Laravel 2022 - L2 (orgineel/Bootstrap)

- Introductie
- Bootstrap Installatie
- Hoe werkt Bootstrap?
- Bootstrap Grid systeem
- Navigatie maken met Bootstrap
- De kunst van blade files
- Productpagina maken en designen
- Producten toevoegen in jouw database
- Producten laten zien vanuit de database en de kracht van Eloquent

Introductie

In Laravel Level 2 ga je jouw webshop verder uitbreiden.

Zo wil je bijvoorbeeld producten tonen in jouw webshop door ze uit de database op te halen. Daarnaast geven wij jou een introductie van het CSS framework Bootstrap en maken we meerdere pagina's door views en controllers aan te maken en de routes goed in te stellen.

Hieronder is er een lijstje wat er in Laravel level 2 wordt behandeld:

- Meer pagina's (views/routings/controllers toevoegen)
- Introductie met Bootstrap
- Een losse pagina maken waar je al jouw producten laat zien

Bootstrap Installatie

Inleiding

Voor de vormgeving en styling gaan we Bootstrap gebruiken.

Bootstrap is een CSS (en Javascript) framework. Je kunt het zien als een groot aantal voorgedefinieerde styles die je kan gebruiken. Op die manier kun je snel 'even' je site oppimpen.

Wat is het en hoe werkt het?

Kijk maar eens op

> https://getbootstrap.com/docs/5.0/examples/cheatsheet/

Daar zie je allemaal user interface-elementen die met Bootstrap zijn vormgegeven. Je herkent vast wel wat van Yii want daar hebben we ook gebruik gemaakt van Bootstrap.

Alle uitgebreide informatie over Bootstrap vind je op:

> https://getbootstrap.com/docs/5.2/getting-started/introduction/

Installatie

Installatie kan op veel manieren.

De eenvoudigste manier is om de volgende regel in de <head> van je pagina te plaatsen:

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeuOxjzrPF/et3URy9Bv1WTRi" crossorigin="anonymous">
```

Hiermee laadt je de (hele grote) CSS style sheet.

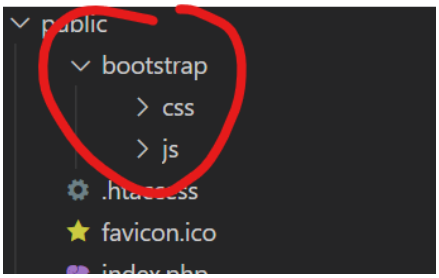
Installatie via download

Een iets beter alternatief is om de stylesheet in je project te plaatsen. Dan zijn we niet afhankelijk van een andere server.

Download daarvoor de "Compiled CSS and JS" files.

Pak de zip file uit en plaats de folder die in de zip file staat in jouw Laravel project in de public folder.

Je krijgt dus dit:



De folder in de zip file heet zoiets als "bootstrap-5.2.2-dist". Deze folder kopieer je en hernoem je daarna zodat de naam van de folder "bootstrap" wordt.

In je header kun je nu de volgende regel plaatsen. Dit zorgt ervoor dat de gedownloade versie van Bootstrap wordt geladen.

```
<link href="{{asset('bootstrap/css/bootstrap.min.css')}}" rel="stylesheet">
```

Opdracht

Installeer Bootstrap in je laravel project via de download-methode.

Plaats vervolgens deze HTML code in je <body> van de product-read.blade.php file. Plaats de code vóór de @foreach.

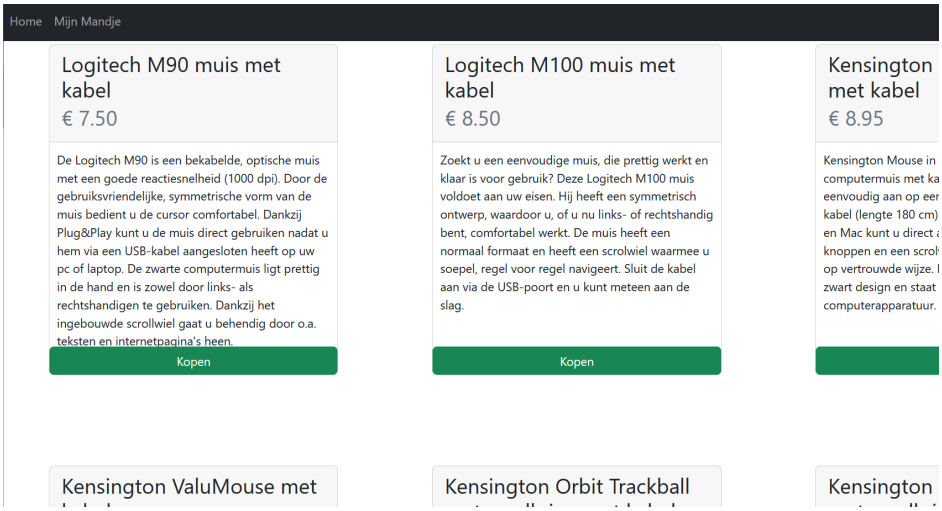
Dus regel 2 t/m10 uit de volgende code worden op de juiste plaats in prodcut-read.blade.php geplaatst.

```
....
<body>
  <nav class="navbar navbar-expand-md navbar-dark fixed-top bg-dark">
      <div class="collapse navbar-collapse">
          <ul class="navbar-nav mr-auto">
              <li class="nav-item"><a class="nav-link" href="/">Home</a></li>
              <li class="nav-item"><a class="nav-link right" href="/mandje">Mijn
Mandje</a></li>
          </ul>
      </div>
  </nav>

  @foreach ($products as $product)
```

```
....
```

Laadt je productpagina opnieuw en je zult een menu-bar zien, die er zo uit ziet:



Inleveren

Screenshot van je product pagina mét Bootstrap menu

--

Hoe werkt Bootstrap?

Zoals in de vorige pagina is verteld kan je Bootstrap gebruiken om makkelijk jouw HTML op te maken zonder al te veel te CSS'en.

Bootstrap maakt gebruik van de attribuut `class=""`. Deze attribuut gebruik je ook weleens om opmaak te doen in CSS door een eigen class naam te geven. In het geval van Bootstrap hebben ze hun eigen class namen bedacht. Zo kan je bijvoorbeeld `text-success` gebruiken om een groene tekst te maken.

Opdracht .1

Open een view (een .blade.php file) en stop dit stukje code in:

```
<span>Maak deze woord</span> <span class="">ROOD!</span>
```

Plaats in de `class` attribuut de classname van Bootstrap die de text rood maakt. Weet je die niet?

Zie Colors · Bootstrap v5.2 (getbootstrap.com)

Jouw text is nu rood!

Je kan ook meerdere classnamen van Bootstrap tegelijk gebruiken. Zo kan je bijvoorbeeld ook een achtergrondkleur toevoegen met `bg-bootstrapkleur`. Om te weten welke bootstrapkleuren er zijn kan je terug kijken naar de documentatie.

Opdracht .2

Open de file van de vorige opdracht.

Voeg een Bootstrap lichtgrijze achtergrondkleur (kleur light) toe aan de `<span>`.

Jouw span heeft nu een rode text met een grijze achtergrondkleur.

Buttons

image-1663545316481.png

Bootstrap heeft ook voorgekauwde opmaak voor Buttons. Zie Buttons · Bootstrap v5.2 (getbootstrap.com)

Zoals je ziet valt het op dat ze dezelfde classnamen gebruiken om kleuren van een element aan te geven. Deze kleuren worden standaard door bootstrap ingesteld en kan dus ook voor andere

compententen gebruikt worden.

Opdracht .3

Open de file van de vorige opdracht

Maak een `button` element aan met een tekst "Klik hier"

Voeg een dark button class toe aan de button. Zorg ervoor dat je checkt welke classnamen je allemaal moet toevoegen om het te laten werken! Buttons · Bootstrap v5.2 (getbootstrap.com)

Je hebt nu een button met een opmaak van Bootstrap gebruikt!

Wat lever je in?

- Een screenshot van jouw blade.php file waarin je hebt gewerkt met Bootstrap

Andere componenten

Zo zijn er veel meer andere elementen die je kan opmaken. Een button met een Bootstrap opmaak is er een van. Als je meer wil zien wat Bootstrap kan aanbieden, dan raden we je ten zeerste aan om de documentatie van Bootstrap te gaan kijken:

Get started with Bootstrap · Bootstrap v5.2 (getbootstrap.com)

Wat heb je geleerd?

- Je hebt geleerd hoe Bootstrap in elkaar zit
- Je hebt geleerd hoe je tekst- en achtergrondkleuren gebruikt.
- Je hebt geleerd welke standaardkleuren Bootstrap heeft
- Je hebt geleerd hoe je een button maakt met een Bootstrap thema.

Bootstrap Grid systeem

Bootstrap maakt niet alleen jouw opmaak makkelijker, maar ook jouw structuur!

Bootstrap heeft het ook makkelijker gemaakt om flexbox te gebruiken. Met flexbox kan je jouw elementen verdelen in verschillende posities. Bootstrap maakt ook gebruik van flexbox, maar nu is het simpel voor je gemaakt en kan je net als de opmaak in classnamen aanroepen.

De layout van Bootstrap

image-1663268535257.png

In Bootstrap heeft het gridysteem een standaard layout. Wij gaan per onderdeel uitleggen hoe dat in elkaar zit:

Container

De container is de buitenste laag. Ze zijn verantwoordelijk om al jouw content responsive te maken.
Je hebt twee type containers die je kan gebruiken om jouw pagina responsvie te maken:

- `.container`: past de max-width aan op basis van jouw schermresolutie. Zie Containers · Bootstrap v5.0 (getbootstrap.com) welke resoluties dat zijn.
- `.container-fluid`: heeft 100% breedte bij alle resoluties

De meeste sites gebruiken .container, omdat het aantrekkelijker maakt voor de gebruiker om jouw content te lezen. Zie hier een voorbeeld van zalando.nl die een .container design gebruikt:

image-1663406146373.png

Zoals je ziet laat Zalando.nl altijd ruimte vrij aan de linker en rechter kant. Mensen vinden het gemakkelijker om te lezen als alle content compact bij elkaar zit. Dat is ook makkelijker voor de gebruiker om te klikken. Dan hoeven zie niet over de hele pagina te klikken.

Rows en Columns

In de container valt het ook op dat Zalando meerdere rows heeft. Binnen deze rows heeft het ook kolommen (columns). Zie hier het voorbeeld:

image-1663405980840.png

Je ziet in een row twee kolommen: `col-8` en `col-4`. Dit zijn de kolommen (columns). In totaal passen er 12 kolommen in een row. Deze kolommen gebruik jouw ruimte in te delen. Daarom zie je bij de eerste row een col-8. Dat is de text. En bij col-4 zie je een plaatje. Dus de verhouding is dan 67% en 33%.

Als je het nog een beetje ingewikkeld vindt, dan kan je nog hiernaar kijken: Bootstrap 5 grid system - examples and tutorial (mdbootstrap.com)

Opdracht: Spelen met Bootstrap

Nu is het tijd om aan de slag te gaan hoe Bootstrap werkt:

image-1663545649307.png

Standaard structuur van Bootstrap

Stappenplan:

- Maak een container
  - Maak in de container een row
    - Maak in de row twee kolommen met de verhouding 25% en 75%
      - In de 25% moet een rode achtergrond van Bootstrap staan
      - In de 75% zet je een titel en een paragraaf met text

Wat lever je in?

1. Een screenshot van jouw .blade.php file waarin je gewerkt hebt
2. Een screenshot van jouw browser met de resultaten van de file.

Navigatie maken met Bootstrap

Om een goede start te maken met Bootstrap, bouwen we eerst een navigatiebar in Bootstrap.

Ga naar: Navbar · Bootstrap v5.2 (getbootstrap.com)

Scroll naar 'Supported Content'. Daar zie je verschillende classnames die je kan gebruiken voor de navigatie.

Meest gebruikte navigation classes voor een Bootstrap navigatie

- `.navbar-brand` gebruik je voor je logo of bedrijfsnaam.
- `.navbar-nav` voor een volledige hoogte van de navigatie. Daarnaast maak je de navigatie responsive (voor mobiel dus) en krijg je een wafel (oftewel een toggler) te zien als je op mobiel ziet.
- `.navbar-toggler` gebruikt de toggle functie van bootstrap. Die kan je gebruiken om bijvoorbeeld een toggler toe te voegen aan je navigatie.
- `.navbar-text` om verticaal gecentreerde text toe te voegen.

Zo ziet een navigatie eruit in Bootsatrap:

image-1662966077287.png

```
<nav class="navbar navbar-expand-lg bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Navbar</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-
target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false"
aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav me-auto mb-2 mb-lg-0">
        <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="#">Home</a>
        </li>
        <li class="nav-item">
```

```
          <a class="nav-link" href="#">Link</a>
        </li>
        <li class="nav-item dropdown">
          <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown"
aria-expanded="false">
            Dropdown
          </a>
          <ul class="dropdown-menu">
            <li><a class="dropdown-item" href="#">Action</a></li>
            <li><a class="dropdown-item" href="#">Another action</a></li>
            <li><hr class="dropdown-divider"></li>
            <li><a class="dropdown-item" href="#">Something else here</a></li>
          </ul>
        </li>
        <li class="nav-item">
          <a class="nav-link disabled">Disabled</a>
        </li>
      </ul>
      <form class="d-flex" role="search">
        <input class="form-control me-2" type="search" placeholder="Search" aria-
label="Search">
        <button class="btn btn-outline-success" type="submit">Search</button>
      </form>
    </div>
  </div>
</nav>
```

Dit is de code van de navigatie. En nu denk je van: zo veel code?! Maar als we alleen de belangrijkste stukjes code meenemen dan valt het wel mee.

Opdracht: Navigatie

1. Kopieer de code van de Bootstrap Navigatie.
2. Zoals op dat plaatje te zien, heb je een search bar met een search knopje. Dat hebben we niet nodig. Dus verwijder dat stukje uit de code.
3. De dropdown hebben we ook niet nodig. Verwijder dat stukje code.
4. De disabled link hebben we ook niet nodig. Ook dat stukje code verwijderen.

Zoals je ziet is de code nu veel kleiner geworden. Nu gaan we kijken wat er allemaal in zit:

Bedrijfsnaam/logo

`<a class="navbar-brand" href="#">Navbar</a>` is de plek waar jouw (bedrijfs)logo moet staan.

Je ziet in de class attribuut `class="navbar-brand"`. Deze classnaam is van Bootstrap. Deze class zorgt ervoor dat de opmaak van jouw bedrijfsnaam of logo goed in de navigatie past.

Navbar toggler

```
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-
target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false"
aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
```

Dit stukje code is alleen op het scherm te zien als je op mobiel zit. Op de mobiel zie je dan een wafeltje op het scherm. Als je erop klikt zie je jouw navigatie items.

De `<span>` in de `<button>` tag is het wafeltje.

Navbar items

Het laatste stukje code zie je de items van de navigatie. Zoals je ziet heb je een Home item en een Link item.

```
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav me-auto mb-2 mb-lg-0">
        <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="#">Home</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Link</a>
        </li>
      </ul>
    </div>
```

- In het stukje code zie je ook een `<ul>` en `<li>`. Herken je deze elementen nog? Die gebruik je om een unordered list te maken.
  - Elk `<li>` is een navbar-item.
- Bootstrap gebruikt de unordered list om de navigatie items naast elkaar te plaatsen. Zoals je ziet zie je helemaal geen bullet points. Bootstrap heeft die dus weggehaald en geoptimaliseerd om in een navigatie te kunnen gebruiken.

- In de `<a>` element zie je een class attribuut met een `nav-link` erin. Deze classnaam is verplicht om `<a>` tags te gebruiken in de navigatie.
    - Bij één van de `<a>` element zie je ook een "active" classnaam hebben. Dat is om aan te duiden welke navigatie item geselecteerd is op basis van welke pagina je staat.

Opdracht: voeg extra navbar items toe.

1. Vervang de `nav-link` naam van "Link" naar "Products"
2. Maak twee nieuwe navigatie items toe: "Login" en "Registreren"
3. Zorg ervoor dat ze allebei een `<li>` hebben met een classnaam nav-item hebben
    1. Maak daarbinnen ook een `<a>` met de classnaam "nav-link"
4. Verander de achtergrondkleur (gebruik één van de bootstrap kleuren).

Wat lever je in?

1. Screenshot van jouw code met jouw persoonlijke navbar

De kunst van blade files

Nu je een prachtige navigatiebar hebt gemaakt, zou je het dus ook op alle andere pagina's de code van de hele navigatie moeten plaatsen toch? Nou nee... Daarvoor hebben we Blades!

Wat is een Blade?

Blade is een template engine dat standaard is toegevoegd in Laravel. Je kan gemakkelijk php code in jouw .blade.php file stoppen. Ook kan je bladefiles hergebruiken in andere bladefiles. Dat betekent dus ook dat je makkelijk jouw navigatiebar kan hergebruiken!

Hoe hergebruik je jouw navigatie met Blade?

Voordat we beginnen met coderen, moet je eerst begrijpen welke syntax er is in Blade.

Zie hier: Blade Templates - Laravel - The PHP Framework For Web Artisans

Om jouw navigatie makkelijk opnieuw te laten gebruiken, gaan we drie blade variabeles gebruiken:
`@section` - door deze variabele om een stukje code te plaatsen, geef je dat gedeelte een sectienaam
`@yield` - plaatst de `@section` op de plek waar `@yield` staat. Dat kan je doen door de sectionnaam in `@yield` te stoppen
`@include` - voegt een andere `.blade.php` file toe in de huidige `.blade.php` file.
`@extends` - breidt de `.blade.php` uit met een andere `.blade.php` file

Hoe gaat dit visueel te werk?

Omdat dit een beetje ingewikkeld is om in tekst uit te leggen, is hieronder een visuele overzicht hoe een herbruikbare navigatie eruit moet zien:

image-1663542895344.jpg

Wat doet elke file?

`iets.blade.php`

Daar plaats je gewoon jouw content neer die je op de pagina wil tonen. Daar verwijs je ook met je routes en controller naartoe!
Deze content zit om een `@section`. De reden hiervoor is zodat je in een andere file deze sectie erin kan stoppen. Zoals je ziet ga je in de reusable file (reusable.blade.php) dezelfde sectie `@yield` 'en.

Maar om dat te kunnen doen, moet iets.blade.php file eerst aan de reusable.blade.php file worden verlengd. Dat doe je dus met `@extends('naam van de file')`. In dit geval moet er dus '*reuse*' staan.

`reuse.blade.php`

De reuse file wordt de pagina loader. Iedere keer als een wordt ingeladen, is het de bedoeling dat deze file geladen wordt door de geëxtende file (`iets.blade.php` in dit geval);. Zoals je daar ziet zet je daar jouw standaard HTML structuur in.

- Je ziet `@yield('content')`. Dat is dus de section die hij pakt van de iets.blade.php.
- Je ziet `@include('navigation')`. Hij zoekt naar een file die `navigation.blade.php` heet, en voegt de code toe in `reuse.blade.php` file.

`navigation.blade.php`

Het spreekt voor zich. Jouw gemaakte navigatie moet je dus hierin zetten! Je kan ook jouw navigatie in de reuse.blade.php stoppen, maar om alles overzichtelijk te maken kan je het beste in een losse file doen.

Opdracht: maak jouw navigatie herbruikbaar!

Stappenplan:

- Open de views folder en maak een file die **reuse.blade.php** (officieel moet het **app.blade.php** heten).
  - Voeg jouw standaard HTML structuur toe.
  - Voeg `@include('navigation')` en `@yield('content')` toe in jouw `<body>`
- Maak de navigatiefile en noem het naar **navigation.blade.php**
  - Plaats jouw naviagtiebar in de file
- Ga naar jouw file waar jouw content geplaatst wordt (kies bijvoorbeeld de file waar je de navigatie van de vorige opdracht hebt gemaakt).
  - Maak de file leeg en voeg `@extends('reuse')` (of officieel `@extends('app')`) toe.
  - Voeg eronder @section en @endsection. En schrijf wat content tussenin.

Wat lever je in?

1. Jouw `.blade.php` file met content erin
2. Jouw `reuse.blade.php` of app.blade.php file
3. Jouw `navigation.blade.php` file

>

Gefeliciteerd! Je hebt jouw content en navigatiebar dynamisch gemaakt met Blade!

Productpagina maken en designen

Nu je weet hoe je jouw content dynamisch maakt, gaan we nu een productpagina maken!

Hoe maken we een productpagina?

Eerst maken we een duidelijk overzicht in stappen hoe je een productpagina maakt.

1. Maak een view aan
2. Maak een controller aan
3. Link de controller aan de routes

Opdracht: maak jouw productpagina

1. Maak eerst een view aan. Zorg ervoor dat je `@extends('reusefile')` en `@section('content')` toevoegt!
2. Als de view aangemaakt is, maak je een controller aan voor de productpagina. Ik zou het zelf `ProductController` noemen. Vergeet niet om hiervoor de terminal te gebruiken om een controller te maken!
    1. Zorg ervoor dat je in jouw controller een `function` maakt. Daarin `return` je jouw view
3. Ga daarna naar de routes. Maak een nieuwe route aan en zorg ervoor dat `/products` naar de `ProductController` gaat.

Nu jouw productpagina gelinkt is aan /products, gaan we nu designen in Bootstrap!

Bootstrap Cards

Een productpagina worden vaak gemaakt in productkaarten. En nu vraag je je af: moet ik nu hele productkaart maken zoals dit?!

image-1663876497176.png

Antwoord: nee! Gelukkig heb je hier Bootstrap voor! Het plaatje hierboven is dus gemaakt in Bootstrap. Zie de documentatie: Cards · Bootstrap v5.2 (getbootstrap.com)

Opdracht: maak een productkaart

Probeer nu een productkaart te maken met Bootstrap:

1. Maak gebruik van het Bootstrap gridsysteem en maak een `container`, `row` en een `col-3` .
2. Voeg een card-body toe
3. Voeg in de card-body een card-title toe en bedenk een titel
4. Voeg in de card-body een card-subtitle toe en bedenk een prijs
5. Voeg in de card-body een card-text toe en bedenk een beschrijving voor jouw product
6. Maak een knopje met de tekst 'Koop' in een blauwe Bootstrap kleur
7. Kopiëer de kolom met de card en paste een paar keer naast elkaar.
8. Verander de teksten van de andere productkaarten

Design nog in jouw eigen smaak!

Wat lever je in?

- Een screenshot van jouw browser met jouw producten
- Screenshots van jouw code waarvan
  - Jouw ProductController.php file
  - Jouw productpagina view
  - Jouw routes.php

Producten toevoegen in jouw database

Nu heb je producten hard-coded toegevoegd. Natuurlijk wil je dat deze producten vanuit de database komen. Daarom gaan we aan de slag om de productdetails die je nu op de productpagina hebt gemaakt, in de database te gaan stoppen.

Product migration en model maken

Om jouw producten in de database te hebben, moet je eerst een tabel maken en die linken aan jouw Laravel project (met model)

Wat zijn migration en models ook alweer?

Migrations: is een 'blauwprint' van jouw tabel. Je geeft aan welke kolommen de tabel heeft en welke datatypes. Vervolgens migreer je het blauwprint in de database. Zo wordt er dan automatisch een tabel gemaakt!

Models: is de 'M' van het MVC model. Models zijn verantwoordelijk voor de link tussen het project en de database (dus niet de migrations! Die regelt alleen de blauwprinten). Zo kan je bijvoorbeeld dan in de controller een model aanroepen om data op te halen uit de database.

Opdracht: een migration en model maken

Laten we beginnen door eerst een migration aan te maken via de console. Hier een stappenlijst:

1. Maak een migration aan via de terminal met de naam `create_products_table`
2. Open de migrationfile en bedenk welke kolommen nodig zijn voor jouw tabel. Kijk wat je op jouw productpagina hebt. Hier alvast een aantal:
   - id
   - title
   - price
3. Voeg deze kolommen toe in jouw migration file
4. Voer de migration uit via de terminal. Mocht je al de tabel hebben, dan moet je het refreshen

5. Check in jouw PHPMyAdmin of jouw tabel is toegevoegd
6. Maak een model via de terminal. Noem de file `Product`
7. Open de model file en voeg de kolomnamen in de model toe van de tabel
8. Voeg in PHPMyAdmin jouw producten toe in de tabel.

> Weet je niet wat de commando's zijn voor het aanmaken van een migration of model? Check dan even Laravel Level 1: Migration en Models!

Wat lever je in?

- Een screenshot van jouw migration file
- Een screenshot van jouw models file
- Een screenshot van jouw gemigreerde tabel in PHPMyAdmin

Producten laten zien vanuit de database en de kracht van Eloquent

Nu je de database gemaakt hebt en producten erin hebt gezet, moet je nog de producten ophalen vanuit de database en tonen op het scherm.

Hoe haal je jouw producten op uit de database?

Zoals in eerdere blokken aangegeven, kan je communiceren met de database via Models. Om aan te geven welk model (dus welk tabel) data wil ophalen, moet je die aangeven in de Controller.

Data opvragen doe je in de Controller

Weet je nog wat de controller doet? Voor recap:
Controllers zorgt voor de logica en verbindt de **Model** met de **View**. In de uitleg van het **MVC model** was de **Controller** de ober. Die moet ervoor zorgen dat de bestellingen van de klant (de view) goed wordt doorgegeven aan de chef (model).

Maar hoe geef je dan aan dat je jouw producten wil via de `Product` model? Daarvoor hebben we **Eloquent**.

Wat is Eloquent?

Eloquent zorgt ervoor dat je **CRUD** acties aansturen met jouw **Models**. In dit geval voor jouw producten, kan je dus **Eloquent** gebruiken bij de **Product** model. Zo kan je nu bijvoorbeeld jouw producten ophalen (C**R**UD van **Read)**

Hoe haal je data op uit een tabel?

Om data op te halen uit een tabel moet je dus aansturen wat de Model moet doen. Alle aansturingen en logica doe je dus in jouw controller. Hieronder is er een voorbeeld hoe je auto's ophaalt:

Carcontroller.php
Stel dat dit een controller is die je hebt gemaakt:

```
namespace App\Http\Controllers;

class CarController extends Controller
  public function showCars(){

    return view('listcars');
  }
}
```

In deze functie wil je alle auto's ophalen, en de data in de view meenemen (daarom heet de functie ook showCars). Daarvoor moet je dus eerst aangeven welk Model je wil aansturen. Voor dit voorbeeld heb ik een `Car` model gemaakt. Om aan te geven welk Model je wil aansturen geef je eerst aan in de controller welk model je wil gebruiken met **use**:

`use App\Models\Car;`

Dit stukje code zet je tussen de namespace en class in:

```
namespace App\Http\Controllers;
use App\Models\Car;

class CarController extends Controller
  public function showCars(){

    return view('listcars');
  }
}
```

Vervolgens geef je in de functie aan met Eloquent wat je wil aansturen met de Car Model. In dit geval wil je dus ophalen met:

`$allCars = Cars::all();`

Dit stukje code zet je in de functie, voordat je de view laat zien zoals hieronder is weergegeven:

```
namespace App\Http\Controllers;
use App\Models\Car;

class CarController extends Controller
  public function showCars(){

  $allCars = Car::all();
```

```
    return view('listcars');
  }
}
```

Opgehaalde data meenemen in een view

Nu heb je alle auto's opgehaald met een model via Eloquent. Die heb je in de variabele `$allCars` gezet. Om de auto's op de view te laten zien, moet je de variabele meenemen waar je jouw view returnt. Dat is in dit geval `return view('listcars');`

De functie `view()` kan je twee parameters geven:

- Paramater 1: jouw view die je wil tonen.
- Parameter 2: data die je in de view wil meenemen (zoals in dit geval de auto's)

Voor parameter 2, kan je op verschillende manieren data (in dit geval de auto's) meenemen. Zo kan je bijvoorbeeld data meenemen hoe de variabelenaam (in dit geval $allCars) heet: `compact('allCars')`

Wat doet `compact()`?

Met compact() kan je een bestaande variabele met data in de controller meenemen naar de view en daarin je dezelfde variabele kan gebruiken.

Dit voeg je dus toe in de `return view()`:

```
namespace App\Http\Controllers;
use App\Models\Car;

class CarController extends Controller
  public function showCars(){

  $allCars = Car::all();
    return view('listcars', compact('allCars'));
  }
}
```

Je kan dus nu in jouw view `$allCars` gebruiken om de data op te halen en te gebruiken (door bijvoorbeeld te tonen)!

Meegenomen data in de view tonen

Met behulp van de compact() functie, kan je dus de variabele $allCars ook gebruiken in jouw view! Zoals je al eerder weet, kan je met Blade PHP code schrijven.

Stel: dit is de view van `listcars`:

```
@extends('reuse')
@section('content')
<div class="container">
      <h1> Mijn Auto's </h1>
      <div class="row">
      <div class="col-4">
          <h3>Porsche 911</h3>
          <p>Supervette auto!</p>
        </div>
      <div class="col-4">
          <h3>Volkswagen Polo</h3>
          <p>Goede prijs-kwaliteit auto!</p>
        </div>
      </div>
</div>
@endsection
```

Zoals je ziet zijn er twee auto's hard-coded gemaakt. Dit gaan we dus dynamisch maken door de data te laten zien uit de database met behulp van Blade Templates:

Deze code:

```
<div class="col-4">
  <h3>Porsche 911</h3>
  <p>Supervette auto!</p>
</div>
<div class="col-4">
  <h3>Volkswagen Polo</h3>
  <p>Goede prijs-kwaliteit auto!</p>
</div>
```

Zetten we nu om naar:

```
@foreach($allCars as $car)
<div class="col-4">
  <h3>{{$car->title}}</h3>
```

```
  <p>{{$car->description}}</p>
</div>
@endforeach()
```

Wat staat er in de code?

- `@foreach` / `@endforeach`: zorgt ervoor dat je jouw array (in dit geval `$allCars`) geloopt wordt. Elke auto kan je aanroepen met `$car`. De variabele kan je zelf noemen.
  - In de loop, loop je ook een `<div class="col-4">`. Dus er passen maximaal 3 kolommen met autogegevens per rij (er passen 3x col-4 per rij).
    - In de kolommen heb je de data die je toont van `$car`. Je kan een specifieke data tonen met behulp van de kolomnaam uit de Model zoals `$car->title` en `$car->description`

Opdracht: data ophalen met Eloquent en op een view laten zien (10p)

Om data van een model (in dit geval jouw **Product** model) op te halen moet je eerst jouw model in jouw controller toevoegen:

Stappenplan:

1. Open jouw `ProductController.php` file.
2. Om jouw model toe te voegen, voeg je deze code toe: `use App\Models\Modelnaam;`. Jouw *Modelnaam* is dus de naam van jouw Model en dat is `Product`
3. Zoek in jouw `ProductController` naar de functie die jouw `view` returnt naar de productpagina.
    1. Maak in de functie code om gegevens op te halen via Product Model met Eloquent en stop die in een variabele.
    2. Maak gebruik met compact() om data mee te nemen in de view
4. Ga in de aangewezen view een toon daar jouw gegevens uit de database met behulp van Blade Templates

Wat lever je in?

- Jouw ProductController.php file
- Jouw Productpagina view