van de community

<https://nodejs.org/en>

Development omgeving

- Besturingssystemen
 - Windows 8.1 of Windows 10
 - Mac OS X 10.10+
 - Linux
- Editor
 - WebStorm
 - Visual studio (Code)
 - Atom
 - ...

Development omgeving



Node.js heeft geen interface

Onthoud: Node.js heeft geen user interface. Het draait puur om code. Het opsporen van fouten (debuggen) kan daarom lastig zijn. Want als een programma een fout bevat, wordt het afgebroken zodra de fout optreedt en kan Node.js alleen een foutmelding in de console schrijven. Hopelijk wordt u daar wat wijzer van, maar vaak is niet meer te achterhalen dan het regelnummer waar de fout is opgetreden. Een visuele debugger waarbij u breekpunten kunt zetten, variabelen en objecten kunt inspecteren en stapsgewijs door de code kunt lopen is dan handig. Daar gaan we in het volgende hoofdstuk op in.

NodeJS testen

- De installatie van node controleren via REPL:
 - Open een opdrachtvenster / terminal en typ node
 - Je komt nu in de REPL interface (Read, Evaluate, Print & Loop) waarin rechtstreeks JavaScript gebruikt wordt
 - Voorbeeld:
 - Afsluiten kan door 2x op ctrl+c te drukken



```
Black & White — node
PeterKassenaar — E
MacBook-Pro:~ PeterKassenaar$ node
> 2 + 2
4
> var x = 'Hello World'
undefined
> x
'Hello World'
>
```

Hello World in Node.js

- Download de voorbeelden uit Resources/CH10_Voorbeelden
- Open /01_helloworld.js

```
// eerste Node.js-bestand. Toon 'Hello World' in de console
var msg = 'Hello World';
console.log(msg);
```

- Je kan dit uitvoeren in het opdrachvenster



Hello World in Node.js

- Bekijk /02_helloworld2.js

```
console.log(2 + 2);
// werken met objecten
var persoon = {voornaam : 'Peter', achternaam : 'Kassenaar'};
console.log('Persoon: ', persoon.voornaam + ' ' + persoon.achternaam);
```

- Uitvoeren via opdrachtvenster:

```
Hello World
4
Persoon: Peter Kassenaar
```

Een eenvoudige webserver maken

- Bekijk /03_server.js
- **Stap 1 De module http laden**
 - Module http wordt beschikbaar gemaakt via de variabele server
 - createServer() heeft een functie als parameter, dewelke wordt aangeroepen als de server een verzoek van de browser ontvangt
 - De parameters request & response
 - **Request bevat alle gegevens van het binnenkomend verzoek**
 - **Response bevat de inhoud die wordt teruggegeven aan de browser**

```
var http = require('http');
var server = http.createServer(function (request, response) {
});
```

Een eenvoudige webserver maken

- **Stap 2 De webserver schrijven**

- Een header geeft aan dat plain text wordt teruggestuurd met als inhoud “hello world”

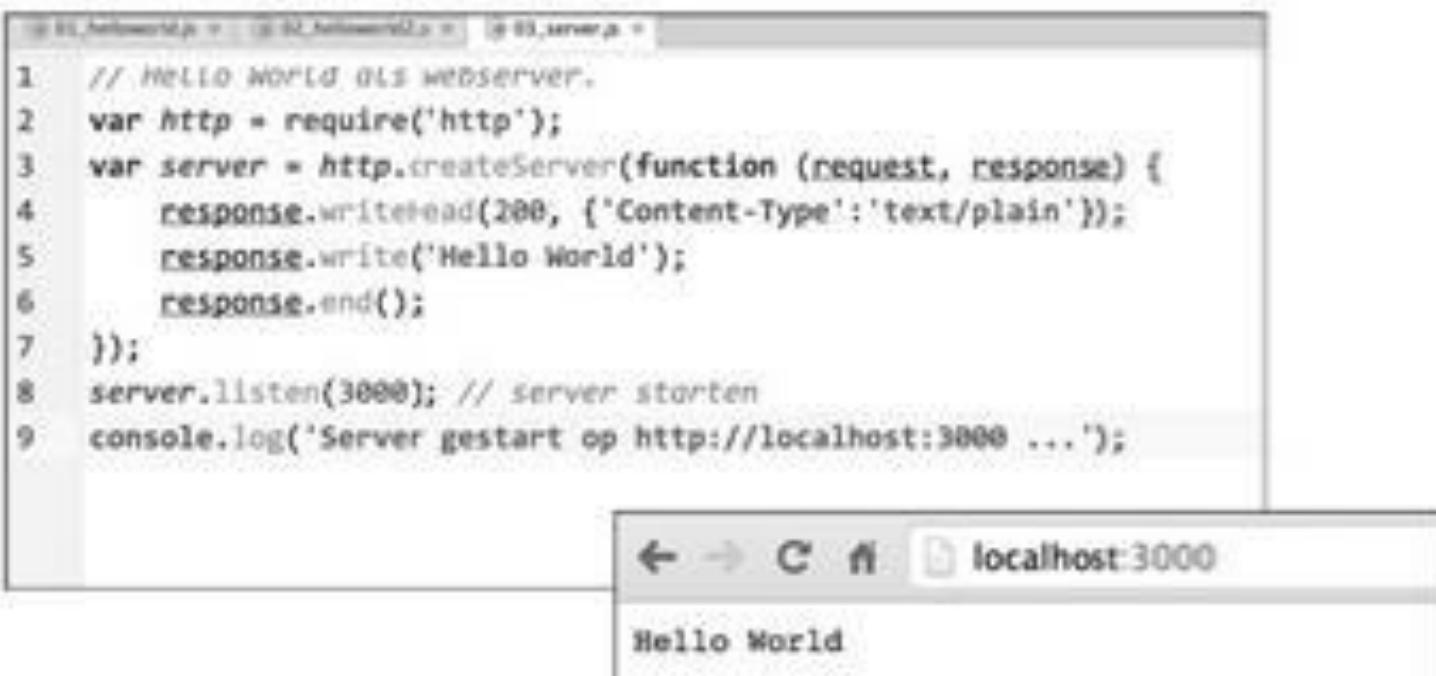
```
var server = http.createServer(function (request, response) {  
    response.writeHead(200, {'Content-Type': 'text/plain'});  
    response.write('Hello World');  
    response.end();  
});
```

- De server wordt gestart door te luisteren naar poort 3000 in onderstaande code (standaardpoort voor nodeApps)

```
server.listen(3000); // start webserver  
console.log('Server gestart op http://localhost:3000..');
```

Een eenvoudige webserver maken

- Open een browser en ga naar `http://localhost:3000`



The image shows a terminal window with two tabs. The active tab contains Node.js code for a simple web server:

```
// HELLO WORLD ALS WEBSERVER.  
var http = require('http');  
var server = http.createServer(function (request, response) {  
    response.writeHead(200, {'Content-Type': 'text/plain'});  
    response.write('Hello World');  
    response.end();  
});  
server.listen(3000); // Server starten  
console.log('Server gestart op http://localhost:3000 ...');
```

Below the terminal is a screenshot of a web browser window titled "localhost:3000". The address bar shows "localhost:3000" and the page content area displays "Hello World".

Een eenvoudige webserver maken



Asynchroon

Het programma dat u nu hebt geschreven is een perfect voorbeeld van de event driven en asynchrone werkwijze van Node.js. De functie `.createServer()` heeft een functie als parameter die pas wordt uitgevoerd als een event binnenkomt (namelijk: een nieuwe request). Als de functie is uitgevoerd keert hij weer terug in de wachtstand. Onderbreek het luisteren op poort 3000 met `Ctrl+C`.

Wat er nu ook gebeurt, bij elke request zal de server Hello World

terugsturen. U kunt probleemloos verschillende URL's testen:

`http://localhost:3000/index.html`, `http://localhost:3000/test.html`, `http://localhost:3000/blablabla`; het resultaat? Hello World!

Een eenouvdige webserver maken

- Bekijk /04_server2.js
- **Stap 3 HTML retourneren**
 - Aanpassingen van content type geeft vervolgens HTML als inhoud weer

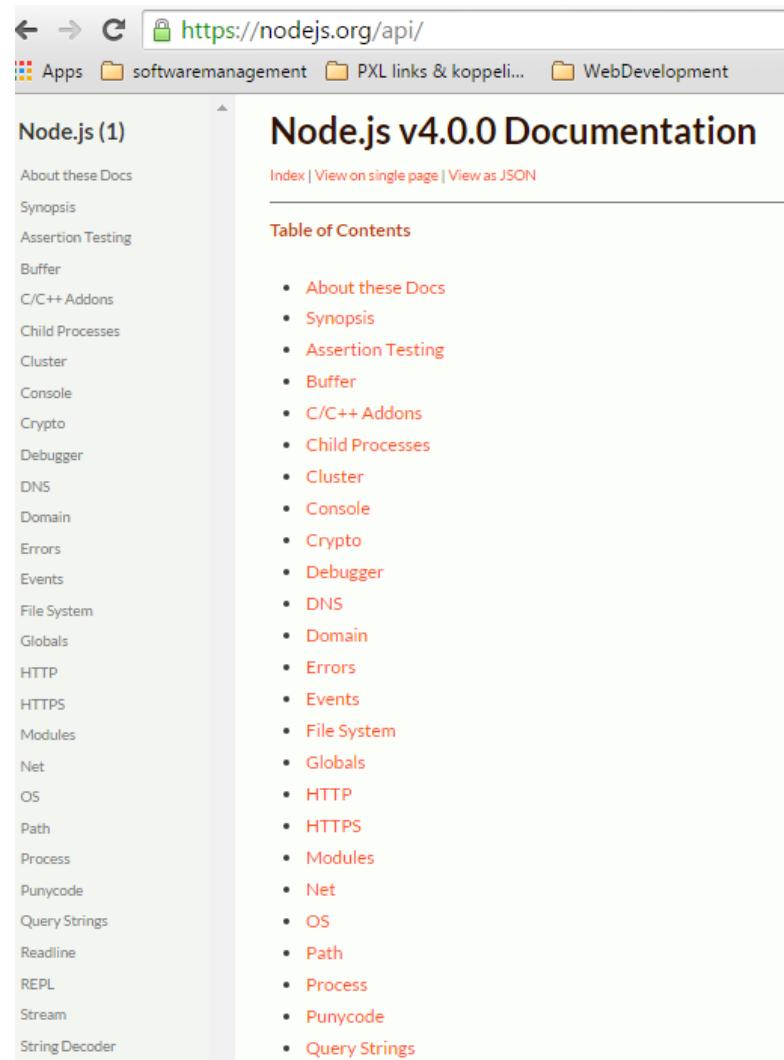
```
var server = http.createServer(function (request, response) {
  var url = request.url;
  response.writeHead(200, {'Content-Type': 'text/html'});
  response.write('<h1>De gevraagde URL: ' + url + '</h1>');
  response.end();
});
```

- Uitvoering in de browser



NodeJS-documentatie leren lezen

- Helpbestanden zijn online te vinden op
<https://nodejs.org/api/>



The screenshot shows a web browser window displaying the Node.js v4.0.0 Documentation. The URL in the address bar is <https://nodejs.org/api/>. The page title is "Node.js v4.0.0 Documentation". On the left, there is a sidebar titled "Node.js (1)" containing a list of API modules: About these Docs, Synopsis, Assertion Testing, Buffer, C/C++ Addons, Child Processes, Cluster, Console, Crypto, Debugger, DNS, Domain, Errors, Events, File System, Globals, HTTP, HTTPS, Modules, Net, OS, Path, Process, Punycode, Query Strings, Readline, REPL, Stream, and String Decoder. To the right, the main content area is titled "Table of Contents" and lists all the same modules from the sidebar, each preceded by a red bullet point.

NodeJS-documentatie leren lezen

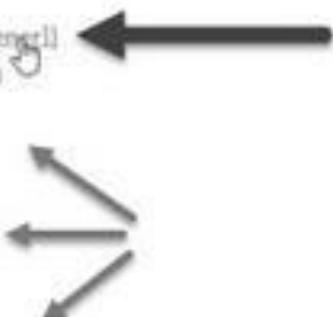
- De module “HTTP”

Table of Contents

- **HTTP**
 - `http.METHODS`
 - `http.STATUS_CODES`
 - `http.createServer([requestListener])`
 - `http.createClient([port], host)`
 - `Class: http.Server`
 - Event: 'request'
 - Event: 'connection'
 - Event: 'close'
 - Event: 'checkContinue'
 - Event: 'connect'
 - Event: 'upgrade'
 - Event: 'clientError'
 - `server.listen(port[, hostname][, backlog][, callback])`
 - `server.listen(path[, callback])`
 - `server.listen(handle[, callback])`

Klik door naar `http.createServer([requestListener])` en vervolgens naar `http.Server`. Daarna leest u dat elke request een instantie is van `http.IncomingMessage` en elke response een instantie is van `http.ServerResponse`.

- Klik ook door op deze klassen om te zien welke eigenschappen beschikbaar zijn en hoe ze worden gebruikt.



Afbeelding 2.14 Het detailoverzicht laat zien welke methodes, classes en events beschikbaar zijn in de module HTTP. Hier staat ook de functie `http.createServer()` beschreven die we hebben gebruikt.

NodeJS-documentatie leren lezen

- De module “HTTP” (vervolg)

The screenshot shows the official Node.js documentation for the `http.IncomingMessage` object. The title is highlighted with a rounded rectangle. Below the title, a detailed description is provided, followed by a section for the `'close'` event and a description of the `message.httpVersion` property.

http.IncomingMessage

An IncomingMessage object is created by `http.Server` or `http.ClientRequest`, and passed as the first argument to the 'request' and 'response' event respectively. It may be used to access response status, headers and data.

It implements the `ReadableStream` interface, as well as the following additional events, methods, and properties.

Event: 'close'

Function () { }

Indicates that the underlying connection was closed. Just like 'end', this event occurs only once per response.

message.httpVersion

In case of server request, the HTTP version sent by the client. In the case of client response, the HTTP version of the connected to server. Probably either '1.1' or '1.0'.

Afbeelding 2.15 Een request in Node.js is van het type `http.IncomingMessage`, zo blijkt uit de documentatie. Hier wordt beschreven welke eigenschappen dit object heeft.

NodeJS-documentatie leren lezen

- Een praktijkvoorbeeld “http.IncomingMessage”
 - Bekijk het bestand /05_Server3.js

```
var server = http.createServer(function (request, response) {
  console.log(request.headers);
```

Behalve `request.headers` zijn ook de eigenschappen `request.rawHeaders`, `request.httpVersion` en vele andere beschikbaar. Lees dit zelf in de documentatie!



Afbeelding 2.17 De headers van een willekeurige aanvraag zijn gelogd in de console.

NodeJS – Modules & packages



Node.js-modules en -packages

- Node gebruikt JS files voor applicaties
- Volledige applicaties zullen nooit in één file geschreven worden
⇒ Opsplitsing in “modules”
- Binnen een module geven we met `module.exports` aan welke modules voor de buitenwereld beschikbaar zijn.
- Waar de module gebruikt moet worden, schrijven we `require('./pad/naar/module')`
- De toevoeging .js is niet verplicht.

Node.js-modules en -packages



http is ook een module

Denk nog even terug aan de eenvoudige webserver uit het vorige hoofdstuk. Hierin hebt u al de regel `require('http')` gebruikt. Dit betekent dat in de applicatie de module http wordt ingeladen. Omdat dit een ingebouwde module is, hoeft u geen pad aan te geven. Later zult u meer in detail zien wanneer u wel- en geen padnaam opgeeft.

Node.js-modules en -packages

- Buiten modules wordt er ook veel gebruik gemaakt van package
- NPM: node **package** manager
- Packages zijn containers met een bundeling van allerlei losse modules die op deze manier samen een applicatie vormen.
- Elke module heeft zijn eigen referenties naar andere modules.
- Door het bestand package.json te installeren via npm worden alle modules (en daarvan afhankelijke modules) opgehaald

Praktijk: een logging module schrijven

- Herbruikbare logger: meldingen, fouten en algemene informatie naar de console schrijven
- Bekijk bestand /1006_logger/logger.js
 - Stap 1 De logger schrijven

```
// logger.js - exporteer verschillende algemene logging-methoden
module.exports.log = function(msg){
    console.log('=> Log: ' + msg);
};

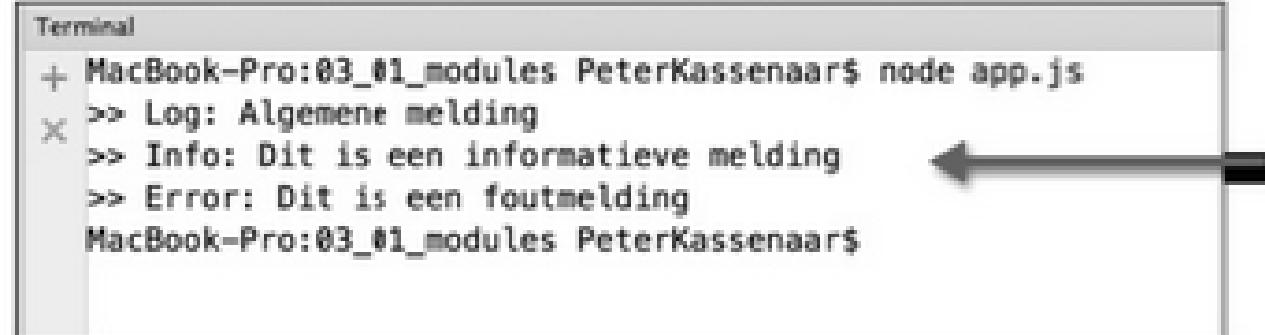
module.exports.info = function(msg){
    console.info('=> Info: ' + msg);
};

module.exports.error = function(msg){
    console.error('=> Error: ' + msg);
};
```

Praktijk: een logging module schrijven

- Bekijk bestand /1006_logger/app.js
 - Stap 2 De app schrijven

```
// app.js - laad de logger en test de drie functies.  
var logger = require('./logger');  
  
logger.log('Algemene melding');  
logger.info('Dit is een informatieve melding');  
logger.error('Dit is een foutmelding');
```



A screenshot of a terminal window titled "Terminal". The command "node app.js" is run, and the output shows three lines of text: "Log: Algemene melding", "Info: Dit is een informatieve melding", and "Error: Dit is een foutmelding". The terminal prompt "MacBook-Pro:03_01_modules PeterKassenaars\$" is visible at the bottom.

```
Terminal  
+ MacBook-Pro:03_01_modules PeterKassenaars$ node app.js  
x >> Log: Algemene melding  
x >> Info: Dit is een informatieve melding  
x >> Error: Dit is een foutmelding  
MacBook-Pro:03_01_modules PeterKassenaars$
```

Afbeelding 3.3 Het resultaat van de logger is niet verrassend, maar bedenk dat nu een goede, modulaire programmeerwijze wordt gebruikt.

Praktijk: een logging module schrijven

- Bekijk bestand /1007_logger2/logger2.js
 - Een andere schrijfwijze voor de logger

```
// logger2.js - exporteer logging-methoden als functie
module.exports = function () {
    this.log = function (msg) {
        console.log('=> Log: ' + msg);
    };

    this.info = function (msg) {
        console.info('=> Info: ' + msg);
    };

    this.error = function (msg) {
        console.error('=> Error: ' + msg);
    };

    return this;
};
```

Praktijk: een logging module schrijven

- Bekijk bestand /1007_logger2/app2.js
 - De app herschreven

```
// app2.js - logger.js exporteert nu een functie met verschillende methods.  
var logger = require('./logger2');  
var myLog = logger(); // functie invoking  
myLog.info('Dit is informatie');  
myLog.error('Dit is een foutmelding');
```

Modules laden in andere modules

- Bekijk bestanden in de map /1108_logger3/

- getTime.js

```
// getTime.js - module een geformateerde tijd (hh:mm:ss) teruggeeft.  
module.exports = function () {  
    var now = new Date();  
    var tijd = now.getHours() + ':' + now.getMinutes() + ':' + now.getSeconds();  
    return tijd;  
};
```

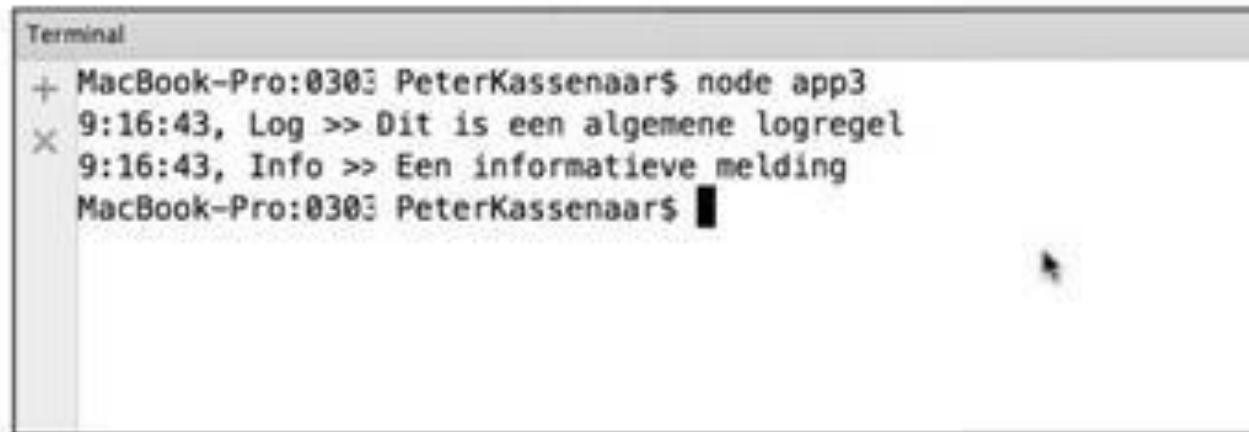
- Logger3.js

```
// logger3.js - require een andere module  
var time = require('./getTime');  
module.exports = function () {  
    this.log = function (msg) {  
        console.log(time() + ', Log >>' + msg);  
    };  
    --  
    return this;  
};
```

Modules laden in andere modules

- Bekijk bestanden in de map /1008_logger3/

Tot slot hoeft in app3.js niks gewijzigd te worden. De publieke interface van logger is immers hetzelfde gebleven. De uitvoer ziet er bijvoorbeeld uit zoals in de afbeelding.



```
Terminal
+ MacBook-Pro:0303 PeterKassenaar$ node app3
× 9:16:43, Log >> Dit is een algemene logregel
× 9:16:43, Info >> Een informatieve melding
MacBook-Pro:0303 PeterKassenaar$ █
```

Afbeelding 3.4 De logger voegt nu ook de tijd toe aan de melding. Deze staat in een afzonderlijke module.

NPM gebruiken

- In de vorige slides werd beschreven hoe je zelf modules kan schrijven
- Via NPM kunnen duizenden modules & packages gedownload & gebruikt worden
- NPM is samen met NodeJS geïnstalleerd

```
npm install <package-naam> // installeer package lokaal, in huidige project.
```

```
npm install <package-naam> -g // installeer package globaal, overal beschikbaar.
```

NPM gebruiken

- Belangrijke NPM-opdrachten om te onthouden zijn:
 - **npm install** installeer een package
 - **npm uninstall** verwijder een package (inclusief dependencies)
 - **Npm cache clean** Maak de cache leeg
 - **npm update** Update een package
 - **npm init** Maak een package.json-bestand met package beschrijving
 - **npm publish** Publiceer een package naar de registry zodat anderen hem ook kunnen gebruiken
- Meer leren over NPM: <http://docs.npmjs.org>

De module ‘moment’ gebruiken

- Bekijk bestanden in de map /1009_moment
- Moment
 - is een bibliotheek voor het werken met datums en tijden
 - Handiger dan default methodes van JavaScript
 - <http://momentjs.com/>

```
npm install moment
```

De module ‘moment’ gebruiken

- Bekijk bestand /1009/logger4.js

```
var moment = require('moment');
module.exports = function () {
    moment.locale('nl'); // Stel Nederlandse notatie in
    this.log = function (msg) {
        console.log(moment().format('lll') + ', Log >> ' + msg);
    };
    ...
}
```

- Het statement `moment.locale('nl')` zorgt er voor dat de Nederlandse datum- en tijdnotatie wordt gebruikt.
- De formatstring `.format('lll')` zorgt voor een standaardweergave in lang datumformaat.
- Zie de documentatie (en de afbeelding) voor meer mogelijkheden voor formatteren van datum en tijd.

De module ‘moment’ gebruiken

- Bekijk bestand /1009_moment/app4.js

In app.js hoeft wederom niets gewijzigd te worden, alle aanpassingen waren op het niveau van de module logger. Draai daarom de applicatie met de volgende opdracht:



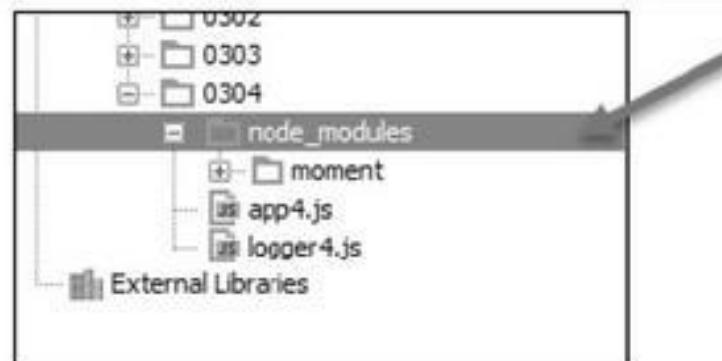
```
Terminal
+ MacBook-Pro:0304 PeterKassenaar$ node app4.js
✖ 29 mei 2015 10:15, Log >> Dit is een algemene logregel
✖ 29 mei 2015 10:15, Info >> Een informatieve melding
MacBook-Pro:0304 PeterKassenaar$
```

Afbeelding 3.7 Moment zorgt voor de weergave in het Nederlandse datum- en tijdformaat.

De map node_modules

- Bekijk de map /1009/node_modules

Packages die via NPM aan het project worden toegevoegd, worden opgeslagen in een map node_modules. NPM maakt deze map als hij nog niet bestaat. Als u nu in het project gaat kijken, zult u deze map dus – na het toevoegen van moment – ongetwijfeld tegenkomen.



Afbeelding 3.8 NPM heeft de map node_modules toegevoegd. Hierin staan lokale modules voor het huidige project.

De map moment is aanwezig als submap van node_modules. Zo is altijd snel te inspecteren welke modules in een project geladen zijn.

Enkele populaire NPM packages

- Underscore & lodash
 - Helper libraries voor functies voor het werken met collecties, arrays, objecten en variabelen
 - <http://underscorejs.org/> en <https://lodash.com/>

```
npm install underscore
```

```
npm install lodash
```

Deze twee bibliotheken zijn overigens direct een uitzondering op de regel dat de variabelenaam meestal gelijk is aan de package-naam. In een applicatie worden ze vaak als volgt gebruikt:

```
var _ = require ('underscore'); // OF  
var _ = require ('lodash');
```

Enkele populaire NPM packages

- Toepassing op Underscore en lodash

```
_first([1,2,3,4,5]);    // 1
_.first([1,2,3,4,5], 3); // 1, 2, 3. Eerste drie elementen uit de array
_.last([1,2,3,4,5]);   // 5
```

Meer informatie over underscore en lodash is te vinden op underscorejs.org en lodash.com. Google op **underscore vs lodash** voor discussies over welke bibliotheek “het beste” is, of de overeenkomsten en verschillen tussen beide.

Enkele populaire NPM packages

- Request

Request is een handige bibliotheek om te werken met http-calls vanuit uw applicatie. Dit is dus iets anders dan de webserver die u in hoofdstuk 2 hebt gemaakt; deze ontvangt http-calls en reageert daarop. De module request wordt bijvoorbeeld als volgt ingezet:

```
npm install request
```

En vervolgens

```
var request = require('request');
request('http://www.webdevelopmentlibrary.nl', function (error, response, body) {
  if (!error && response.statusCode == 200) {
    console.log(body) // Toon de HTML van webdevelopmentlibrary.nl in de console
  }
})
```

Meer informatie over request is te vinden op
www.npmjs.com/package/request

Enkele populaire NPM packages

- Colors (/1010_colors)
 - Geeft meldingen in de console weer in verschillende kleuren

```
npm install colors
```

Daarna kunnen we de logger als volgt aanpassen:

```
var moment = require('moment'),
    colors = require('colors');
module.exports = function () {
    moment.locale('nl'); // Stel Nederlandse notatie in
    this.log = function (msg) {
        var prefixString = moment().format('lll') + ', Log >> ';
        console.log(prefixString.green + msg);
    };
    console.log(colors.green(prefixString + msg));
}
```

Zelf packages maken met npm init

- **Package.json**
 - Handig om in dit bestand te beschrijven van welke andere modules deze package afhankelijk is
 - Behalve de afhankelijkheden staat in dit .json-object nog veel meer informatie; auteur, versie, gebruikerslicentie, testopdrachten beschikbaar, minimale versie van node.js, enz

Zelf packages maken met npm init

- Maak een nieuwe directory en plaats hierin alle bestanden die je zelf geschreven hebt (app.js en logger.js)
- Open de CLI in de nieuwe folder en typ **npm init**
- Beantwoord de vragen die verschijnen (niet alle vragen zijn verplicht)

A screenshot of a terminal window titled "Terminal". The window shows the command "npm init" being typed and the subsequent configuration steps. The configuration fields are highlighted with a large gray rectangle.

```
Terminal
+ See 'npm help json' for definitive documentation on how to
  and exactly what they do.
x
npm init

Use 'npm install <pkg> --save' afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (0306) app
version: (1.0.0)
description: Een eenvoudige logging-app
entry point: (app.js)
test command:
git repository:
keywords: test, logging, colors, moment
author: Peter Kassenaar
```

Zelf packages maken met npm init

- Package.json

```
3  "version": "1.0.0",
4  "description": "Een eenvoudige logging-app",
5  "main": "app.js",
6  "scripts": {
7    "test": "echo \\\"Error: no test specified\\\" && exit 1"
8  },
9  "author": "",
10 "license": "ISC",
11 }
12 }
```

Zelf packages maken met npm init

- Modules toevoegen aan package.json vanuit de project dir:

```
npm install moment
```

```
npm install --save colors
```

- Deze modules komen in de map .\node_modules te staan. De link wordt gelegd in de dependencies binnen package.json

```
    "main": "app5.js",
    "dependencies": {
        "moment": "^2.10.3",
        "colors": "^1.1.0"
    },
```

Zelf packages maken met npm init

- Package (her)installeren
 - Met `npm install` (dus zonder packagenaam) gaat:
 - NPM zoeken naar `package.json` in de huidige map
 - Alle dependencies uit `package.json` laden in de folder `node_modules`
 - Je kan dit testen door de map `node_modules` te verwijderen en `npm install` uit te voeren

Regels voor require()

- Als een padnaam is aangegeven, zoekt Node de module in het aangegeven pad
Voorbeeld: `require('./logger');`
- Als er geen padnaam is aangegeven, zoekt Node in `node_modules`. De module moet met `npm install <package>` geïnstalleerd zijn
Voorbeeld: `require('moment');`
- Als hij niet wordt aangetroffen in `node_modules`, is de volgende stap dat Node zoekt in beschikbare globale modules (bv: `http`, `path`, `file system`, ...)
Voorbeeld: `require('http');`
- Indien hij nog niet gevonden is: `error: cannot find module`

De webserver uitbreiden



Enkele belangrijke variabelen en modules

- Globals `_filename` en `_dirname`

Node.JS kent deze variabelen om de huidige directory en bestandsnaam makkelijk beschikbaar te maken.

```
console.log('De bestandsnaam is: ', _filename);
console.log('De huidige directory is: ', _dirname);
```

Enkele belangrijke variabelen en modules

- De module Path
- Globale module die altijd beschikbaar is (zoals bv ook http)
- Eerst require gebruiken:

```
var path = require('path');
```
- Vervolgens kunnen volgende methodes gebruikt worden:
 - path.normalize(pad) normaliseren van paden
 - path.join([pad1], [pad2]) voegt parameters samen tot 1 pad
 - path.resolve(pad) geeft het absolute pad van de meegegeven dir
 - path.dirname(pad) returned de dir naam uit een pad
 - path.basename(pad) returned laatste deel uit een pad (bestandsnaam)
 - path.extname(pad) returned de extentie van het pad

Enkele belangrijke variabelen en modules

- De module path
 - Bekijk de bestanden in /1012_Path

```
var path = require('path');
var voorbeeldPath = ('Users/Peter Kassenaar/Documents/test.html');
console.log('normalize: ', path.normalize(voorbeeldPath));
console.log('resolve: ', path.resolve(voorbeeldPath));
console.log('dirname: ', path.dirname(voorbeeldPath));
console.log('basename: ', path.basename(voorbeeldPath));
console.log('extname: ', path.extname(voorbeeldPath));
```

Enkele belangrijke variabelen en modules

- De module File System
 - Afgekort fs -> var fs = require('fs');
- Globale module die altijd beschikbaar is (zoals bv ook http)
- Volgende methodes kunnen gebruikt worden:
 - `fs.readFile(filename[, options], callback)`
De asynchrone manier om bestanden in te lezen. Callbackfunctie om aan te geven wat er moet gebeuren met het bestand na het inlezen.
 - `fs.readFileSync(filename [, options])`
De synchrone manier om bestanden in te lezen. Dit wordt afgeraden omdat de thread geblokkeerd is zolang Node.js bezig is met het inlezen van bestanden.
 - Daarnaast ook `fs.writeFile()` en `fs.writeFileSync`

Enkele belangrijke variabelen en modules

- De module fs
 - Bekijk de bestanden in /1013_FileSystem

```
var fs = require('fs');
var msg = 'Hello World';

// #1. Bestand opslaan
fs.writeFile('hello.txt', msg, function () {
    console.log('bestand opgeslagen!')
});
```

```
// #2. Bestand inlezen en in de console tonen
fs.readFile('hello.txt', 'utf8', function (err, data) {
    if (err) {
        console.log('Error: ', err);
    } else {
        console.log('bestand ingelezen: ', data);
    }
});
```

Bij het inlezen moet u het encoding-type ogeven. Anders retourneert Node een <Buffer>-object.

Enkele belangrijke variabelen en modules

- De module fs
 - Bekijk de bestanden in /1012_Path

```
// #4. Eerst testen of bestand bestaat, voordat unlink wordt aangeroepen:  
fs.exists('test.txt', function (exists) {  
  if (exists) {  
    deleteFile('test.txt');  
  } else {  
    console.log('text.txt niet gevonden! Kan niet verwijderen.')  
  }  
});|
```

```
// #3. Bestand verwijderen via unlink  
fs.unlink('text.txt', function (err) {  
  if (err) {  
    console.log('Error: ', err);  
  } else {  
    console.log('Text.txt is verwijderd');  
  }  
});|
```

De webserver uitbreiden

- Bekijk het bestand `/1014_server/server_01.js`

```
// Een eenvoudige webserver.  
// 0. initialisatie en variabelen  
var http = require('http'),  
    fs = require('fs'),  
    path = require('path'),  
    root = __dirname + '/public/'; // magic variable  
  
// 1. Maak de webserver  
var server = http.createServer(function (req, res) {  
  
});  
  
// 2. Start de server  
server.listen(3000);  
console.log('Server gestart op http://localhost:3000');
```

De webserver uitbreiden

- De homepage serveren (en andere pages)
 - Dit is het bestand \public\index.html

```
// 1. Maak de webserver
var server = http.createServer(function (req, res) {
    // 1a. Check of de root wordt opgevraagd.
    var fileName = '';
    var url = req.url;
    if (url === '/') {
        url = 'index.html'; // redirect als geen bestandsnaam is opgegeven
    }
    fileName = root + url; // root = '/public/'
    console.log('Gevraagd bestand: ', path.basename(fileName));
```

De webserver uitbreiden

- Checken of het gevraagde bestand bestaat

```
// 1b. Check of bestand bestaat.  
fs.exists(fileName, function (exists) {  
    if (exists) {  
        serveFile(fileName); // ja.  
    } else {  
        fileName = root + '404.html'; // nee  
        serveFile(fileName);  
    }  
});
```

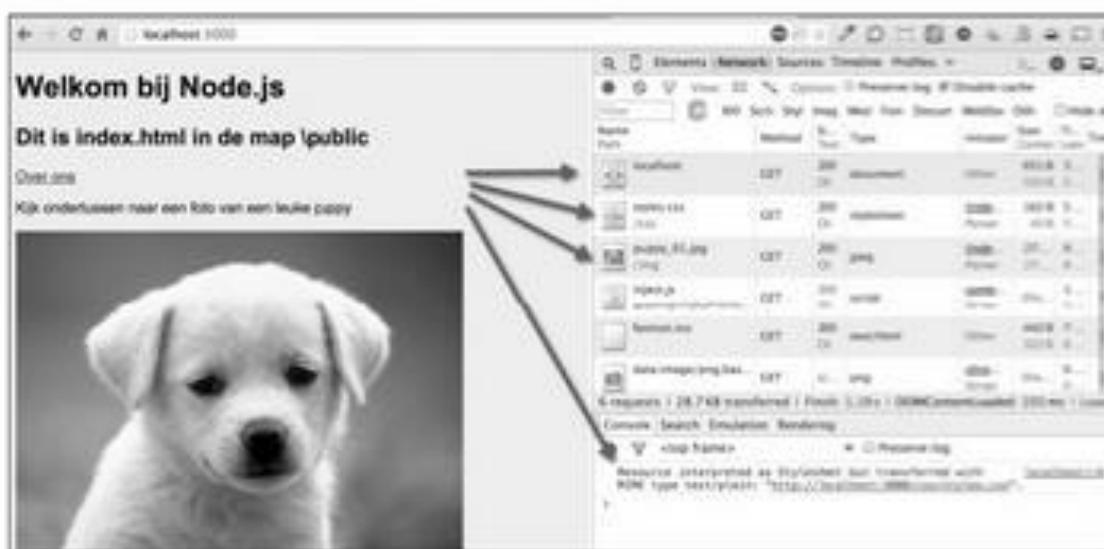
De webserver uitbreiden

- Streams – het bestand serveren via een helperfunctie en events
 - Events ‘on’ met 2 parameters: wat er binnenkomt en een callback function

```
// 1c. Serveer gevraagde bestand.  
function serveFile(requestFile) {  
    // 2. Maak een stream en server op basis van Events  
    var stream = fs.createReadStream(requestFile);  
    stream.on('data', function (chunk) {  
        res.write(chunk);  
    });  
    stream.on('end', function () {  
        res.end();  
    });  
    stream.on('error', function (err) {  
        console.log('error: ' + err);  
    });  
}
```

De webserver uitbreiden

- MIME-types
 - Gaat goed omdat de browser een gok doet en minimale headers voorziet
 - Afhandeling zou serversided moeten gebeuren



Afbeelding 4.6 De pagina bevat nu HTML, stylesheets en een afbeelding.

Name Path	Method	Status Text	Type	Initiator	Size Content	Ti... Latency	Time
localhost	GET	200 OK	document	Other	651B 531B	3ms 2ms	

6 requests | 28.7 KB transferred | Finish: 1.19s | DOMContentLoaded: 155 ms | Load: ...

X Headers Preview Response Timing

Request Method: GET

Request URL: localhost:1000/

Response Headers view source

Connection: keep-alive
Date: Mon, 01 Jun 2015 11:29:35 GMT
Transfer-Encoding: chunked
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch

Afbeelding 4.7 In de netwerktab van de developer tools is te zien dat elk bestand nu nog met een minimale set http-headers wordt gereturneerd.

De webserver uitbreiden

- MIME-types
 - Eenvoudigste oplossing:
 - Afbeeldingen / css zal ook als “tekst/html” verzonden worden.

```
function serveFile(requestFile) {  
    // 2. Maak een stream en serveer op basis van events  
    var stream = fs.createReadStream(requestFile);  
    res.writeHead(200, {'Content-Type' : 'text/html'});  
    stream.on('data', function (chunk) {  
        res.write(chunk);  
    });  
    ...  
}
```

De webserver uitbreiden

- MIME-types
 - Dynamische bepaling van MIME-types kan met de module mime
`npm install --save mime`
 - Vervolgens kan na de require een lookup gedaan worden van de MIME types
`var mime = require('mime');`

`mime.getType('index.html');`
`mime.getType('styles.css');`
`mime.getType('foto.png');`

De webserver uitbreiden

- Bekijk het bestand /1014_server/server_02.js

```
var http = require('http'),
  fs = require('fs'),
  path = require('path'),
  mime = require('mime'),
  root = __dirname + '/public/' // magic variable

function serveFile(requestFile) {
  // 2. Maak een stream en server op basis van Events
  res.writeHead(200, {'Content-Type': mime.getType(requestFile)});
  var stream = fs.createReadStream(requestFile);
  stream.on('data', function (chunk) {
    res.write(chunk);
  });
  stream.on('end', function () {
    res.end();
  });
  stream.on('error', function (err) {
    console.log('error: ' + err);
  });
}
```

De webserver uitbreiden

- Betere methode voor 404
 - Momenteel wordt de 404 error ook met een http-header 200 OK
 - Aparte methode voorzien voor het renderen van de 404 pagina:
 - Hier met fs.readFile() in plaats van streams

```
// 1d. Serveer 404, inclusief juiste http-header
function serve404(requestFile) {
    res.writeHead(404, {'Content-Type': mime.lookup(requestFile)});
    fs.readFile(requestFile, 'utf8', function (err, data) {
        if (err) {
            console.log('Error: ', err);
        } else {
            res.end(data);
        }
    })
}
```

De webserver uitbreiden

- Betere methode voor 404
 - Aanpassen fs.exists() methode:

```
// 1b. Check of bestand bestaat.  
fs.exists(fileName, function (exists) {  
    if (exists) {  
        serveFile(fileName); // ja.  
    } else {  
        fileName = root + '404.html'; // nee  
        serve404(fileName);  
    }  
});
```

NodeJS – Webapps met express



Wat is Express?

- Express is ook bekend als een ‘middleware-framework’
 - Onderliggend worden er een groot aantal zaken gereld en aangeboden als één functie
 - Bijvoorbeeld: in kale node.js moet een `http.createServer()` en `.listen()` opgeroepen worden. Binnen express gebruiken we enkel `.listen()`
- De globale opbouw van Express:
 - Express ontvangt een http-request
 - De request wordt langs middleware-functies gestuurd waarmee data bekeken en verwerkt kan worden voordat het de app binnentreedt of verlaat

Wat is Express

- Meest gebruikte library voor web projecten (standaard)
- Standaard uitwisselingsformaat in Express is JSON
- Beschikt over ingebouwde functie om JSON in de body van een request te lezen en verwerken
 - Geen handmatige parsing meer nodig

Een express-app maken

- Express installeren
- Routes definiëren
- JSON retour zenden

Download de voorbeelden uit Resources/CH11_Voorbeelden

Een express-app maken

- Maak een nieuwe directory en installeer Express via NPM
 npm install express
- Maak in de directory een bestand server.js met volgende inhoud:

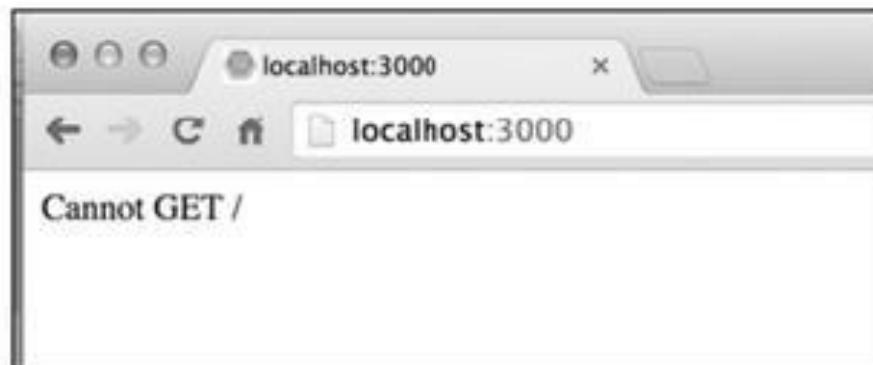


```
server.js
1 var express = require('express');
2 var app = express();
3
4 app.listen(3000);
5 console.log('Express server gestart op localhost:3000...');
```

Een express-app maken

- Start de applicatie
`node server.js`
- Probeer de homepage op te vragen

Omdat nog geen routes of inhoud is gedefinieerd, wordt een eenvoudige foutmelding getoond (zie afbeelding).



Afbeelding 5.3 Met enkele regels code is een Express-webserver gemaakt. Er is echter nog geen inhoud gedefinieerd.

Een express-app maken

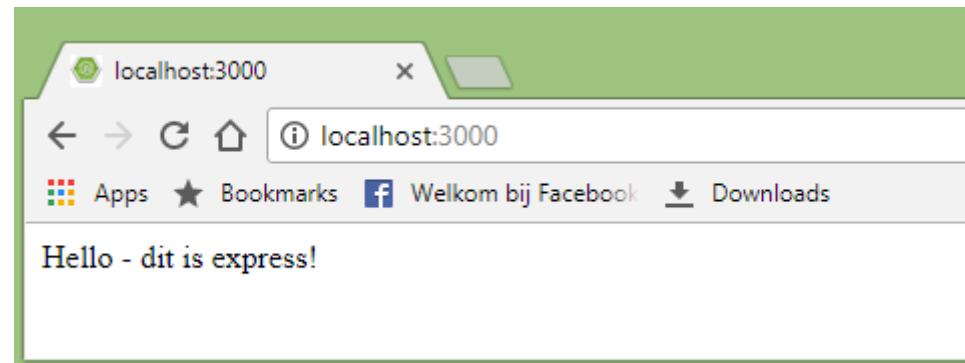
- Bekijk het bestand /1101/server.js
- Routes definiëren
 - Routes moeten gedefinieerd worden om te bepalen hoe een inkomend verzoek moet worden afgehandeld.
- Breid de server uit als volgt:



```
server.js
1 var express = require('express');
2 var app = express();
3
4 app.get('/', function(req, res){
5   res.send('Hello - dit is express!');
6 });
7
8 app.listen(3000);
9 console.log('Express server gestart op localhost:3000...');
```

Een express-app maken

- Vraag de homepage opnieuw op
 - Let ook op de Response headers in de developer tools. Express heeft automatisch de headers opgesteld



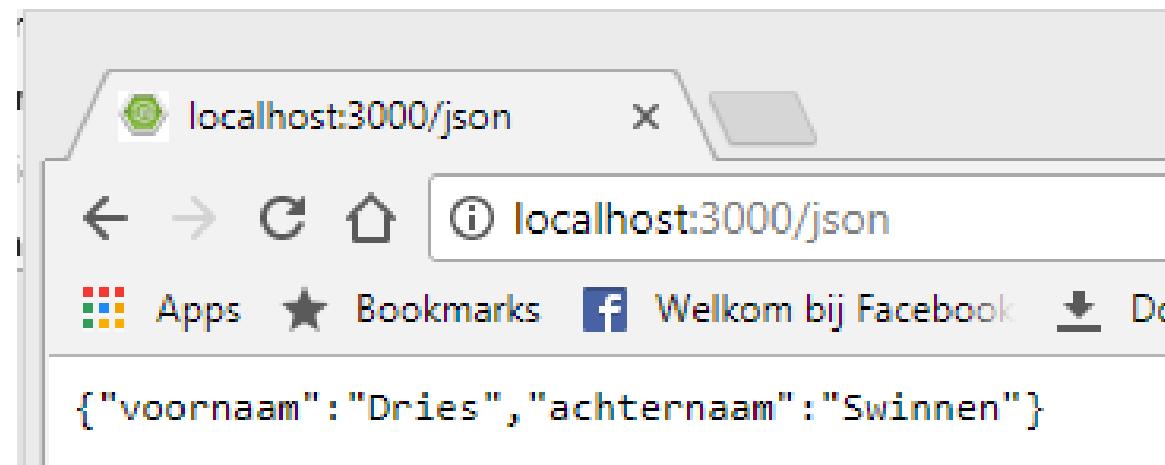
Een express-app maken

- JSON retour zenden
 - Het response object uit de callback methode heeft meer mogelijkheden als enkel .send
 - Methodes zoals .json zijn mogelijk
- Voorzie volgende route:

```
var persoon = {  
    voornaam: 'Dries',  
    achternaam: 'Swinnen'  
};  
app.get('/json', function(req, res){  
    res.json(persoon);  
});
```

Een express-app maken

- Start de applicatie en test de nieuwe route
 - Je merkt dat het opnieuw uitvoeren van node server.js nodig is om wijzigingen te verwerken. Dit kan automatisch met de module [nodemon](#)
- De conversie naar JSON is automatisch toegepast door Express (dubbele aanhalingstekens)

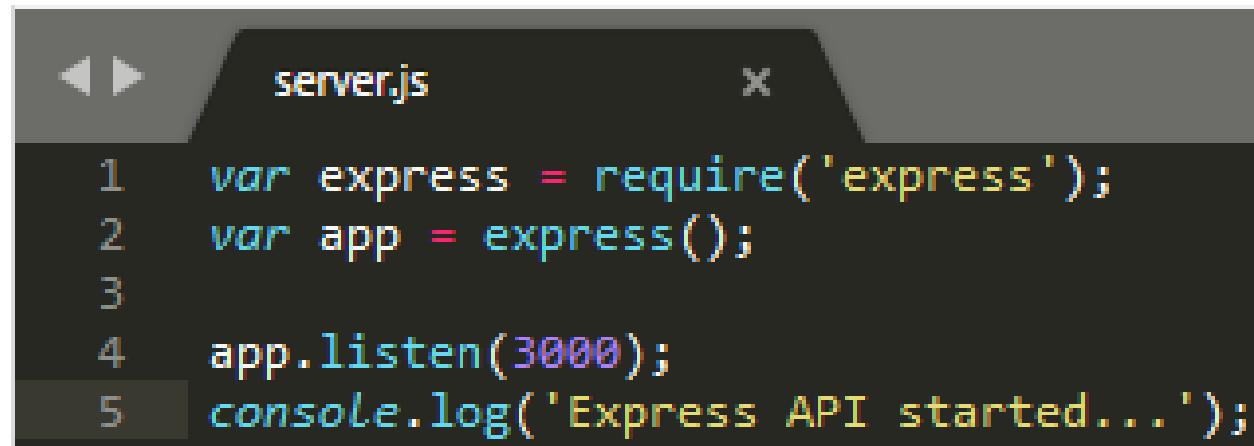


Een express-API maken

- Bekijk de bestanden in de map `/1102_API`
- Aan de hand van routes en het teruggeven van JSON kunnen we een heel eenvoudige API opzetten.
- Volgende routes worden voorzien in onze app:
 - De route `/api` moet korte instructies voor het gebruik van de API teruggeven
 - De route `/api/autoren` moet een JSON-tabel met auteurs retourneren
 - De route `/api/boeken` moet een JSON-tabel met boeken retourneren

Een express-API maken

- Maak een minimale express app in een nieuwe folder en installeer express:



```
server.js

1 var express = require('express');
2 var app = express();
3
4 app.listen(3000);
5 console.log('Express API started...');
```

Een express-API maken

- Voorzie een route voor de instructie van de API:

```
app.get('/', function(req, res){  
  var msg = '<h1>Express API</h1>;  
  msg += '<p>Gebruik \\\\api\\\\auteurs voor een lijst met auteurs.</p>;  
  msg += '<p>Gebruik \\\\api\\\\boeken voor een lijst met boeken.</p>;  
  res.send(msg);  
});
```

- De uitleg wordt nu getoond bij het laden van de app op
<http://localhost:3000>

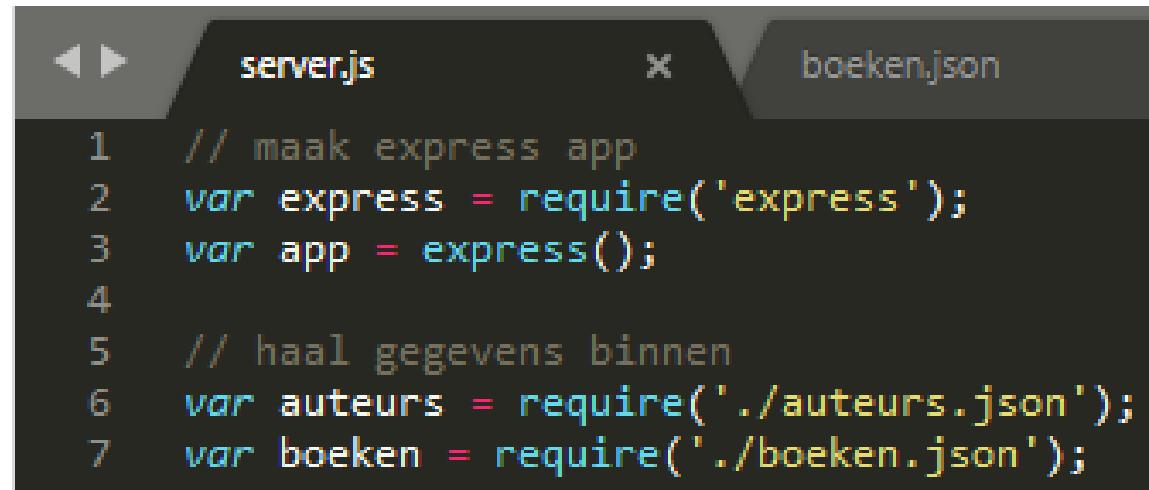
Een express-API maken

- Data invoegen met require()
 - In onze site zijn 2 JSON bestanden voorzien (zie H12/1202_API/)
 - Auteurs.json
 - Boeken.json

```
[  
  {  
    "id": 1,  
    "titel": "Web Development Library - AngularJS",  
    "auteur": "Peter Kassenaar",  
    "ISBN": "9789059407879"  
  },  
  {  
    "id": 2,  
    "titel": "Focus op Fotografie: Portretfotografie",  
    "auteur": "Pieter Dhaeze",  
    "ISBN": "9789059408128"  
  },  
  {  
    "id": 3,  
    "titel": "App Development Library - Apps maken voor Android",  
    "auteur": "Michiel de Rond",  
    "ISBN": "9789059408395"  
  },  
  {  
    "id": 4,  
    "titel": "Ontdek de Windows-tablet",  
    "auteur": "Bob van Duuren",  
    "ISBN": "9789059406186"  
  }  
]
```

Een express-API maken

- Data invoegen met require()
 - Voorzie volgende regels boven de declaratie van de route:



The image shows a screenshot of a code editor. On the left, there is a navigation bar with arrows and file names: 'server.js' (which is the active tab), 'x', and 'boeken.json'. The main area displays the contents of the 'server.js' file:

```
1 // maak express app
2 var express = require('express');
3 var app = express();
4
5 // haal gegevens binnen
6 var auteurs = require('../auteurs.json');
7 var boeken = require('../boeken.json');
```

Een express-API maken

- Vervolgens voegen we de routes toe aan onze applicatie

```
app.get('/api/auteurs', function(req, res){  
    res.json(auteurs);  
});  
app.get('/api/boeken', function(req, res){  
    res.json(boeken);  
});
```

- Start de applicatie opnieuw (node server.js of via nodemon)

Een express-API maken

- Routeparameters gebruiken kan via de routes
 - Opbouw lijkt op die van angular:

```
app.get('/api/boeken/:idx', function(req, res){  
});
```

Een express-API maken

- De parameter opvragen kan met `req.params.<paramnaam>`
- Gebruiken de javascript functie `forEach` om door alle boeken te lopen

```
app.get('/api/boeken/:idx', function(req, res){  
    var id = req.params.idx;  
    var gezochtBoek;  
    boeken.forEach(function(boek){  
        if(boek.id === parseInt(id)){  
            gezochtBoek = boek;  
        }  
    });  
    res.json(gezochtBoek);  
});
```

Een express-API maken

- Routeparameters gebruiken

Dit werkt prima, zoals de afbeelding laat zien. Hier is het boek met id 4 opgevraagd.



A screenshot of a web browser window. The address bar shows the URL `localhost:3000/api/boeken/4`. The main content area of the browser displays a JSON object:

```
{  
  id: 4,  
  titel: "Ontdek de Windows-tablet",  
  auteur: "Bob van Duuren",  
  ISBN: "9789059406186"  
}
```

Afbeelding 5.8 Met routeparameters zijn dynamische URL's te creëren.

Statische bestanden serveren

- Bekijk de bestanden in de map `/1103_static`
- De opdracht `app.use()` en `express.static()`
 - Statische bestanden worden in de webserver in een public-map gezet en worden aangegeven door één regel code
 - `app.use` betekent dat Express een middleware functie moet uitvoeren op binnenkomende request(s)

```
var auteurs = require('./auteurs.json');
var boeken = require('./boeken.json');

// 3. Stel middleware in voor serveren van statische bestanden (HTML, CSS, images)
app.use(express.static(__dirname + '/public'));

// 4. Start de server.
app.listen(3000);
console.log('Express app gestart op localhost:3000');
```

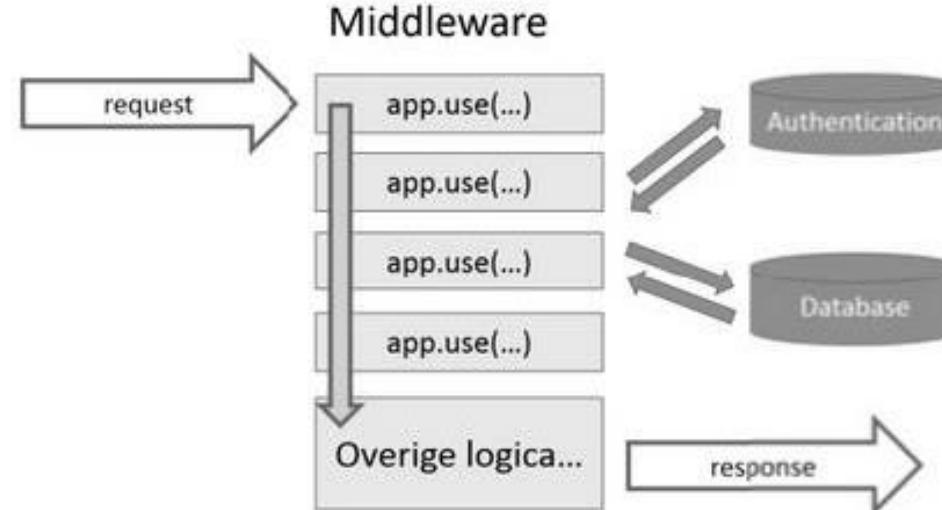
Statische bestanden serveren

- De HTML site kan nu geladen worden via onze server



Werken met middleware

- Bekijk de bestanden in de map /1103_static
- Middleware is een functie die wordt uitgevoerd op het binnenkomende request
 - Resultaat wordt doorgegeven aan de volgende functie in de stack tot er een functies meer z
 - Daarna wordt d



Werken met middleware

- app.use() wordt gebruikt om middleware functies te definiëren
- Parameters voor app.use()
 - Req: het request object
 - Res: het result object
 - Next: de volgende functie in de middleware-keten

```
app.use(function(req, res, next){  
    console.log('Requested file: ', req.url);  
    next();  
});
```

- De methode next() moet opgeroepen worden, anders blijft de request hangen

Werken met middleware

- Verschillende soorten middleware beschikbaar:
 - Application-level, router-level, error-handling, builtin, third-party
- De volgorde van de functies is belangrijk
- Meer informatie over middleware
 - <http://expressjs.com/guide/using-middleware.html>
- In het voorbeeld wordt de middleware gebruikt als logging:

```
$ node server.js
Express app gestart op localhost:3000
Requested file: /
Requested file: /js/lib/bootstrap/dist/css/bootstrap.min.css
Requested file: /boeken.html
Requested file: /js/lib/bootstrap/dist/css/bootstrap.min.css
Requested file: /index.html
Homepage opgevraagd: 2018-01-13T14:21:10.691Z
Requested file: /js/lib/bootstrap/dist/css/bootstrap.min.css
```

POST-requests

- Bekijk de bestanden in /1105_POST
- Voor GET-requests maakte we gebruik van app.get()
- Voor POST-requests bestaat de functie app.post() met dezelfde structuur als app.get()
 - Methodes bestaan voor elk type request in Express
- Extra middleware / [body-parser](#) is vereist voor een vlotte werking
 - Parsed de formulier velden uit de body van een request

```
npm install body-parser
```

POST-requests

- Body-parser toevoegen en configureren
 - Eerst require
 - Daarna via middleware (app.use()) configureren voor gebruik
 - bodyParser.urlencoded of bodyParser.json() afhankelijk van de soort request

```
// Middleware laden voor het parsen van JSON in het request
var bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({
  extended: true
}));
```

POST-requests

- Vervolgens kan een post request opgevangen worden via app.post() zoals hieronder:

```
// Een POST-request verwerken
var user = {};
app.post('/', function (req, res) {
    // verwerk binnenkomende request. We gaan er van uit
    // dat de parameter 'username' en 'email' aanwezig zijn.
    // TODO: error checking!
    console.log(req.body);
    user.username = req.body.username;
    user.email = req.body.email;

    // Echo het user-object naar de client
    res.json(user);
});
```

POST-requests

- Bij het opstarten van server2.js uit het voorbeeld kan de post ook uitgevoerd worden via <http://localhost:3000/login.html>
- Simuleren van http requests kan via:
 - Via browser plugins/add-ons zoals postman / ARC
 - Via tools zoals Fiddler
 - Via curl

De functie Router()

- Bekijk de bestanden server.js en routes.js in de map /1106/Router
- Als de applicatie wordt gestart, is de uitvoer naar verwachting
- De hoofdmodule server.js is nu een stuk compacter



De functie Router()

```
server.js      x
// Express en externe routes laden
var express = require('express');
var routes = require('./routes');
var app = express();

app.use('/', routes);
app.listen(3000, function () {
  console.log('Express-server gestart op http://localhost:3000');
});
```

```
routes.js      x
// routes.js - middleware via express.Router()
var express = require('express');
var router = express.Router();

// Dit mag ook in één regel:
// var router = require('express').Router();

// middleware specifiek voor deze router - merk op dat
router.use(function(req, res, next){
  console.log('Router aangeroepen: ', new Date());
  next();
});

// homepage voor deze router.
router.get('/', function(req, res){
  res.send('Homepage van de router');
});

router.get('/about', function(req, res){
  res.send('Over ons - vanuit de router');
});

router.get('*', function(req, res){
  res.send('onbekende route...');
});
module.exports = router;
```

De functie Router()

- Abstracte routes
 - In voorgaand voorbeeld gebruiken we app.use('/', routes) voor de koppeling van de routes aan onze root directory
 - Kunnen dit specifiek maken voor verschillende entry-points:
 - Bv aparte route /team waarop de routes van toepassing zijn
 - Enkel app.use() aanpassen (zie 1106_router/server2.js)

```
// De router is nu abstract gemaakt en werkt alleen voor de route \team in de
// request. Elke andere route is nu ongeldig (of kan via een andere router worden afgehandeld).
app.use('/team', routes);
```

De functie Router()

- Abstracte routes

Nu is de ‘gewone’ homepage niet meer bereikbaar, maar worden de routes alleen toegepast op de route /team in de URL. Dit alles zonder dat u routes.js hoeven aanpassen.



De functie Router()

- Router gebruiken in de applicatie
 - Bekijk het voorbeeld 1106_router/server4.js
 - De routes staan in een eigen bestand (./router/index.js)
 - De locatie van de data is aangepast (boeken.json en auteurs.json)

```
// server4.js
var express = require('express');
var bodyParser = require('body-parser');
var routes = require('./router');

var app = express();
app.use(bodyParser.urlencoded({
    extended: true
)));
app.use(express.static(__dirname + '/public'));

app.use('/', routes);
```

De functie Router()

- Router gebruiken in de applicatie
 - Routes zijn als volgt gedefinieerd

```
// .\router\index.js - bestand met routes voor boeken, auteurs en login.  
var router = require('express').Router();  
var auteurs = require('../data/auteurs.json');  
var boeken = require('../data/boeken.json');  
  
router.get('/api', function (req, res) {  
    //... stuur data.  
});  
  
//... overige routes.  
  
module.exports = router;
```

NodeJS – MongoDB - Mongoose

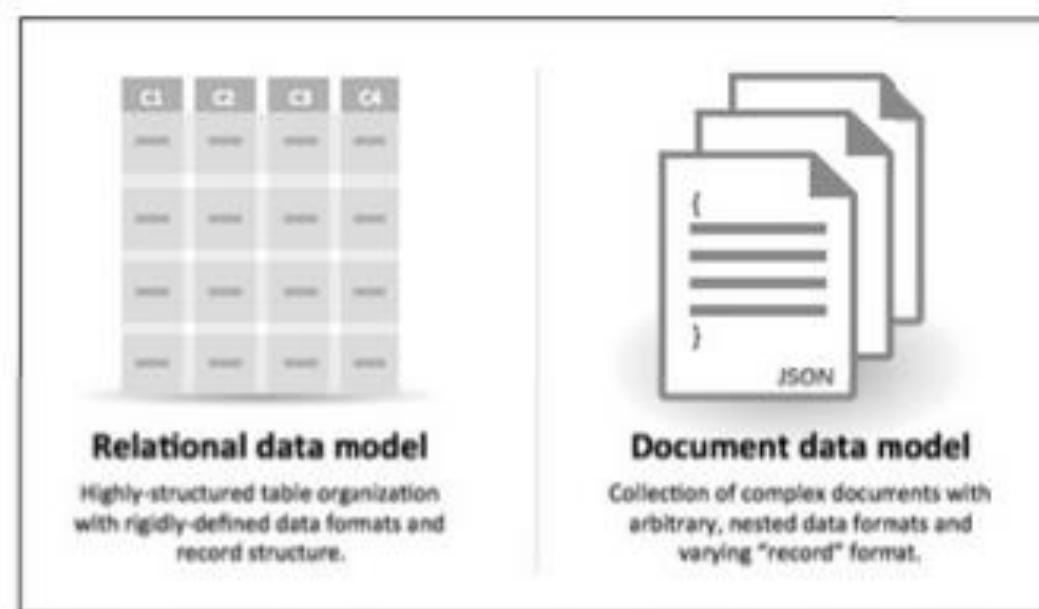


Inleiding Databases en Node.js

- Er bestaan verschillende drivers voor het samenwerken met verschillende databases
 - MySQL, SQL server, MongoDB, ...
- Wij leggen de focus op MongoDB in combinatie met de npm-module mongoose
- Download de voorbeelden uit Resources/CH12_Voorbeelden

Inleiding Databases en Node.js

- Document databases (MongoDB)
 - NoSQL database
 - Type database dat JSON-objecten rechtstreeks kan opslaan in de vorm van documenten



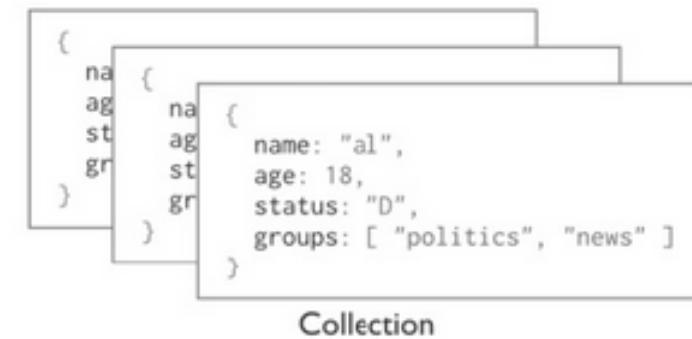
Inleiding Databases en Node.js

- Document databases (MongoDB)
 - Qua structuur zijn er geen voorgedefinieerde tabellen, records, joins en andere typische database-onderdelen, zoals in de RDBMS
 - De enige overeenkomsten tussen verschillende documenten in de database is dat ze een uniek (id)-veld hebben. In MongoDB heet dit letterlijk `_id` en is van het type `ObjectId()`

Inleiding Databases en Node.js

- Document databases (MongoDB)
 - Het is een JSON-database die met JavaScript wordt bediend
 - Een document is een JSON-object
 - Een collection, een verzameling van documenten

```
{  
  name: "sue",           ← field: value  
  age: 26,              ← field: value  
  status: "A",           ← field: value  
  groups: [ "news", "sports" ] ← field: value  
}
```



Inleiding Databases en Node.js

- Document databases (MongoDB)
 - Het veld `_id`
 - Wordt automatisch gemaakt bij het invoegen van nieuwe documenten
 - Het is een unieke sleutel met een random waarde
 - Vergelijkbaar met een primary key in RDBMS
 - Er is geen schema voorzien in MongoDB

```
{  
    "_id" : ObjectId("55ac8c69e16ba6e51644ce05"),  
    "titel" : "Web Development Library - AngularJS",  
    "auteur" : "Peter Kassenaar",  
    "ISBN" : "978059407879"  
}
```

MongoDB installeren en in gebruik nemen

- MongoDB bestaat uit twee delen
 - Mongo Deamon: deamon die op de achtergrond draait, gegevens ophaalt en opslaat in databases. Deze moet als eerste gestart worden (**mongod.exe**)
 - Mongo shell: Dit is de CLI omgeving waarmee we opdrachten geven aan de deamon. Gebruik van databases, documenten toevoegen, documenten opvragen, ... (**mongo.exe**)

MongoDB installeren en in gebruik nemen

- Installatiedetails voor de database zijn te vinden op [docs.mongodb.org](https://docs.mongodb.org/manual/installation/) onder de rubriek “installation”



Afbeelding 6.5 De installatie van MongoDB wordt online uitgebreid toegelicht.

MongoDB installeren en in gebruik nemen

- Ga naar <http://mongodb.org> en klik rechts boven op “get mongodb”
- Kies voor “server” en selecteer de community server en kies het juiste besturingssysteem
- Installeer de complete version (standaard in c:\Program files\mongodb)
- MongoDB Compass moet niet geïnstalleerd worden
- Maak een data directory. Hierin worden de databases opgeslagen
 - **De standaardlocatie is c:\data\db. Deze locatie nemen wij over. Maak zelf deze map in de verkenner.**

MongoDB installeren en in gebruik nemen



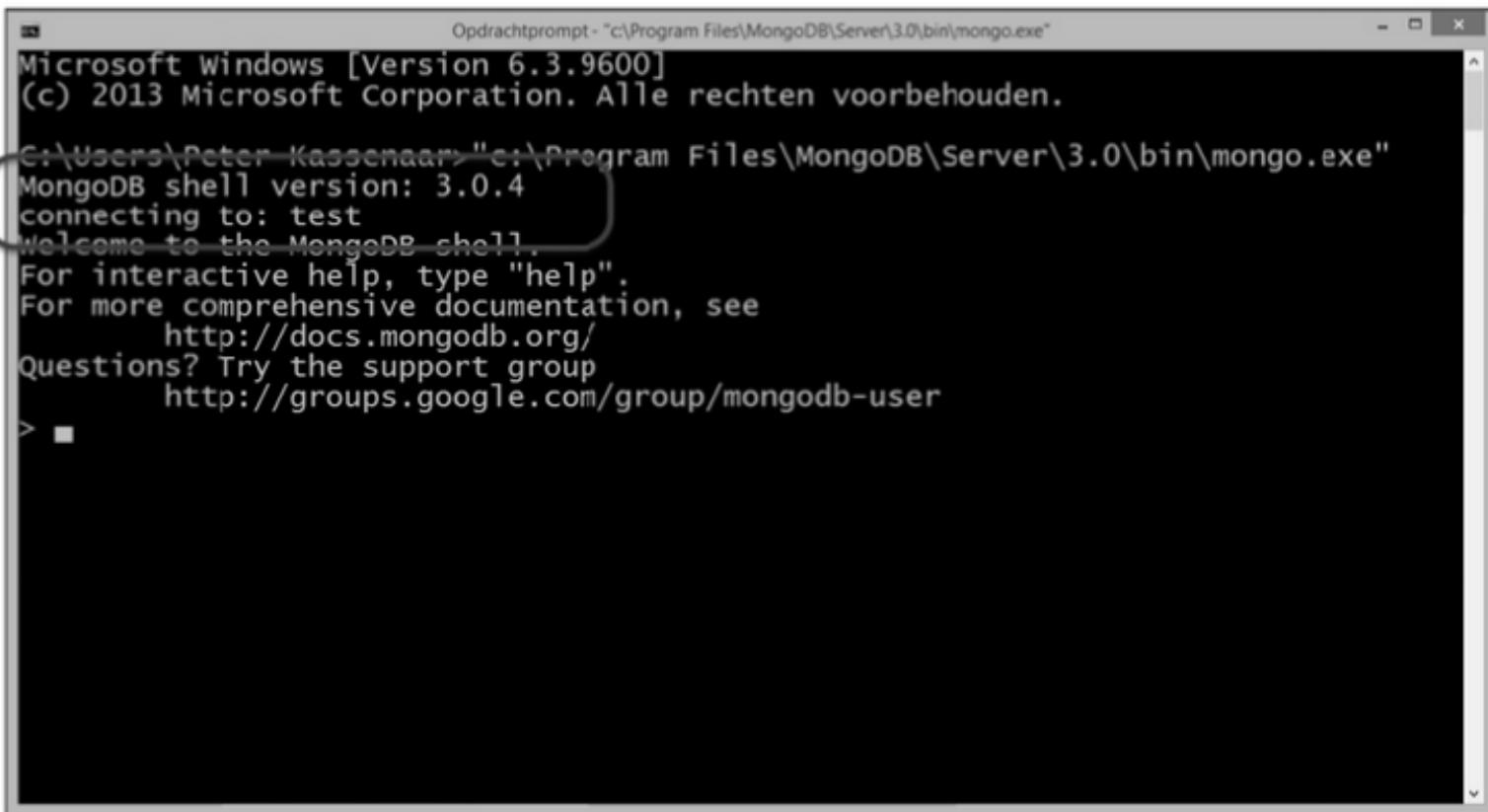
MongoDB starten als een service

Let er op dat u telkens de deamon start als u aan het werk wilt met MongoDB. Het opdrachtvenster van de deamon moet geopend blijven (u mag het wel minimaliseren). Als u elke keer als de computer wordt gestart automatisch MongoDB wilt starten, kunt u Mongo installeren als een *service* in Windows. In de installatiedocumenten leest u hoe dat gaat.

MongoDB installeren en in gebruik nemen

- MongoDB starten
 - Open een nieuw terminal venster. Start de MongoDB-deamon
 - Bij windows: c:\program files\mongoDb\server\4.0\bin\mongod.exe
 - Bij mac: /Applications/mongodb/bin/mongod
 - Gelukt indien de laatste regels “waiting for connections” bevat.
 - Dit venster moet geopend blijven voor het runnen van de server!
 - Open een nieuw terminal venster en start de shell. Dit is mongo.exe in dezelfde folder

MongoDB installeren en in gebruik nemen



A screenshot of a Windows command prompt window titled "Opdrachtprompt - c:\Program Files\MongoDB\Server\3.0\bin\mongo.exe". The window displays the following text:

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\Peter.Kessenaar>"c:\Program Files\MongoDB\Server\3.0\bin\mongo.exe"
MongoDB shell version: 3.0.4
connecting to: test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
    http://docs.mongodb.org/
Questions? Try the support group
    http://groups.google.com/group/mongodb-user
> ■
```

Afbeelding 6.6 De mongo-shell nadat hij voor de eerste keer gestart is.

MongoDB installeren en in gebruik nemen

- U bent nu verbonden met de databaseserver en kunt databases en documenten gaan maken. De indeling is als volgt:
 - **Database** Er is meestal één database per app. Maar er kunnen meerdere databases bij een server worden opgeslagen.
 - **Collection** Een database bestaat uit verzamelingen (collections). In een collection worden de documenten opgeslagen. Je dat dit vergelijken met een tabel
 - **Document** Een document is één enkel item in de database. Het is best te vergelijken met een record uit een sql-database.

Documenten opslaan/bewerken in dedatabase

- Database maken en document toevoegen
 - Maak een nieuwe database

```
> use driesTest
switched to db driesTest
>
```

- Op de achtergrond heeft MongoDB in de data directory een aantal bestanden gemaakt.

Documenten opslaan/bewerken in dedatabase

- Database maken en documenten toevoegen
 - Volgende opdracht voegt een document in :

```
> db.users.insert({name: "Dries Swinnen"})
WriteResult({ "nInserted" : 1 })
>
```

- Een query uitvoeren: Opvragen van documenten

```
> db.users.find()
{ "_id" : ObjectId("5a5a6b67d7c6508a5f517806"), "name" : "Dries Swinnen" }
>
```

Documenten opslaan/bewerken in dedatabase

- Een query uitvoeren: andere zoekmethodes:

```
> db.users.find(ObjectId("5a5a6b67d7c6508a5f517806"))
{ "_id" : ObjectId("5a5a6b67d7c6508a5f517806"), "name" : "Dries Swinnen" }
>
> db.users.find({name: 'Dries Swinnen'})
{ "_id" : ObjectId("5a5a6b67d7c6508a5f517806"), "name" : "Dries Swinnen" }
```

Documenten opslaan/bewerken in dedatabase

- Gegevens in documenten wijzigen
 - Hiervoor wordt de methode .update() gebruikt

```
> db.users.update({name: "Dries Swinnen"}, {$set:{name: "John Doe"})>
> writeResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.users.find()
{ "_id" : ObjectId('5a5a6b67d7c6508a5f517806'), "name" : "John Doe" }
```

- \$set zorgt ervoor dat andere properties van dat document niet undefined worden

Documenten opslaan/bewerken in dedatabase

- Documenten verwijderen
 - Hiervoor wordt de methode .remove() gebruikt

```
> db.users.remove({})
WriteResult({ "nRemoved" : 1 })
>
```

Documenten opslaan/bewerken in dedatabase

- Meer CRUD-operaties
 - Meer info is terug te vinden op
<https://docs.mongodb.com/manual/crud/>

```
db.<collectie>.find()  
db.<collectie>.insert()  
db.<collectie>.update()  
db.<collectie>.remove()
```

Documenten opslaan/bewerken in dedatabase

- Databases opvragen

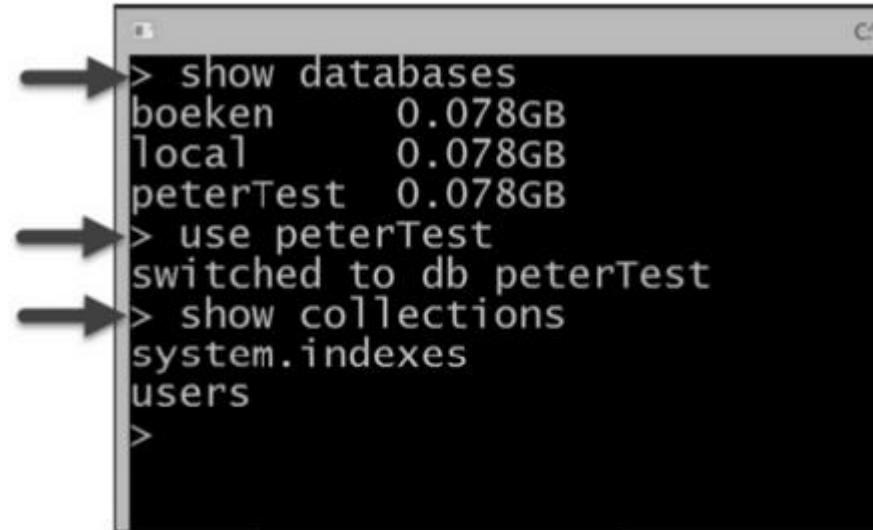
```
show databases
```

```
show collections
```

- Drop databases

```
use <databasenaam>
```

```
db.dropDatabase()
```



A screenshot of a terminal window showing the MongoDB shell. The shell has a dark background with white text. Three arrows point from the left towards the terminal window, indicating the flow of commands from the user's input to the database output.

```
> show databases
boeken      0.078GB
local        0.078GB
peterTest    0.078GB
> use peterTest
switched to db peterTest
> show collections
system.indexes
users
>
```

Documenten opslaan/bewerken in dedatabase



JavaScript-bestand uitvoeren in MongoDB

In plaats van de opdrachten in de mongo-shell te typen, kunt u ze ook opslaan in een JavaScript-bestand en laden in de shell. De opdracht hiervoor is `load(mijnOpdrachten.js)`. In de paragraaf **Getting Started with the Mongo shell** in de online documentatie leest u meer van dit soort handige Mongo-tips.

De rol van Mongoose

- Mongoose is een laag tussen Node.js en MongoDB en acteert als een Object Document Mapper of ODM
- Het vertaalt objecten (vanuit uw applicatie) naar documenten (voor gebruik in MongoDB) en omgekeerd
- Mongoose is een npm-module. Homepage en documentatie op mongoosejs.com

De rol van Mongoose

- Mongoose wordt aan de applicatie toegevoegd:

```
npm install mongoose --save
```

- Modellen en schema's: Basiscode voor een model 'user' ziet eruit als volgt:

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/driesTest', function(){
    console.log('MongoDb connected');
});

var User = mongoose.model('User', {name: String}); // model
var user = new User({name: 'Bob bobbie'}) // instance
user.save(function(err){
    console.log('user gemaakt');
});
```

Een Mongoose CRUD app maken

- Voorbeeldcode: /1201
-



CRUD?

CRUD is een bekende databaseterm en is een afkorting voor *Create, Retrieve, Update* en *Delete*, de meest voorkomende handelingen in databaseapplicaties.

Een Mongoose CRUD app maken

- Centrale connectie met database
 - Maak een aparte file db.js voor de logica voor het verbinden met MongoDB
 - Zorg ervoor dat de mongoDB-deamon draait op de achtergrond

```
//db.js - logica voor verbinden MongoDB
var mongoose = require('mongoose');
var db = mongoose.connect('mongodb://localhost/boeken', function(){
    console.log('mongoose connected');
});
module.exports = db;
```

Een Mongoose CRUD app maken

- Model voor boeken
 - Aparte file /models/boeken.js
 - Voorzien van een schema dat mapt naar de structuur van een collection
 - Koppeling aan een model
 - Uitbreiding van mongoose -- mongoose.model()

```
// boeken.js : Model voor boeken in MongoDB-database
var mongoose = require('mongoose');

const boekSchema = mongoose.Schema({
    titel : {type: String, required: true},
    auteur: {type: String, required: true},
    isbn  : {type: String, required: true},
    date  : {type: Date, required: true, default: Date.now}
});

var Boek = mongoose.model('Boek', boekSchema);
module.exports = Boek;
```

Een Mongoose CRUD app maken

Verder ziet u dat het type voor een boek tamelijk strikt is vastgelegd. Dit doen we door velden op te geven en het type van het veld in het Mongoose-model te definiëren. Uiteraard gebeurd dit met een JavaScript-objectnotatie.



Hoofdletter

Omdat een variabele Boek hier als het ware als een klasse fungeert, schrijven we deze met een hoofdletter (deze techniek wordt ook gebruikt in programmeertalen als Java en C#). Instanties van die klasse (de eigenlijke boek-objecten) worden later met een kleine letter gemaakt.

Omdat de variabele Boek is afgeleid van een Mongoose-object, heeft het ook methods als `.find()` en `.save()`. Dit gaat zeker nog van pas komen, bijvoorbeeld om nieuwe boeken in de database op te slaan.

Een Mongoose CRUD app maken

- API-endpoints maken in de server
 - Gebruik maken van de Express module uit het vorige hoofdstuk.

```
// server.js - applicatie voor het ophalen en
// opslaan van boeken in MongoDB
var express = require('express');
var bodyParser = require('body-parser');
var db = require('./db');
var Boek = require('./models/boeken');

var app = express();
app.use(bodyParser.json());

// 1. Eenvoudige instructie
app.get('/api', function (req, res) {
    res.json({'Gebruik': 'voer een GET of POST-call uit naar /boeken'})
});

// 2. POST-endpoint: nieuw boek in de database plaatsen.
app.post('/api/boeken', function (req, res, next) {
    // TODO
});
app.listen(3000, function () {
    console.log('server gestart op poort 3000');
});
```

- Best opsplitsen a.d.h.v. Express router!

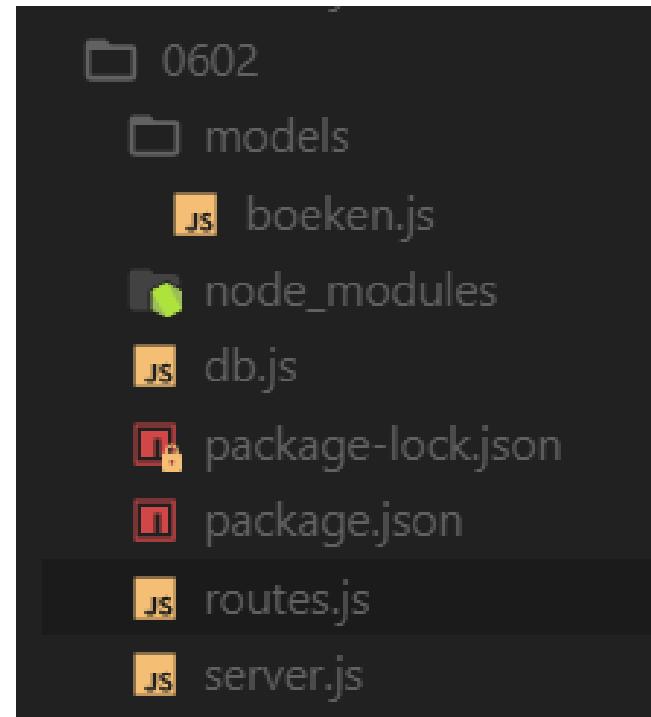
Een Mongoose CRUD app maken

- bodyParser.json()
 - In het vorige hoofdstuk maakte we gebruik van bodyParser.urlencoded(...)
 - Binnen Angular >2 gebruikt de HttpClientModule standaard JSON om een request te versturen
 - In onze backend geven we dus aan dat de body van onze binnenkomende request van het type JSON is via de middleware functie.

```
|  
app.use(bodyParser.json());
```

Een Mongoose CRUD app maken

- Structuur backend applicatie



Een Mongoose CRUD app maken

- POST-endpoint aanvullen met onze code:
 - In aparte file routes.ts

```
router.post('/boeken', (req, res) => {
  let boek = new Boek({
    titel: req.body.t any ,
    auteur: req.body.auteur,
    isbn: req.body.isbn
  });

  boek.save((err, boek) => {
    if(err)
      res.send(err);

    res.json(boek);
  });
});|
```

Een Mongoose CRUD app maken

- In deze code gebeurt het volgende:
 - Eerst wordt een nieuw boekobject gemaakt op basis van het model. De velden worden uit req.body gehaald
 - Omdat het boekobject is afgeleid van het Mongoose-object, heeft het een methode .save(). Hiermee wordt het boek opgeslagen in de database.
 - De _id is automatisch gegenereerd door MongoDB
 - Het resultaat wordt in Node afgehandeld door een callbackfunctie.

Een Mongoose CRUD app maken

- Code testen met Postman
 - Zorg ervoor dat de MongoDB-deamon is gestart
 - Start de node.js-server (node server.js)

The screenshot shows the Postman interface with the following details:

- Request URL: `http://localhost:3000/`
- Method: `POST`
- Endpoint: `http://localhost:3000/api/boeken`
- Body tab is selected.
- Content type: `JSON (application/json)`
- Raw JSON payload:

```
1 {  
2   "titel": "Lord of the rings: The Two towers",  
3   "auteur": "J.R.R. Tolkien",  
4   "ISBN": "1111222222"  
5 }
```

Een Mongoose CRUD app maken

- In de mongo shell kunnen we vervolgens checken of het boekobject inderdaad in de database is opgeslagen:

```
> use boeken  
switched to db boeken
```

```
> show collections  
boeks  
system.indexes  
> db.boeks.find()  
{ "_id" : ObjectId("55adecb8d1b4de802d73a097"), "titel" : "Handboek Word 2013",  
"auteur" : "Peter Kassenaar", "ISBN" : "877542342", "date" : ISODate("2015-07-21  
T06:54:48.997Z"), "__v" : 0 }  
>
```

- Mongoose heeft zelf de collectie “boeks” gemaakt (afgeleide van het Engels)

Een Mongoose CRUD app maken

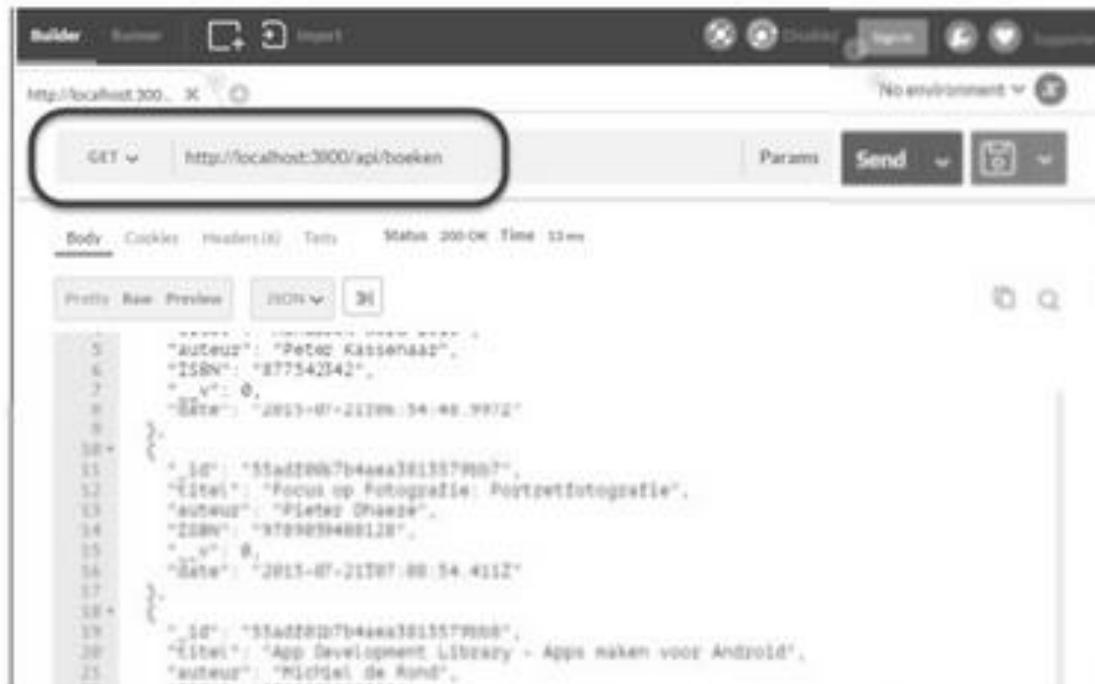
- Voorbeeldcode: /1202
- GET-requests verwerken
 - Dezelfde endpoint, ander soort request
 - Voorbeeldcode: /1202

```
router.get('/boeken', (req, res) => {
  Boek.find((err, boeken) => {
    if(err)
      res.send(err);

    res.json(boeken);
  });
});
```

Een Mongoose CRUD app maken

In deze code wordt rechtstreeks de method `.find()` op de klasse Boek aangeroepen. Omdat geen parameter is meegegeven (alleen een callbackfunctie), worden alle boeken geretourneerd.



```
GET http://localhost:3000/api/boeken
```

```
[{"_id": "55a2f80b7b44aa3813579b97", "Titel": "Focus op Fotografie: Portretfotografie", "Auteur": "Pieter Dhaeze", "ISBN": "9789059488128", "v": 0, "Date": "2015-07-21T08:54:48.997Z"}, {"_id": "55a2f80b7b44aa3813579b98", "Titel": "App Development Library - Apps maken voor Android", "Auteur": "Michiel de Rindt", "ISBN": "9789059488128", "v": 0, "Date": "2015-07-21T08:54:48.411Z"}]
```

Afbeelding 6.14 Er zijn inmiddels enkele boeken in de database geplaatst. Met een GET-request worden ze opgehaald.

Een Mongoose CRUD app maken



Specifiek boek ophalen

Wilt u een specifiek boek ophalen (met een aangegeven id, of van een bepaalde auteur)? Zorg dan voor een routeparameter en geef deze als object mee aan de callbackfunctie. De notatie wordt dan iets als Boek.find({ id: req.params.id}, function()...). Ook in de volgende paragraaf wordt een routeparameter gebruikt. Hij wordt ingezet om een bepaald document uit de database te kunnen verwijderen.

Een Mongoose CRUD app maken

- Delete-requests verwerken
 - `app.delete()`
 - Gebruik van parameters (zie hoofdstuk 12)
 - `Req.params.<paramnaam>`

```
app.delete('/api/boeken/:id', function(req, res, next){  
    Boek.remove({_id: req.params.id}, function(err, removed){  
        if(err) {  
            return next(err);  
        }  
        console.log(req.params.id);  
        res.json(removed);  
    });  
});
```

Een Mongoose CRUD app maken

The screenshot shows a POSTMAN interface with the following details:

- URL: `http://localhost:3000/`
- Method: `DELETE`
- Path: `http://localhost:3000/api/boeken/5a5a7a4611812ca704316aa9`
- Type: `No Auth`
- Body (JSON):

```
1 {  
2   "n": 1,  
3   "ok": 1  
4 }
```
- Status: `200 OK`

Een Mongoose CRUD app maken

- Put-request verwerken
 - app.put()
 - Gebruik _id om boek op te halen & te updaten

```
app.put('/boeken', (req, res) => {
  Boek.findById(req.body._id, (err, boek) =>{
    boek.titel = req.body.titel;
    boek.auteur = req.body.auteur;
    boek.isbn = req.body.isbn;
    boek.date = req.body.date;

    boek.save((saveErr, saveBoek) => {
      if(saveErr)
        res.send(saveErr);

      res.send(saveBoek);
    });
  });
});
```

Een Mongoose CRUD app maken

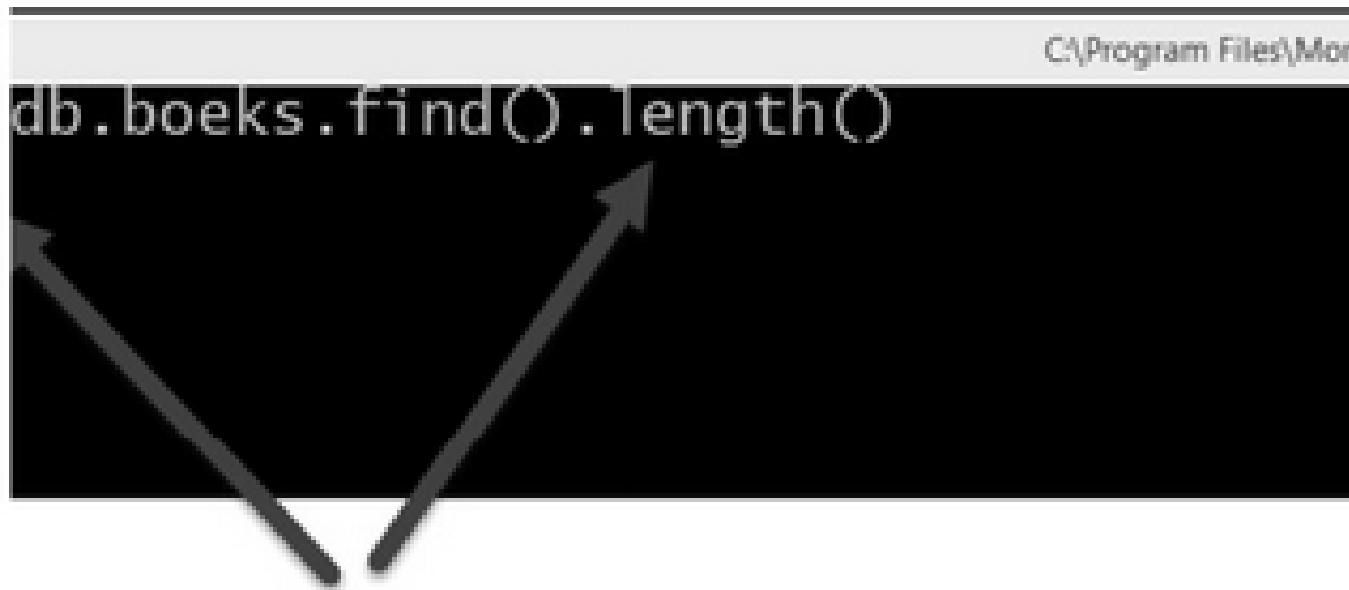


Aantal items opvragen

Met de MongoDB-functie `.length()` is op te vragen hoeveel documenten in een bepaalde collectie aanwezig zijn. Dit is vergelijkbaar met de eigenschap `.length` voor JavaScript-arrays.

Alleen is het in MongoDB een functie die wordt uitgevoerd op het resultaat van `.find()`, dus moet u een hakenpaar gebruiken. De complete notatie is bijvoorbeeld `database.boek.find().length()`.

Een Mongoose CRUD app maken



Afbeelding 6.16 *Na het verwijderen zijn nog drie boeken over in de database.*

Een Mongoose CRUD app maken

- Onze API beschikbaar maken voor een Angular Front-end
 - Cross-Origin Resource Sharing (CORS)
 - Toevoegen van localhost:4200 voor alle soorten requests via middleware functie in server.js

```
app.use(function(req, res, next){  
  res.setHeader('Access-Control-Allow-Origin', 'http://localhost:4200');  
  res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, PATCH, DELETE');  
  res.setHeader("Access-Control-Allow-Headers", "Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");  
  next();  
});
```

Een Angular frontend applicatie voorzien

- Een Angular frontend applicatie bouwen
ng new mongooseFrontend
- Voorbeeldcode: 1202_frontend
- Bevat volgende elementen:
 - BoekService: aanspreken van de API
 - Boek model: model voor het mappen van boeken
 - AppComponent: Weergeven van onderstaande components
 - BoekListComponent: Weergeven van boeken in een tabel
 - AddBoekComponent: Formulier voor het toevoegen van een boek
 - EditBoekComponent: Formulier voor het bewerken van een boek

Een Angular frontend applicatie voorzien

Titel:

Auteur:

ISBN:

Datum uitgave: dd/mm/jjjj

Toevoegen

Titel	Auteur	ISBN nummer	Datum uitgave	Acties
The hobbit	J.R.R. Tolkien	123-45678-90	2019-01-18T00:00:00.000Z	edit delete
Divergent	Veronica Roth	123-45678-90	2019-01-17T13:30:31.298Z	edit delete
Testboek123	Testauteur	111-1111111-11	2019-01-17T13:31:04.436Z	edit delete

Oefeningen: MongoDB & Mongoose

Oefening 1

Schrijf een NodeJS Webserver met Express & Mongoose. Voor deze applicatie maak je gebruik van een database met de naam "H13oef". Hierin gebruik je een collection "cars" waarin je car objecten aanmaakt. Deze objecten hebben volgende structuur:

```
> db.cars.find().pretty()
{
  "_id" : ObjectId("5a646b5cadda6c304551f67c"),
  "manufacturer" : "Porsche",
  "model" : "911",
  "price" : 135000,
  "wiki" : "http://en.wikipedia.org/wiki/Porsche_997"
}
{
  "_id" : ObjectId("5a646b7aadda6c304551f67d"),
  "manufacturer" : "Nissan",
  "model" : "GT-R",
  "price" : 80000,
  "wiki" : "http://en.wikipedia.org/wiki/Nissan_Gt-r"
}
{
  "_id" : ObjectId("5a646ba5adda6c304551f67e"),
  "manufacturer" : "BMW",
  "model" : "M3",
  "price" : 60500,
  "wiki" : "http://en.wikipedia.org/wiki/Bmw_m3"
}
```

Maak het nodige model in Mongoose en voorzie volgende routes in je NodeJS Webservers:

- app.get('/cars'): Geeft alle car objecten terug
- app.get('/cars/:id'): Krijg een specifiek car object terug

Zorg ervoor dat de body-parser JSON formaat gebruikt. Test je routes uit met een tool zoals postman.

Oefening 2

Maak een eenvoudige Angular applicatie die de data uit de vorige oefening ophaalt en toont in de console.

Oefening 3

Vertrek vanuit de NodeJS webserver van oefening 1. Voorzie volgende routes bovenop de reeds bestaande routes:

- app.post('/cars'): Voegt een nieuw car-object toe aan de databank
- app.delete('/cars/:id'): Verwijder een car-object op basis van de id.
- app.put('/cars'): Update car-object op basis van de id.

Test de routes uit met een tool zoals postman.

Oefening 4

Vertrek vanuit oefening 2. Zorg voor een angular applicatie waarbij je basis CRUD functionaliteit voorziet.

- Voor het tonen van de data maak je gebruik van een car-detail component. Voor elk car object wordt deze component geladen. Voorzie ergens een link naar het wikipedia artikel.
- Voor het toevoegen van data maak je gebruik van een model driven form.

Uitbreidung: Voorzie een bank-account Service die start met een saldo van € 120 000,00. Voorzie in de car-detail component een knop “buy” waarmee je een auto kan kopen. Het bedrag van de auto wordt dan afgetrokken van je bank saldo.

Probeer gebruik te maken van de 3rd party module toaster (`npm install ng2-toastr`) om een melding te geven “auto gekocht”. Indien je niet genoeg geld hebt, toon je de melding “Niet voldoende geld!” van diezelfde module.

CSS Preprocessors

SASS



Wat is SASS

- CSS Preprocessor
 - scriptingtaal
 - Voegt functionaliteit toe aan je stylesheet markup
- Verschillende CSS preprocessors bestaan:
 - **SaSS:** Syntactically Awesome Style Sheets
 - Less
 - Stylus
- SaSS compileert naar CSS

Waarom SaSS

- CSS heeft een achterhaalde opbouw
 - Specifieke prefixes
 - Vervangen van kleurcodes in een project
 - Dezelfde stijlen komen voor in verschillende CSS blokken
 - Complexe en lange CSS selectoren worden gebruikt doorheen het project

Waarom SASS

- Sass lost deze problemen op
 - Maakt CSS schaalbaar, herbruikbaar en slim
 - 2 conventies:
 - .SASS
 - .SCSS
- SASS live testen in de browser:
<https://www.sassmeister.com/>

.SCSS

- SCSS volgt de algemene CSS bracket syntax

```
body #main .value {  
    color: #eee;  
  
    a.link{  
        color: #101010;  
        text-decoration: none;  
    }  
}
```

.SASS

- SASS werkt aan de hand van indentation

```
body #main .value
    color: #eeee

a.link
    color: #101010
    text-decoration: none
```

SASS in Angular

- SASS is eenvoudig te integreren in nieuwe Angular projecten

```
C:\Users\20003125\workspace>ng new voorbeeldsass
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use?
  CSS
  > SCSS  [ http://sass-lang.com      ]
    SASS  [ http://sass-lang.com      ]
    LESS   [ http://lesscss.org       ]
    Stylus [ http://stylus-lang.com ]
```

- Of is bestaande projecten

```
ng config schematics.@schematics/angular:component.styleext scss
```

SaSS functionaliteiten

- Gebruik van variabelen
- Nesting
- Inheritance
- Mixins
- Partials
- Operators

SASS variabelen

- Variabelen maken het mogelijk om een waarde aan een placeholder te koppelen

```
$varname: waarde  
selector {  
    property: $varname;  
}
```

- Werkt met hex-values, strings, kleuren, getallen, booleans,...
- Variabelen zijn scoped
 - Enkel beschikbaar in de blok waar ze gedeclareerd worden

SASS Variabelen

```
$font-stack:    Helvetica, sans-serif;  
$primary-color: #333;  
  
body {  
    font: 100% $font-stack;  
    color: $primary-color;  
}
```

SASS Nesting

- Nesting maakt lange css selectoren onnodig
- Veel vaak terugkomende niveaus zijn maar één maal gedefinieerd
- Duidelijke link in structuur van HTML en SASS

```
nav {  
    ul {  
        margin: 0;  
        padding: 0;  
        list-style: none;  
    }  
  
    li { display: inline-block; }  
  
    a {  
        display: block;  
        padding: 6px 12px;  
        text-decoration: none;  
    }  
}
```

SASS Nesting

```
1 body #main .value {  
2   color: #eee;  
3  
4   a.link {  
5     color: #101010;  
6     text-decoration: none;  
7   }  
8  
9   span.highlight {  
10    font-weight: bolder;  
11  }  
12 }
```

```
1 body #main .value {  
2   color: #eee;  
3 }  
4 body #main .value a.link {  
5   color: #101010;  
6   text-decoration: none;  
7 }  
8 body #main .value span.highlight {  
9   font-weight: bolder;  
10 }  
11
```

SASS Inheritance

- Een van de handigste functies van SASS
- @Extend geeft ons de mogelijkheid om selectoren uit te breiden
 - Laat overerven van blokken css properties

```
%equal-heights {  
    display: flex;  
    flex-wrap: wrap;  
}  
  
.message {  
    @extend %equal-heights;  
}
```

- Verhoogt herbruikbaarheid van onze CSS code

SASS Inheritance

```
1  %message-shared {  
2      border: 1px solid #ccc;  
3      padding: 10px;  
4      color: #333;  
5  }  
6  
7  .message {  
8      @extend %message-shared;  
9  }  
10  
11 .success {  
12     @extend %message-shared;  
13     border-color: green;  
14  }  
15  
16 .error {  
17     @extend %message-shared;  
18     border-color: red;  
19  }  
20  
21 .warning {  
22     @extend %message-shared;  
23     border-color: yellow;  
24  }  
  
1  .message, .success, .error, .warning {  
2      border: 1px solid #ccc;  
3      padding: 10px;  
4      color: #333;  
5  }  
6  
7  .success {  
8      border-color: green;  
9  }  
10  
11 .error {  
12     border-color: red;  
13  }  
14  
15 .warning {  
16     border-color: yellow;  
17  }  
18
```

SASS Mixins

- Vaak extra css regels voorzien om properties te ondersteunen op verschillende browsers
- Vaak bundel van css regels die in verschillende selectoren gebruikt wordt
- In SASS kan dit in een mixin voorzien worden om vervolgens doorheen het project te gebruiken
- Mixins kunnen ook argumenten meekrijgen in de vorm van een variabele

SASS Mixins

```
1  @mixin overlay() {  
2      bottom: 0;  
3      left: 0;  
4      position: absolute;  
5      right: 0;  
6      top: 0;  
7  }  
8  
9  .modal-background{  
10     @include overlay();  
11     background: black;  
12     opacity: 0.9;  
13 }
```

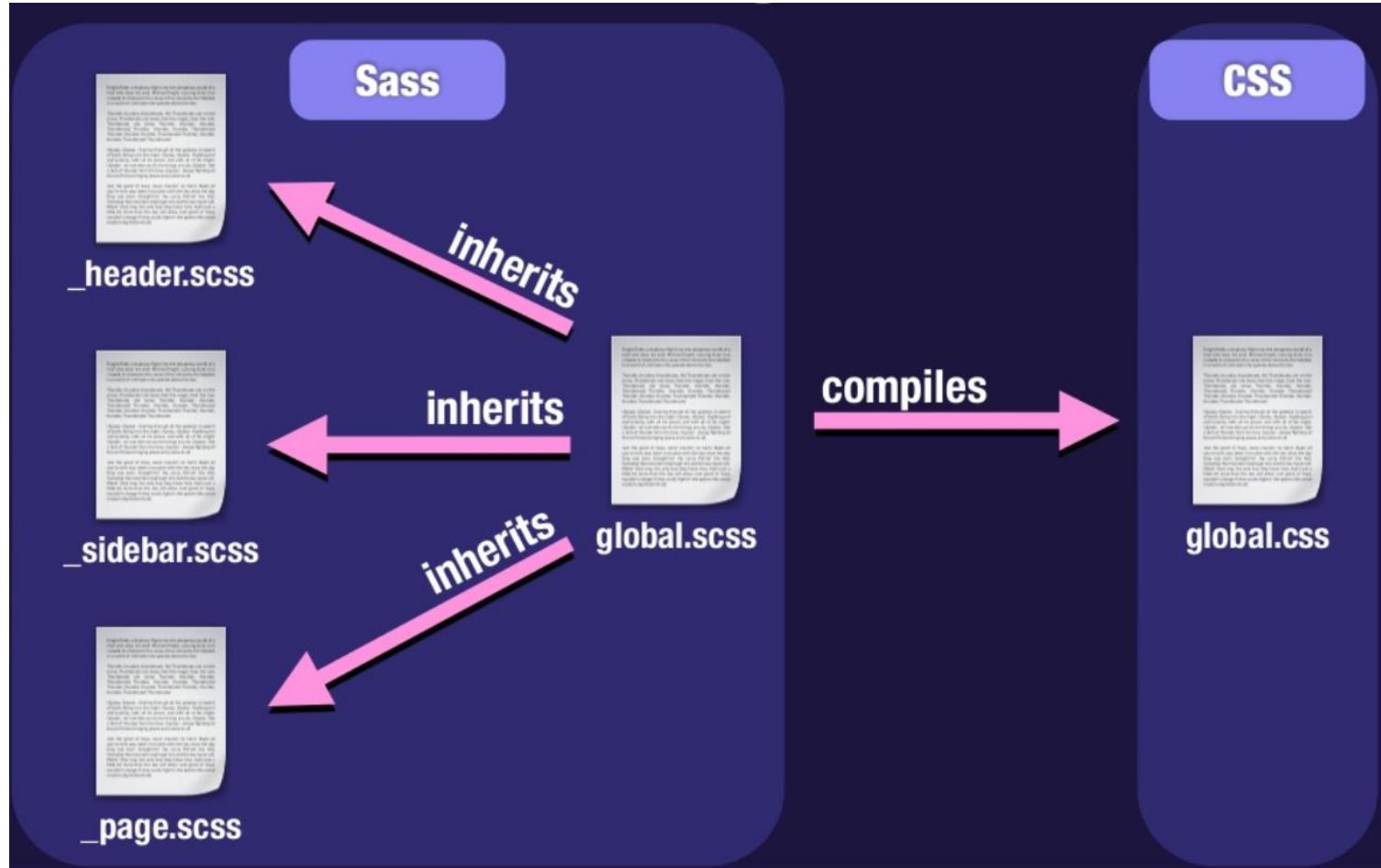
```
1  .modal-background {  
2      bottom: 0;  
3      left: 0;  
4      position: absolute;  
5      right: 0;  
6      top: 0;  
7      background: black;  
8      opacity: 0.9;  
9  }  
10 }
```

Partials

- CSS meestal in één bestand
- Partials maakt het mogelijk om CSS op te splitsen in meerdere bestanden (partial files)
- Partials starten altijd met een underscore (_)
- In de global.scss file importeren:

```
@import("header", "sidebar", "page");
```

Partials



Operatoren

- In SASS kunnen wiskundige operatoren zoals + - * en % gebruikt worden

```
1 .container {  
2   width: 100%;  
3 }  
4  
5 article[role="main"] {  
6   float: left;  
7   width: 600px / 960px * 100%;  
8 }  
9  
10 aside[role="complementary"] {  
11   float: right;  
12   width: 300px / 960px * 100%;  
13 }
```

```
1 .container {  
2   width: 100%;  
3 }  
4  
5 article[role="main"] {  
6   float: left;  
7   width: 62.5%;  
8 }  
9  
10 aside[role="complementary"] {  
11   float: right;  
12   width: 31.25%;  
13 }
```