Laravel 2022 - L1

- Inleiding
- Introductie
- Installatie
- Views
- Routes
- Controllers
- Migrations
- Models
- Refacturing
- Blade template
- Tot slot

Inleiding

Laravel

Laravel is een PHP web applicatie framework die ervoor zorgt dat je makkelijk applicaties kan opbouwen met een versimpelde syntax in PHP.

Zo kan je bijvoorbeeld in Laravel gemakkelijk een webshop bouwen of snel een site in elkaar zetten. Omdat Laravel zo groot en uitgebreid is, hebben we het verdeeld in drie Levels:

Level 1:
- Installatie
- Views
- Routes
- Het MVC model
- Controllers
- Migrations en database
- Models

Level 2:
- Producten tonen
- Bootstrap
- Pagina's opmaken met Bootstrap
- Eloquent
- Productpagina vanuit de database tonen

Level 3:
- Registreren, Login en Sessions
- (REST) API's
- Testen met Postman
- Betaalsysteem van Mollie
- Emails versturen na een betaling

Tijdens het volgen van deze module bouw je in kleine stappen een simpele webshop in Laravel. Bedenk voor jezelf wat voor producten je wilt verkopen in jouw webshop! Wellicht kan je een e-commerce starten? :)

Introductie

Welkom bij Laravel!

Laravel is een webapplicatie framework met een eigen syntax. Een web framework biedt een structuur en startpunt voor het creëren van jouw applicatie, zodat je kan focussen op het creëren en niet aan andere zaken zoals:

- Queries schrijven in SQL
- Dingen vanaf scratch bouwen zoals:
  - Loginsystemen
  - E-mails
  - Rollen
  - CRUD
- Ingewikkelde PHP-code in HTML

Laravel streeft ernaar om een ontwikkelaar een geweldige ervaring te bieden terwijl het krachtige functies biedt. Of je nu nieuw bent met PHP-web frameworks of al jaren ervaring hebt, Laravel is een framework dat met je mee kan groeien. Laravel helpt je bij je eerste stappen als web ontwikkelaar of geven je een duwtje in de rug als je je expertise naar het volgende niveau wilt tillen.

Waarom Laravel?

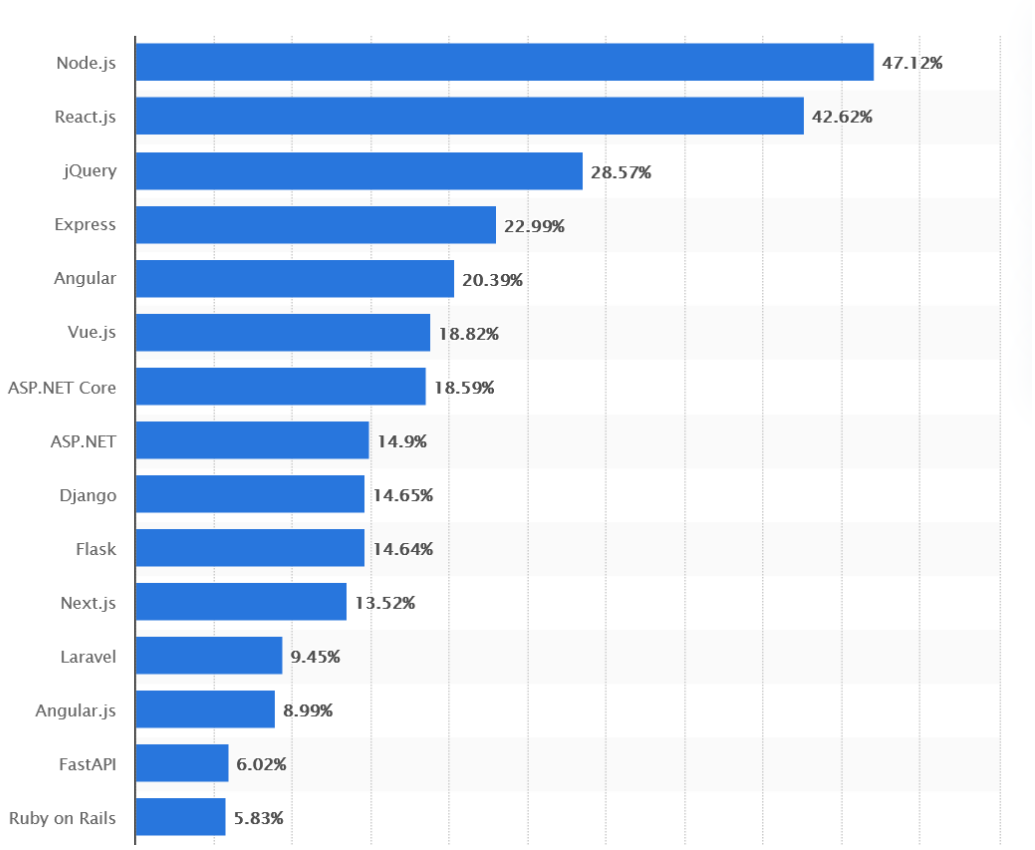
Er zijn verschillende tools en frameworks beschikbaar voor het bouwen van een webapplicatie. Laravel syntax is completer dan bijvoorbeeld Yii/Symfony. Daarnaast is Laravel de afgelopen jaren veel gegroeid als framework en wordt het veel gebruikt op het internet.

> Weetje: Symfony zit zelf in Laravel, dus dan weet je dat je meer krijgt uit het Laravel framework!

Hoeveel wordt Laravel gebruikt?

Alle overzichten die je vindt op het internet zijn anders omdat ze anders worden gemeten. Tegenwoordig wordt er veel in React (met Node.js en jQuery) ontwikkeld. Kijk je op het huidige internet dan is de install base (dat is het aantal bestaande web pagina's) in PHP heel groot en

waarschijnlijk is PHP op dit moment op het web het meest gebruikt. Dat komt omdat veel web applicaties zijn gebouwd met WordPress (=PHP). Van alle PHP ontwikkel-frameworks is Laravel het meest gebruikte framework. WordPress tel ik hierbij niet mee, dat is een CMS-framework.



Volgens https://www.statista.com heeft Laravel een marktaandeel van ruim 9%.

Laravel documentatie

De officiële documentatie van Laravel is heel duidelijk en is het handig om bij het ontwikkelen van een website altijd de documentatie erbij te pakken.

Maar je kunt heel veel tutorials, filmpjes, voorbeelden en uitleg vinden op het internet over Laravel. Dat is één van de redenen waarom het zo populair is.

--

Installatie

Laravel installeren

Voordat je aan Laravel kunt beginnen dien je je ontwikkelomgeving uit te breiden met Composer. Composer is de installer voor PHP.

- Composer → https://getcomposer.org/doc/00-intro.md#installation-windows

Let op: heb je Yii gedaan dan heb je waarschijnlijk Composer al geïnstalleerd!

Een Laravel project aanmaken

Er zijn 2 manieren om een Laravel project aan te maken. Via Composer of de Laravel installer.

1. Composer

```
laravel new example-app
```

2 Laravel Installer

Om dit te kunnen gebruiken run je eerst het volgende command:

```
composer global require laravel/installer
```

Je hebt zojuist de Laravel installer geïnstalleerd, deze kun je nu als volgt gebruiken om een Laravel project aan te maken:

```
laravel new example-app
```

Opdracht: project aanmaken

Met Laravel zal je iets meer met de command line moeten werken (terminal). Hiermee kun je bijvoorbeeld Controllers, Models en meer aanmaken. Standaard commands runnen via de command line is ook mogelijk. Het eerste command wat we moeten runnen om met Laravel te kunnen beginnen is het volgende:

```
composer create-project laravel/laravel webshop
```

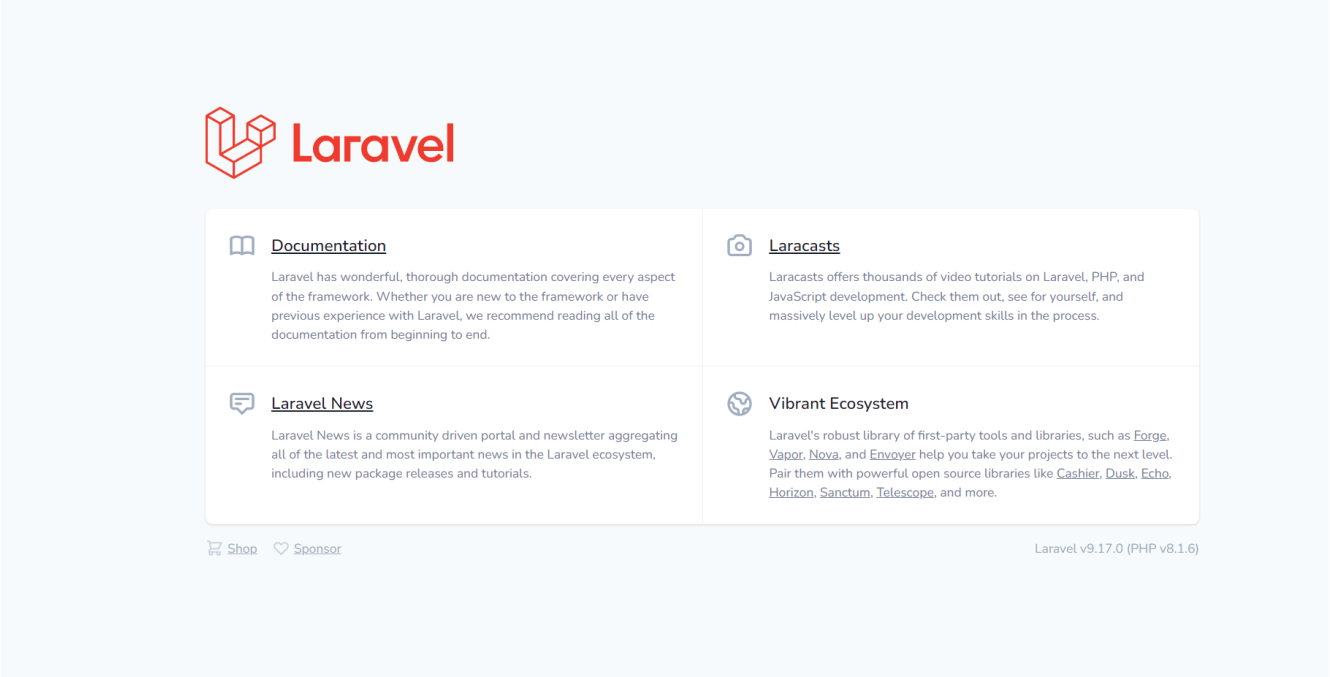
Er is nu een nieuwe map aangemaakt, **webshop**, hierin kom je veel mappen en bestanden tegen.

Open nu de map **webshop** in Visual Studio Code. Om vervolgens je project te kunnen bewonderen moeten we het project starten.
Ga naar tab Terminal van VSCode en open een nieuwe Terminal.
Run het volgende command:

```
php artisan serve
```

> Herken je dit nog van Yii, daar was het commando `php yii serve`. Onthoud dit commando, je gebruikt dit iedere keer om jouw Laravel project op te starten!

Er is nu een standaard webpagina beschikbaar, gemaakt door Laravel. Deze kunnen we aanpassen naar onze wensen.



Open bovenstaande pagina in de browser met http://localhost:8000Links to an external site.

Gelukt? Dan is Laravel goed geïnstalleerd!

Inleveren

1. Een screenshot van jouw Laravel startpagina, zorg dat het hele scherm zichtbaar is.

--

Views

Wat is een View?

In een framework heb je een plek nodig om al je data visueel te maken voor de gebruiker. Hiervoor gebruiken we Views.

Een View is een plek waarin je vooral veel HTML zal vinden (zelfde als bij Yii). In de View kun je verschillende logica's gebruiken zoals:

- Data tonen (met PHP)
- CSS en Javascript files toevoegen en gebruiken

In Laravel staan alle views in de `resources/views` map. Zoals je ziet is er een standaard view die `welcome.blade.php` heet. Als je daarin kijkt, zie je HTML code.

Vieuws in Laravel eindigen in `.blade.php` Dat komt omdat de views "blade templates" zijn. Dat wordt later meer over uitgelegd.

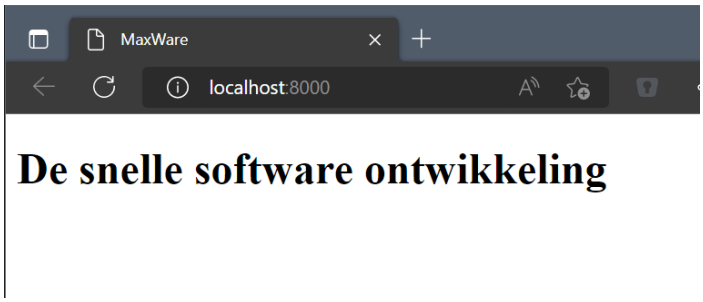
Opdracht: welcome.blade.php aanpassen

Je gaat nu het bestand leegmaken en de code aan jouw eigen voorkeur aanpassen.

Stappenplan

1. Open de `welcome.blade.php` file
2. Maak het HTML bestand leeg
3. Plaats de standaardHTML-structuur (! en dan tab)
4. Bedenk je eigen bedrijfsnaam en vul dit in als titel
5. Bedenk een eigen slogan om jouw webshop te promoten en plaats dit als H1 in je website.

   Dit zou je output bijvoorbeeld kunnen zijn:



Inleveren

1. Een screenshot van de code van jouw .blade.php bestand
2. Een screenshot van jouw browser

Routes

Inleiding

Je hebt eerder al een View aangepast, een nieuwe View aanmaken doe je door in de `/resources/views/` map een file te maken die eindigt op `.blade.php`.

Hoe kan deze nieuwe View getoond worden? Daarvoor hebben we dus R**outes**.

- Open het bestand `/routes/web.php`

Je ziet nu één standaard Route die verwijst naar de homepagina (welcome.blade.php):

```
Route::get('/', function () {
    return view('welcome');
});
```

Wat zijn Routes?

Routes zijn links die verwijzen naar jouw webpagina's.

HTTP kent verschillende methoden om gegevens op te vragen. Zo hebben we bij formulieren gezien dat je de methode GET en POST kan gebruiken.

Weet je nog hoe we forms maakte en hoe we de methode GET bij een form gebruikte?
`<form action="action.php" method="GET">`

GET is de meest gebruikte methode. Als je 'gewoon' een URL via je browser opvraagt dan gebruik je altijd (vanzelf) de methode GET.

We zullen later in deze Laravel module nog meer methodes bespreken.

Voor nu hebben we alleen GET nodig om onze eigen blade view te bereiken.

web.php

Zoals hierboven vermeld, staat er al standaardcode in `web.php`.

In web.php staat:

```
Route::get('/', function () {
    return view('welcome');
});
```

Wat staat er in deze code?

- Je begint met een **class** Route, hiervan gebruik je de methode GET. In deze functie geef je 2 **arguments** mee (zie Laravel documentatieLinks to an external site.):
  - DE URL van je website: Dat is in dit geval `/` (de root oftewel de homepagina)
  - De action (functie): Dat is in dit geval `return view('welcome');` (laat de View welcome.blade.php zien)

In Yii hebben we automatische routing gebruikt en ging je altijd via de controller naar de view. In Laravel kun je op deze manier ook rechtstreeks naar de view gaan en de controller dus overslaan. Voor statische pagina's waar geen gegevens uit de database wporden getoodn is dat natuurlijk prima.

Standaard zoekt de Route automatisch in /resources/views/ naar een `.blade.php` bestand. Je maakt dus een route naar welcome en dat wordt vertaald naar `/resources/views/welcome.blade.php`

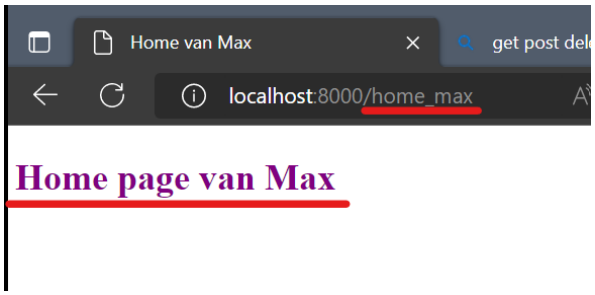
Opdracht : maak een nieuwe View en Route

Bij deze opdracht maken we een nieuwe view en voegen we het toe aan een route.

1. Maak een nieuwe View aan en noem het als volgt: `home_jouwvoornaam.blade.php` (stel je heet Max dan wordt het `home_max.blade.php`)
2. Schrijf wat HTML code in jouw home_jouwvoornaam.blade.php. Gebruik juiste HTML code.

3. Ga naar de `web.php` file en maak een nieuwe GET route
   - Verwijs het naar `/home_jouwvoornaam`
   - Return de View naar je zojuist gemaakte view (`home_jouwvoornaam.blade.php`)

   Voorbeeld:



Inleveren

1. home_jouwvoornaam.blade.php
2. web.php
3. Screenshot van de browser die jouw site toont met URL `/home_jouwvoornaam (zie voorbeeld hierboven).`

--

Controllers

Inleiding

Je hebt een nieuwe View en Route gemaakt.

De view laat statische data zien (= niet uit de database). Als we data uit de database willen tonen (=dynamische data) dan hebben we net als in Yii de **controller** nodig.

Dit is een lang verhaal waarin MVC en het nut van een controller wordt uitgelegd. We hebben dit ook gehad bij Yii, maar het is belangrijk dat je precies weet wat een controller en een view doet. Lees dit stuk dus goed door.

Wat is een Controller?

Een Controller zorgt ervoor dat de V**iew** requests kan ontvangen en/of versturen naar de **database**. Deze structuur noemen we ook wel een ModelViewController (**MVC**).

Wat is een MVC?

De MVC is een basis software architectuur waardoor je makkelijk en overzichtelijk projecten kunt bouwen. Het MVC model maakt elke code los om een specifieke taak uit te voeren. Daardoor is bekend wat elk onderdeel doet en bij bugs is het makkelijk te zien waar het probleem zit.

> De MVC architectuur wordt ook gebruikt in andere frameworks.

Wat doet een MVC?

De acties bij een MVC kun je vergelijken met een restaurant.

Scenario: Het restaurant (MVC):
Het restaurant is de **MVC** architectuur. Zij zorgen er allemaal voor dat het restaurant (in webdevelopment, de site) soepeltjes verloopt en dat alle bestellingen worden uitgevoerd.

image-1656834908474.jpg

De klant (View)

De klant is de **View**. Zij zijn technisch gezien de front-end en zijn verantwoordelijk wat jij op jouw browser te zien krijgt.
image-1656831284319.jpg
Dit zijn de klanten van het restaurant. Zij willen graag `pizza` en `pasta` bestellen. Zij voeren een `GET` request uit:

```
Request - GET: items: {'pizza', 'pasta'}
```

> Onthouden! Alle **Views** staan in het mapje `/resources/views/` met als filenaam *`filenaam`*`.blade.php`.

De ober (Controller)

De ober is de **Controller**. Zij zijn verantwoordelijk voor het communiceren tussen de **View** (klant) en **Model** (keuken). Je kan ze zien als de tussenpersoon.
JEMA GER 1459-03-min.jpg
Dit is de ober. De ober heeft de bestelling (`GET` request) ontvangen en zorgt ervoor dat dit gecommuniceerd wordt naar de **keuken**.
Hieronder zie je hoe een Controller te werk gaat in Laravel:

```
//De class Controller is de ober
class OberController extends Controller {


    //Een ober heeft als functie om gerechten op te halen van de Keuken
    public function getDishes(Request $request){


        //De GET request (pizza & pasta) is wat de Controller (ober) gekregen heeft van de
View (klant)
          //(zie dit hetzelfde als een $_GET)
        $bestellingen = $request->query('items');


          //De Controller (ober) vraagt aan de Model (keuken) om de gerechten te maken en
het uit te voeren
          $gerechten = Keuken::maakGerechten($bestellingen)->get();


          //De Controller (ober) geeft het gerecht aan de klant (View)
          return view('klant', compact('gerechten'));
    }
}
```

De keuken (Model)

De keuken is de **Model**. Zij zijn verantwoordelijk dat ze de taken van de **controller** uitvoeren en dit te communiceren met de **database**.
image-1656833917750.jpg
Dit is de keuken. Ze hebben een taak van de ober ontvangen om de gerechten te maken en te geven aan de ober..

Hoe maak je een Controller?

Nu je weet wat een controller doet, gaan we er eentje maken. Dat gaat heel simpel!
Een controller maken kan je makkelijk met een commando via de terminal maken:

- `php artisan make:controller JouwvoornaamController`
  Let op dat Jouwvoornaam met ene hoofdletter begint!

Voer het commando uit. Jouw Controller is nu aangemaakt! Je kan jouw gemaakte Controller terugzien in de map `app/Http/Controllers` met de naam `JouwvoornaamController.php`

Als je de file opent, zie je deze code:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class JouwvoornaamController extends Controller
{
    // Hier maak je jouw functions.
}
```

Dit is jouw Controller. Alleen is het leeg. In de class maak je alle logica en CRUD's aan.

Jouw View tonen vanuit de Controller

We gaan nu code maken die de view die we eerder hebben gemaakt aanroept vanuit de controller.

Om dat te kunnen doen moet je eerst een `public function functieNaam(){}` in de `class` maken. Daarin plaats je `return view('viewNaam');` in de codeblok van `public function functieNaam(){}`. Hieronder een voorbeeld:

```
public function functieNaam(){
  return view('viewNaam');
}
```

Gefeliciteerd! Je hebt nu een Controller gemaakt die een view laat zien! Alleen laat hij dat niet zien, omdat hij niet weet naar welk URL hij moet gaan...

Daarom moet je hem nog via de **Routes** naar de Controller verwijzen.

Jouw Controller verwijzen in de Routes

Om jouw Controller te verwijzen in de Routes moet je `routes/web.php` aanpassen.

Je maakt een route naar de controller op de volgende manier:

```
// plaats dit bovenaan in de routing file (web.php)
use App\Http\Controllers\UrlController;

// Plaats vervolgens je routing informatie
Route::get('/url', [UrlController::class, 'functionName']);
```

/url: dit is de URL die je koppelt aan de controller.

UrlController: de naam van de Controller

functionName: de naam van de van de functie die Controller zit

Een voorbeeld van de Routes bij het restaurant scenario (op basis van de code bij "*De ober (Controller)*":

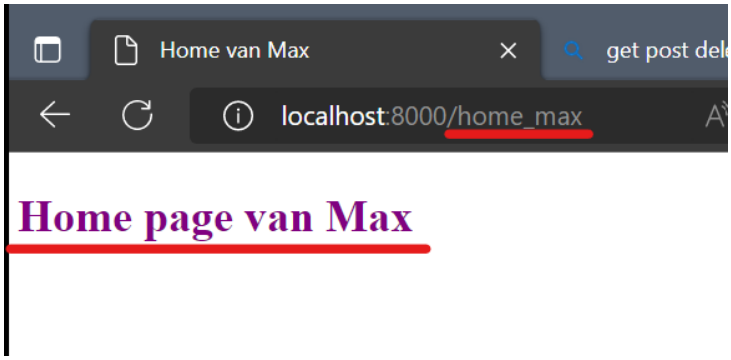
`Route::get('/dishes', '[Ober*Controller::class*, 'getDishes');`

Dit zorgt ervoor dat de URL */dishes* wordt gekoppeld aan de functie *getDishes* die staat in het bestand *App\Http\Controllers\OberController*

Opdracht - Controller maken en linken aan de Route

1. Maak een controller aan via de terminal en geef de ControllerNaam `homeController`
2. Ga naar de controller en maak in de `class` een `public function` aan die `index()` heet. Toon jouw eerder gemaakte View in de Controller.
3. Ga naar jouw `web.php` en zoek naar de `Route::get` die de URL `/` heeft
   Vervang de verwijzing i.p.v. naar de view nu naar de `homecontroller`. Ook moet hij verwijzen naar de `index` functie.

   De output is hetzelfde als in de vorige opdracht, maar dit keer ga je via de ***controller*** naar de view.



Inleveren

1. Een screenshot van de terminal die aantoont dat je een Controller hebt gemaakt
2. Een screenshot van de code van je Controller
3. Een screenshot van de code van je Routes

Migrations

Inleiding

Migrations zijn als versiebeheer voor je database.

Je beschrijft met scripts hoe je database eruit moet komen te zien. Als je dan later je database verandert dan kun je met dezelfde scripts de bestaande databases aanpassen (=migreren). Dat maakt het makkelijk als je code en database aan wilt passen in de productieomgeving. In plaats van in phpmyadmin tabel voor tabel aant e passen draai je gewoon een script.

Migrations heb je ook in Yii, die hebben we alleen niet behandeld.

Database config

Laten we eerst een database maken. We maken via localhost/phpmyadmin een nieuwe database aan en noemen dat *webshop*.

In de root in ons Laravel project openen we dan de `.env` file en we zorgen ervoor dat er in de file het volgende komt te staan.

```
...
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=webshop
DB_USERNAME=root
DB_PASSWORD=
...
```

Migrations

In de terminal in Visual Studio Code voer je het volgende commando uit:

```
php artisan make:migration create_product_table --create=product
```

Dit commando maakt een file. Deze staat in `database/migrations/` en heet

`{{datetime}}_create_product_table.php`

Open dit bestand en plaats de volgende code:

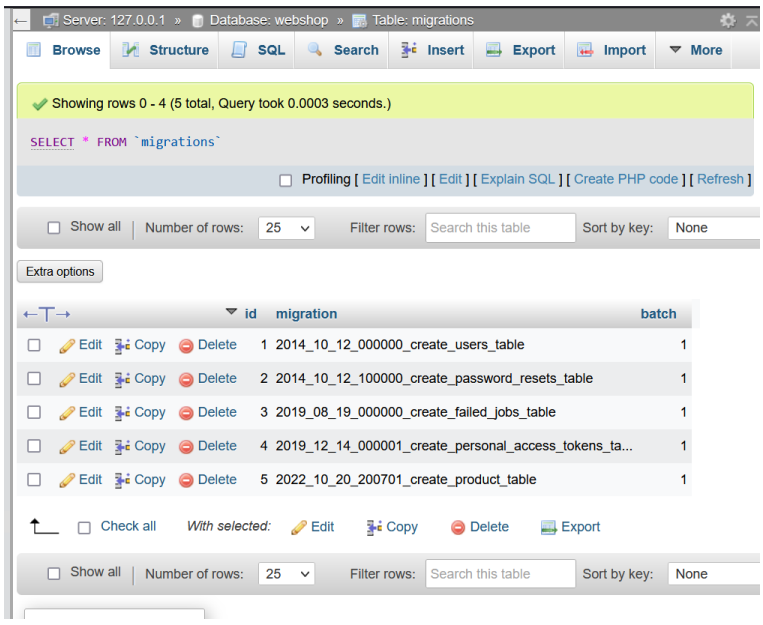
```
public function up()
{
    Schema::create('product', function (Blueprint $table) {
        $table->increments('id');
        $table->string('name');
        $table->text('description');
        $table->decimal('amount', 8, 2);
        $table->timestamps();
    });
}
```

In dit script wordt in 'Laravel-taal' beschreven hoe de tabel *product* moet worden gemaakt.

In de terminal kan je nu met het volgende commande de tabel maken.

```
php artisan migrate
```

Ga naar localhost/phpmyadmin en controleer of de tabel aangemaakt is.



Naast de tabel product worden er nog meer tabellen aangemaakt. Dit zijn tabellen die Laravel standaard aanmaakt.

Als je nog een keer `php artisan migrate` uitvoert dan gebeurt er niets meer. Dat komt omdat de migratie al is uitgevoerd. Laravel houd bij (in de database tabel *migations*) welke migraties zijn

uitgevoerd.

Wil je een migratie toch nog een keer willen uitvoeren dan kan dat met het commando.

```
php artisan migrate:refresh
```

Inleveren

1. Een schermafdruk van je browser met phpmayadmin waarin te zien is dat de tabel is aangemaakt (zoals in het voorbeeld).
2. Het bestand `2022_10_20_200701_create_product_table.php` (in database/migrations).

--

Models

Model

Een model in de communicatie tussen je Database en je Controller. Een model is verantwoordelijk voor het uitvoeren van de taken die de Controller vraagt. Bijvoorbeeld:

- Ik wil graag **alle** rijen ontvangen uit het **users** tabel
- Ik wil graag een nieuwe rij aanmaken in het **users** tabel

Kort gezegd is je Model dus de laag tussen je Controller en de Database.

Hoe maak ik een model?

Bij het installeren van Laravel is er al een Model aanwezig, dit is het User model. Deze wordt standaard gebruikt om gebruikers te beheren in Laravel. Naast het standaard Model kunnen we ook een eigen model aanmaken om dit later te kunnen gebruiken. Dit doe je als volgt:

```
php artisan make:model Product
```

Nadat je een model hebt aangemaakt kun je dit vinden in de map `app/Models`.

In het model hoeven we alleen maar de tabelnaam te specificeren.

```
class Product extends Model


{
    protected $table = 'product';
    use HasFactory;
}
```

Regel 4 uit dit voorbeeld voegen we toe aan het model `app/Models/Product.php` en we maken een link naar de tabel.

In web.php zetten we de volgende route.

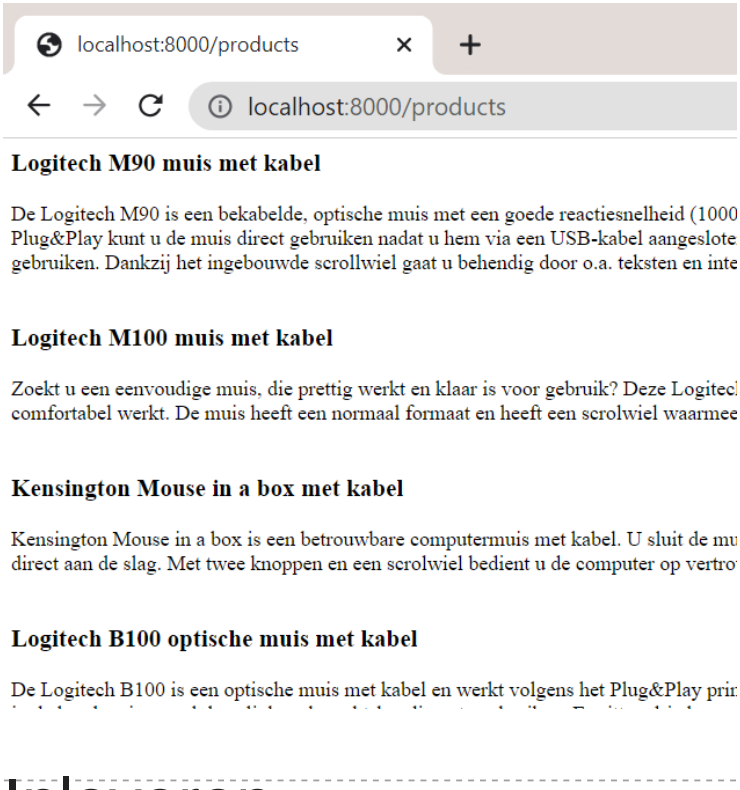
```
Route::get('/products', function () {
$products = \App\Models\Product::all();
```

```
    foreach($products as $product) {
        echo "<h3>";
        echo $product['name'];
        echo "</h3>";
        echo $product['description'];
        echo "<br><br>";
    }
});
```

Dit is een route `/products` De route verwijst niet naar een Controller of naar een View. Dat doen we in de volgende stap. We zetten nu tijdelijk alle code in de route zelf.

In de code worden alle *producten* ingelezen uit de database. Daarna worden in een loop van elk product de *name* en de *description* afgedrukt.

Als we nu deze nieuwe route uitproberen dan zien we het bijvoorbeeld het volgende (in dit voorbeeld is de database gevuld met computermuizen).



Inleveren

1. Een schermafdruk van de browser waarin je de URL kan zien en waarin je jouw producten laat zien (zoals bijvoorbeeld in het voorbeeld hierboven).
2. Jouw `web.php`

--

Refacturing

Inleiding

We hebben een hele eenvoudige Read gemaakt van de producten.

We gaan nu een nette output maken, maar voordat we dat gaan doen gaan we eerst onze code netjes opdelen in controller en view (=refacturing). Nu hebben we (om te testen) even een hele eenvoudige output in de routing file gezet. Leuk om te testen, maar als we alle code zo zouden maken dan krijgen we een hele grote overzichtelijke web.php

We gaan dus eerst de code die we nu hebben op de juiste plaats zetten. Daarna gaan we onze output fraaier maken.

Controller

We maken eerst een controller aan voor product.

```
php artisan make:controller ProductController
```

In de controller zetten we de volgende functie.

```
public function read() {
    $products = \App\Models\Product::all();
    return view('product-read', ['products' => $products]);
}
```

De functie verwijst naar de `product-view` en geeft de variabele `$products` mee.

View

We maken in de views folder een nieuw bestand `product-read.blade.php`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
</head>
<body>
    <?php dd($products); ?>
</body>
</html>
```

Met de dd() functie zien we de inhoud van $products. Dat ziet er ongeveer zo uit. Maar voordat je dat ziet zal je eerst de route moeten aanpassen.

```
Illuminate\Database\Eloquent\Collection {#303 ▼ // resources\views/product-read.blade.php
  #items: array:10 [▼
    0 => App\Models\Product {#305 ▼
      #connection: "mysql"
      #table: "product"
      #primaryKey: "id"
      #keyType: "int"
      +incrementing: true
      #with: []
      #withCount: []
      +preventsLazyLoading: false
      #perPage: 15
      +exists: true
      +wasRecentlyCreated: false
      #escapeWhenCastingToString: false
      #attributes: array:6 [▼
        "id" => 1
        "name" => "Logitech M90 muis met kabel"
        "description" => "De Logitech M90 is een bekabelde, optische muis met een goede
reactiesnelheid (1000 dpi). Door de gebruiksvriendelijke, symmetrische vorm van de muis
bedient u  ▶"
        "amount" => "7.50"
        "created_at" => null
        "updated_at" => null
      ]
    ......
........
```

Route

In web.php zet je de volgende route

```
// plaats dit bonveaan in de routing file (web.php)
use App/Http/Controllers/ProductController

//plaats dit bij de andere routes in de routing file
Route::get('/products', [ProductController::class, 'read']);
```

Hiermee koppel je de url */products* aan de controller. En de controller gaat naar de view en de view laat de dd() zien.

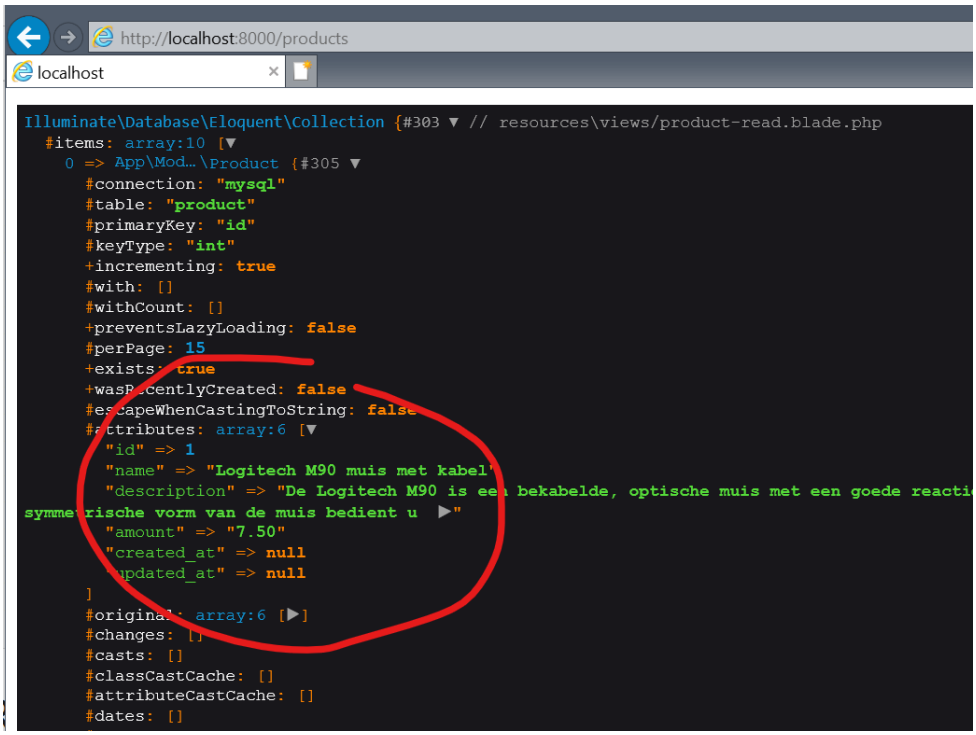
Zorg er voor dat de oude route naar /products verwijderd wordt want je kunt geen twee dezelfde routes hebben want dan weet Laravel niet waar je heen wil.

In de volgende les gaan we de view goed maken.

Inleveren

1. Screendump van je browser waarin je de dd() laat zien. Zoals in het voorbeeld hierboven bij de view.
   Zorg dat jouw eigen data uit jouw database te zien is.

   Voorbeeld:



--

Blade template

Inleiding

Een Blade template is een HTML plus extra codes om bijvoorbeeld gegevens uit de database af te drukken.

Je kunt Blade dus zien als een soort uitbreiding op HTML.

In **Yii** gebruikte je PHP code in je HTML, bijvoorbeeld

```
<?php echo $voorNaam ?>
```

bij een **Blade** template gaat dat anders:

```
{!! $voorNaam !!}
```

Voorbeelden

Een paar veel voorkomende voorbeelden van Blade code zijn:

if-then-else

```
@if (count($records) === 1)
    I have one record!
@else
    I don't have any records!
@endif
```

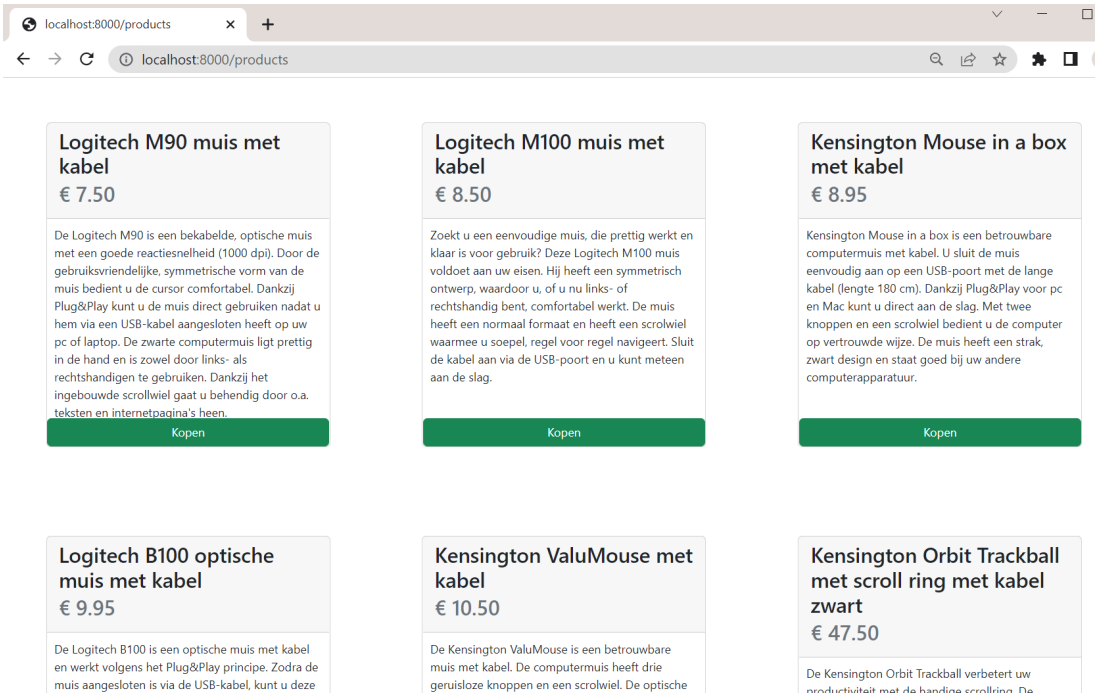
foreach

```
@foreach ($products as $product)
 <h3>{{ $product->name }}</h3>
    <p>{{ $product->description }}</p>
@endforeach
```

Voor een compleet overzicht ga naar: https://laravel.com/docs/9.x/blade

Opdracht

We gaan ons productoverzicht maken. Het moet er ongeveer zo uit komen te zien.



In jouw overzicht moeten je eigen producten staan.

OK hoe gaan we dit doen?

We openen de view `product-read.blade.php` en zetten plaatsten standaard HTML code in de pagina. In de body komt de volgende code:

```
@foreach ($products as $product)
        <p>{{ $product->name }}</p>
        <p>{{ $product->amount }}</p>
        <p>{{ $product->description }}</p>
@endforeach
```

Dit is de basis. Test of het werkt!

Nu ga je met CSS de stylen maken. Dat kan met <style></style> in de header, **bijvoorbeeld**:

```
<style>
    .myFlex {
        display: inline-flex;
        flex-direction: row;
        flex-wrap: wrap;
        margin: 20px;
    }
    .myItem {
```

```
        flex: 0 0 auto;
        margin: 40px;
      }
    ...
    ...

    </style>
```

Je zou ook gebruik kunnen maken van Bootstrap, zet daarvoor in de head de volgende regel:

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

Wat zijn de minimale eisen?

1. Je laat al jouw (10) producten op één pagina zien. De productnaam, omschrijving en prijs komen uit de database.
2. De producten staan per product in een box, waarin de titel, de prijs en de omschrijving te zien is.
3. Voor de prijs staat het Euro teken.
4. Bij elk product staat een knop kopen, deze knop doet (nog) niets.

Inleveren

1. Screendump van je browser met daarin een overzicht van jouw producten. Voorbeeld staat hierboven.
2. Jouw product-read.blade.php

--

Tot slot

Je hebt nu de basis van Laravel gehad en dat is best veel:

- Je weet hoe je Laravel moet **installeren**.
- Je weet waar de **database configuratie** staat (`.env`) en je weet wat een **database migration** is.
- Je weet hoe de basis van de **routing** werkt in Laravel.
- Je weet dat Laravel net als Yii met **controllers**, **models** en **views** werkt. Dit is de **MVC** architectuur.
- We hebben een eenvoudige **controller** gemaakt.
- We hebben de Read van CRUD in met een Laravel **Blade template** gemaakt.

In de volgende module gaan we eens verder kijken naar het CSS frame work Bootstrap en hoe we dat in Laravel kunnen gebruiken.

--