|  |  |
| --- | --- |
| Auteur(s): |  |
| Ron van Asseldonk |  |

|  |
| --- |
|  |
| WEB DEVELOPMENT |
| Introductie Laravel |



Colofon

De reader Web Development dient voor de ICT opleidingen van het Summa College te Eindhoven.

© Summa College, ICT, 2019

Datum: 25 augustus 2019

Auteur(s): Ron van Asseldonk

Redactie: Jeroen van Amsterdam

Update: 28 september 2020 naar Laravel versie 8.

Summa ICT

Sterrenlaan 10

5631 KA te Eindhoven

[www.summa-ict.nl](http://www.summa-ict.nl)

Inhoudsopgave

[1. Leeswijzer 5](#_Toc52208565)

[2. Inleiding 7](#_Toc52208566)

[2.1 Wat is Laravel? 7](#_Toc52208567)

[2.2 Waarom Laravel? 7](#_Toc52208568)

[3. Algemene tips webdevelopment 8](#_Toc52208569)

[3.1 Handige sites 8](#_Toc52208570)

[3.2 Versiebeheer 8](#_Toc52208571)

[3.2.1 Public v.s. private 8](#_Toc52208572)

[4. Installatie 9](#_Toc52208573)

[4.1 Configuratie 10](#_Toc52208574)

[5. Een Laravel project 11](#_Toc52208575)

[5.1 ENV file 11](#_Toc52208576)

[5.2 Composer.json en package.json bestanden 11](#_Toc52208577)

[5.3 Routes folder 12](#_Toc52208578)

[5.4 MVC in Laravel 13](#_Toc52208579)

[6. Een project in de praktijk 14](#_Toc52208580)

[6.1 Routing 14](#_Toc52208581)

[6.2 In combinatie met de database 16](#_Toc52208582)

[6.3 Het MVC principe 18](#_Toc52208583)

[6.4 Zelfstandige opdracht 21](#_Toc52208584)

[7. Views 22](#_Toc52208585)

[7.1 Blade basics 22](#_Toc52208586)

[7.2 Data doorgeven aan views 23](#_Toc52208587)

[7.3 Switchen van views 23](#_Toc52208588)

[7.4 Zelfstandige opdracht 24](#_Toc52208589)

[8. Forms 25](#_Toc52208590)

[8.1 Zelfstandige opdrachten 26](#_Toc52208591)

[9. Form request validation 27](#_Toc52208592)

[9.1 Error messages 28](#_Toc52208593)

[9.2 Zelfstandige opdracht 28](#_Toc52208594)

[10. CRUD met Eloquent 29](#_Toc52208595)

[10.1 Create 30](#_Toc52208596)

[10.2 Read 31](#_Toc52208597)

[10.2.1 Collections 31](#_Toc52208598)

[10.3 Update 32](#_Toc52208599)

[10.4 Delete 32](#_Toc52208600)

[10.4.1 Softdeleting 32](#_Toc52208601)

[10.5 Timestamps 33](#_Toc52208602)

[10.6 Zelfstandige opdracht 33](#_Toc52208603)

[10.7 Zelfstandige opdracht 34](#_Toc52208604)

# Leeswijzer

Deze reader is opgebouwd uit de volgende onderdelen:

**Hoofdstuk 1:** Leeswijzer

**Hoofdstuk 2:** Introductie Laravel (wat is Laravel en waarom wordt dit gegeven)

**Hoofdstuk 3:** Algemene tips webdevelopment

**Hoofdstuk 4:** Installatie en configuratie Laravel

**Hoofdstuk 5:** Een Laravel project (wat houdt een kale installatie van Laravel in)

**Hoofdstuk 6:** Een project in de praktijk (samen een Laravel project maken)

**Hoofdstuk 7:** Het gebruik van meerdere views.

**Hoofdstuk 8:** Forms

**Hoofdstuk 9:** Form request validation en error handling

**Hoofdstuk 10:** Het maken van CRUD’s met Eloquent

De student wordt verondersteld bekend te zijn met HTML, CSS, PHP, Javascript, MySQL.

# Inleiding

Ondertussen hebben jullie in leerjaar 1 kennis gemaakt met verschillende talen, zoals: HTML, CSS, PHP en Javascript. Nu gaan jullie werken met laravel.

## Wat is Laravel?

Laravel is een PHP framework dat ontwikkeld is door Tylor Otwell in 2011. Het is bedoeld voor het ontwikkelen van webapplicaties via het MVC principe. Laravel is tegenwoordig een van de meest populaire PHP frameworks.

## Waarom Laravel?

Er is gekozen om jullie ook Laravel te leren, aangezien het bedrijfsleven staat te springen om Laravel programmeurs. Daarbij is het ook zo dat Laravel niet alleen heel populair is en dus veel gebruikt wordt, maar ook sterk ontwikkeld (<https://laravel.com/docs/8.x>).

Verder is er afgelopen jaar ook de mogelijkheid gekomen om een Laravel certificering te halen (<https://certification.laravel.com/>).

Mede om deze redenen lijkt het ons een goede skill om op een ICT’er zijn/haar cv te hebben staan.

# Algemene tips webdevelopment

In dit hoofdstuk worden wat algemene tips gegeven voor webdevelopment. Zo worden er handige sites gegeven voor het werken met Laravel en worden er een aantal editors voorgesteld.

## Handige sites

Laravel is een van de meest populaire PHP frameworks, mede omdat Laravel een grote community heeft. Hierdoor is er veel documentatie te vinden en zijn er ook handige sites.

|  |  |
| --- | --- |
| Site | Doel |
| <https://laravel.com/docs/8.x> | Alle documentatie is hier te vinden over hoe de kern van Laravel werkt. |
| <https://laracasts.com> | Deze site is opgezet door Jeffrey Way en hier vind je veel tutorials voor Laravel in video vorm. |

## Versiebeheer

Versiebeheer van webapplicaties is heel belangrijk. Hierbij wordt vaak Git gebruikt. Nu zijn er meerdere services die versiebeheer aanbieden via Git. Kies voor het versiebeheer dat het beste bij jou past. Denk hier bijvoorbeeld aan:

* Github
* Bitbucket
* Gitlab
* Azure DevOps

### Public v.s. private

Wil je je code delen met de wereld dan zet je hem op public (Open Source). Bedrijven gebruiken vaak de private setting.

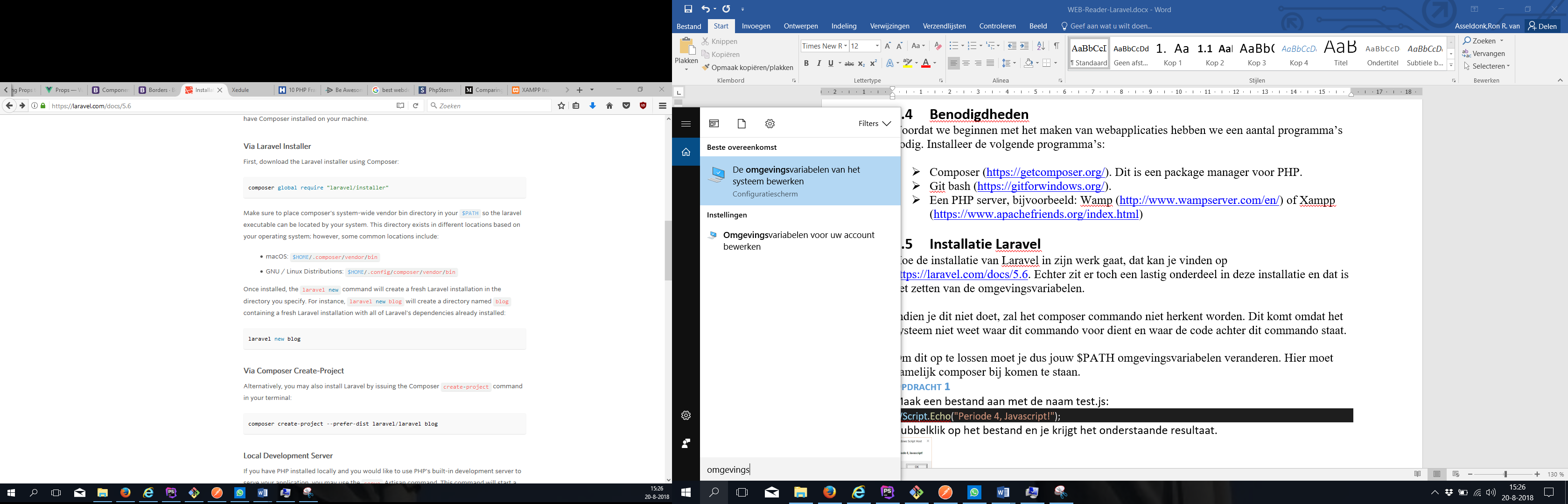
# Installatie

Voordat we beginnen met het maken van webapplicaties hebben we een aantal programma’s nodig. Installeer de volgende programma’s:

* Composer (<https://getcomposer.org/>). Dit is een package manager voor PHP.
* Git (<https://gitforwindows.org/>).
* Een PHP server, bijvoorbeeld: Wamp (<http://www.wampserver.com/en/>) of Xampp (<https://www.apachefriends.org/index.html>)

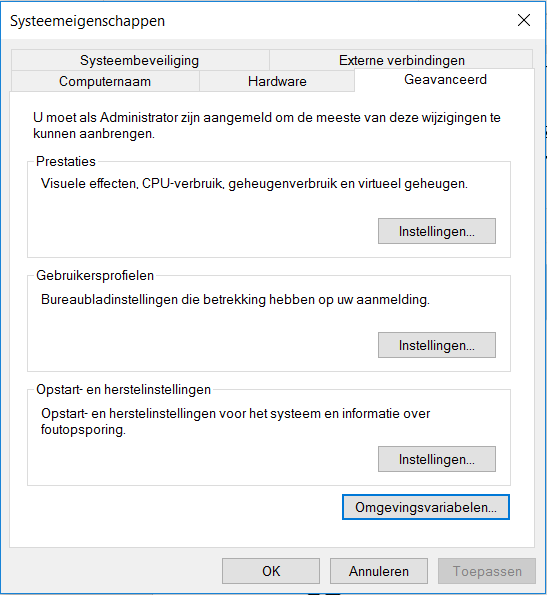
Hoe de installatie van Laravel in zijn werk gaat, dat kan je vinden op <https://laravel.com/docs/8.x>. Echter zit er toch een lastig onderdeel in deze installatie en dat is het zetten van de omgevingsvariabelen.

Indien je dit niet doet, zal het composer commando niet herkend worden. Dit komt omdat het systeem niet weet waar dit commando voor dient en waar de code achter dit commando staat.

Om dit op te lossen moet je dus jouw $PATH omgevingsvariabelen veranderen. Hier moet namelijk composer bij komen te staan.

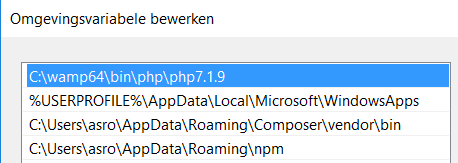
Ga naar het onderdeel “De omgevingsvariabelen van het systeem bijwerken” in het Configuratiescherm.

Vervolgens klik je in dit scherm op de knop *Omgevingsvariabelen…*



In het scherm wat nu getoond wordt, staat de $PATH omgevingsvariabele. Dubbelklik op de path variabele.

Zorg dat in deze lijst de referentie naar Composer komt te staan. Dit is de vendor\bin map van waar jij Composer geïnstalleerd hebt.



Zodra dit gedaan is, kan composer de Laravel installer installeren. Open Git bash en voer hier het volgende commando uit:



Hierdoor kan er nu vanaf elke locatie waar jij een Laravel webapplicatie wil beginnen een nieuwe versie van Laravel geïnstalleerd worden. Dit kan door het “laravel new <naam van het project>” commando, bijvoorbeeld zo:



## **Configuratie**

Voor de rest van de configuratie van het zojuist gerealiseerde nieuwe Laravel webapplicatie zie het hoofdstuk *Configuration* van de documentatie (<https://laravel.com/docs/8.x#configuration>).

# **Een Laravel project**

Indien je de installatie van Laravel doorlopen hebt, een Laravel project aangemaakt hebt met een bepaalde naam en ten slotte dit project geconfigureerd hebt, heb je een nieuw Laravel project.

Er is een hele mappenstructuur opgebouwd en er zijn al bestanden geïnstalleerd waar het e.e.a. aan code in staat. Dit hoofdstuk gaat over de kern van Laravel en wat deze mappenstructuur en bestanden dus inhouden.

## ENV file

Een belangrijk bestand in een Laravel project is de .env bestand. Deze is te vinden in de root map van de Laravel installatie. In dit bestand staan de omgevingsvariabelen. Denk hier bijvoorbeeld aan: welke database wordt er gebruikt, waar vind je die en wat zijn hiervan de credentials.

Een Laravel project kan meerdere .env bestanden hebben. Hierdoor kan je bijvoorbeeld een test .env bestand en een productie .env bestand hebben. Hierdoor wordt het mogelijk om bijvoorbeeld een andere database te gebruiken voor het testen als voor de website die draaiende is.

## Composer.json en package.json bestanden

Verder staat er in de root van de Laravel applicatie ook een composer.json en een package.json bestand.

In het composer.json bestand staat een lijst van PHP packages die momenteel gebruikt worden door het Laravel project.

In het package.json bestand staat een lijst van javascript packages die momenteel gebruikt worden door het Laravel project.

## Routes folder

In de routes folder staan een aantal PHP bestanden. De belangrijkste voor ons zijn het api.php bestand en het web.php bestand.

De routes in het api.php bestand zijn naar API routes. Deze routes zijn standaard geprefixt door /api/<de specifieke route>. Dit zijn over het algemeen de routes die puur alleen data ophalen (geven dus geen view terug).

De routes in het web.php bestand zijn naar WEB routes. Deze routes hebben geen prefix en geven een view terug.

Een voorbeeld van het verschil tussen een WEB route en een API route voor de /schools endpoint.

|  |  |  |
| --- | --- | --- |
|  | API | WEB |
| Route | <https://www.desite.nl/api/schools> | <https://www.desite.nl/schools> |
| Return value |  |  |

## MVC in Laravel

Deze paragraaf geeft een kleine overview over hoe de MVC principe toegepast wordt in Laravel. Later in de reader gaan we zien hoe dit in de praktijk toegepast wordt.

Het MVC (Model View Controller) principe wordt dus toegepast in de Laravel structuur. Zo zijn de models te vinden in de map *app,* de views te vinden in de map *resources/views* en de controllers in de map *app/Http/Controllers*.

Mocht het MVC principe nog niet duidelijk zijn, kijk dan eens naar deze tutorial: <https://www.youtube.com/watch?v=TuH_lqeDnUc>

**Model**

Elk model dat je binnen jouw Laravel project aanmaakt extends de Illuminate\Database\Eloquent\Model klasse. Dit is een standaard klasse die door Laravel gegeven wordt.

Deze klasse zorgt ervoor dat er al het e.e.a. ingesteld kan worden voor een model. Een voorbeeld van zo’n instelling zijn timestamps. Als deze variabele op true wordt gezet, dan zorgt Laravel er automatisch voor dat er in de database een created\_at veld gezet wordt op het moment dat je een model aanmaakt in de database en een updated\_at veld gezet wordt op het moment dat je een model opnieuw opslaat.

Er is hier veel mogelijk, dus voor meer instellingen zie: <https://laravel.com/docs/8.x/eloquent#defining-models>

**View**

Een view is in Laravel een .blade.php file. Dit klinkt gek, maar blade is een templating engine. Je kan blade code gebruiken in combinatie met basis PHP code. Later in de reader duiken we nog verder in blade.

Om je vast een kort idee te geven van blade, zie: <https://laravel.com/docs/8.x/blade#introduction>

**Controller**

Een controller kan eigenlijk twee soorten resultaten terug geven, namelijk platte data of een view. Dit ligt eraan of een deze controller benaderd wordt vanuit een API of een WEB route.

Een controller voorziet view’s van data uit een model, communiceert met models en views, toont resultaten (data of view) en wordt voornamelijk benaderd vanuit een route of view (request).

# Een project in de praktijk

In dit hoofdstuk gaan we samen een boodschappenlijstje voor een verjaardag maken met behulp van Laravel. Deze opdracht dient puur alleen ter praktische introductie van Laravel.

## Routing

Het begint bij een webapplicatie allemaal bij een bepaalde route (URL). Hierdoor weet een website welke pagina hij moet laden en dus wat hij moet doen.

**Opdracht 1:** Maak een nieuw Laravel project aan met de naam shoppingcart.

**Stap 1:** Open Git bash

**Stap 2:** Ga naar de map waar je het project wil hebben (Commando: cd <path>)

(cd staat voor change directory)

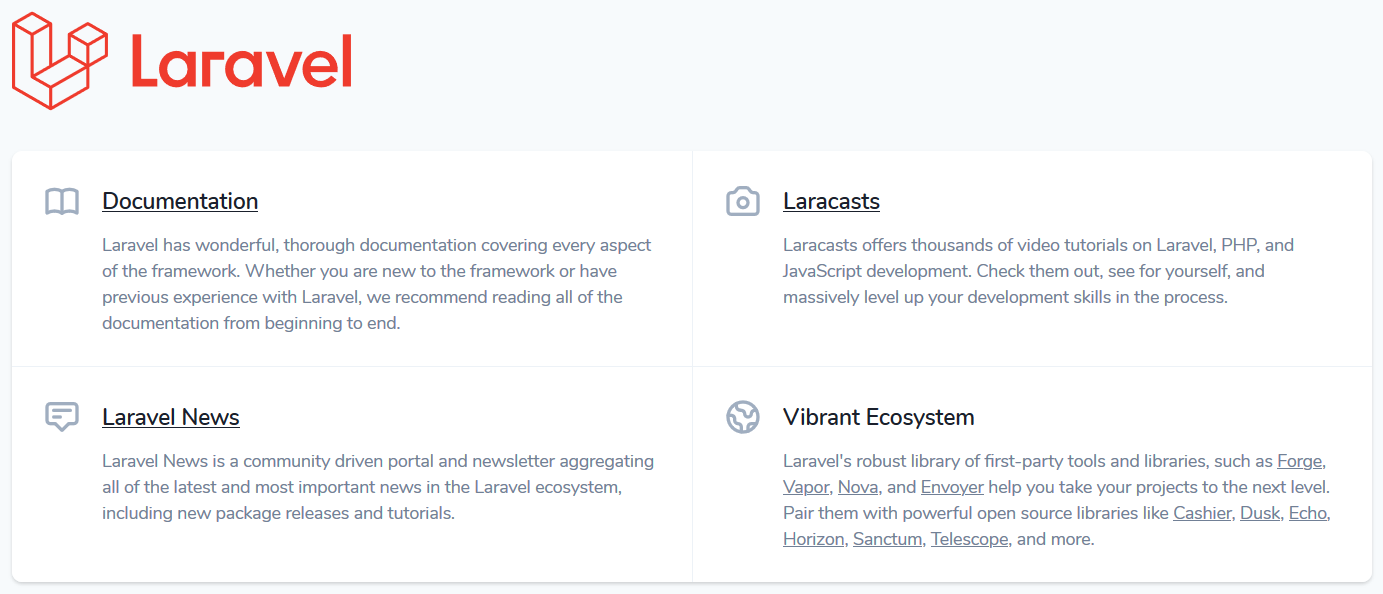
**Stap 3:** Maak het project aan (Commando: laravel new shoppingcart).

**Site draaien vanuit Wamp of Xampp:**

Als je nu naar localhost/shoppingcart/public/index.php gaat, dan zie je als het goed is de onderstaande pagina.

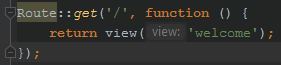
**Site draaien vanuit het “php artisan serve” commando:**

Als je op de link klinkt die gegeven wordt nadat je dit commando uitgevoerd hebt, dan zie je als het goed is de onderstaande pagina.



Blijkbaar is er dus al een homepage route die ervoor zorgt dat er een soort homepage getoond wordt.

Dit klopt ook. Kijk maar eens in het *routes/web.php* bestand. Hier staat de volgende code:

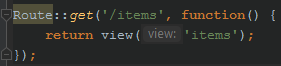


Wat hier staat is het volgende: als ik een get request krijg naar / (de homepage dus), geef dan de view *welcome* terug. Views stonden in de map *resources/views*. Als we in deze map kijken, dan zien we dat daar een *welcome.blade.php* bestand staat.

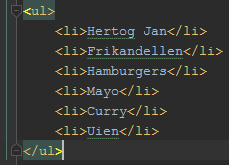
Dit bestand wordt dus geopend als er een get request verstuurd wordt naar de homepage.

**Opdracht 2:** Maak een get route naar /items die een lijst van boodschappen (view) terug geeft.

**Stap 1:** Voeg de route aan het *routes/web.php* bestand toe.



**Stap 2:** Maak de view items aan (hier zijn geen artisan commands voor, dus maak een *items.blade.php* file aan in de *resources/views* folder)  
**Stap 3:** Maak in het *items.blade.php* bestand een lijstje met boodschappen, bijvoorbeeld:



**Stap 4:** Ga naar: <http://localhost/shoppingcart/public/items>. Als het goed is zie je nu een lijst van boodschappen. Indien je het “php artisan serve”commando gebruikt om jouw website te draaien, ga dan naar: <http://127.0.0.1:8000/items> of <http://localhost:8000/items> .

## In combinatie met de database

We hebben nu een hard coded boodschappenlijst. Het zou natuurlijk een stuk mooier zijn als deze data uit de database komt.

**Tip:** Voor opdracht 1 kun je ook het bijgeleverd sql bestand gebruiken.

**Opdracht 1:** Maak een database aan.

**Stap 1:** Maak een database aan met de naam *shoppingcart*.

**Stap 2:** Maak een tabel aan met de naam *items*.

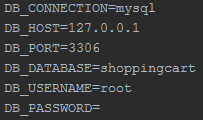
**Stap 3:** Geef de tabel 2 velden: id (integer, primary key, auto increment) en title (varchar(255)).

**Stap 4:** Zet de lijst die je bij de vorige opdracht gebruikt hebt in de database.

Nu hebben we de database gereed. We hebben gegevens die we op kunnen halen. Volgens het MVC principe is het model verantwoordelijk voor het ophalen van data. Dus we moeten een model aanmaken.

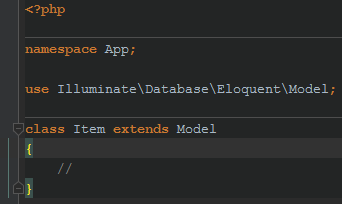
**Opdracht 2:** Maak de database connectie.

**Stap 1:** Pas de database gegevens aan in het .env bestand. De database gegevens in dit bestand zien er nu bijvoorbeeld zo uit:



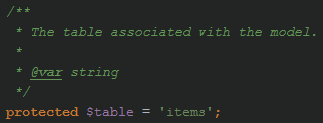
**Opdracht 3:** Maak het Item model aan.

**Stap 1:** Maak het Item model aan (commando: php artisan make:model Item).

Als je naar de map *app/Models* gaat, dan zie je dat er nu een bestand genaamd Item.php bij gekomen is. Dit bestand is nu een leeg model genaamd Item dat Model extends.

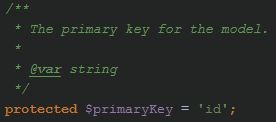
Oftewel: Item heeft alle functionaliteiten van de Illuminate\Database\Eloquent\Model klasse.

**Stap 2:** Zet de tabelnaam van het model (zo weet het model waar hij zijn data vandaan moet halen.



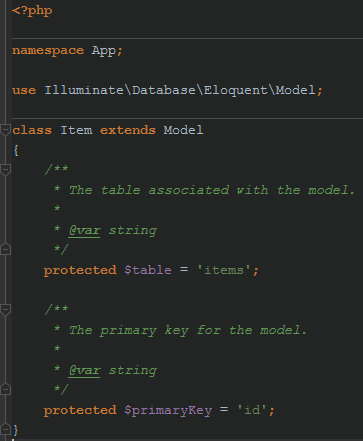
Dit hoeft eigenlijk niet, omdat laravel zelf al standaard een "s" achter de model naam plakt om de tabelnaam te zoeken. Maar als je tabel in de database dus bijvoorbeeld "item" is, dan kan laravel geen match maken. Dan gebruik je dus "protected $table='item'".

**Stap 3:** Geef aan wat de primary key van de tabel is.

Als de primary key van de tabel “id” is, dan is het voor Laravel niet nodig om dit aan te geven. In alle andere gevallen moet dit aangegeven worden.

Het kan geen kwaad om dit toch aan te geven.

Als het goed is ziet het complete Item model er nu zo uit:



We hebben dus een Model genaamd Item, dat alle functionaliteiten heeft van de Laravel Model klasse.

We hebben aangegeven dat de tabel in de database ‘items’ heet en dat de primary key in die tabel ‘id’ is.

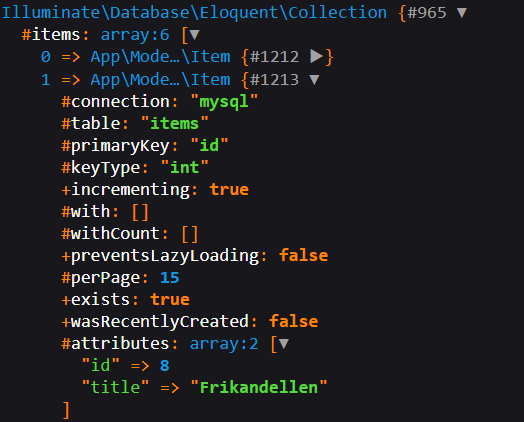
Verder weet Laravel door de .env file nu welke database hij moet benaderen en met welke credentials.

We zijn dus klaar om data op te halen en deze door te geven aan een view. Als je MVC kent, dan weet je dat we nu een controller nodig hebben.

**Stap 4:** Laten we even nog een test doen zonder controller. Ga naar de "routes/web.php" en maak een nieuwe route onder de reeds bestaande:  
*Route::get('/test', function () {  
dd(Item::all());  
});*

Bovenaan de " routes/web.php" staat een regel met " use Illuminate\Support\Facades\Route;". Plaats net daarboven de volgende regel:  
*use App\Models\Item;*

We gebruiken dus het model dat voor ons alle data uit de tabel zal gaan halen (::all()) en dan zal tonen met het commando "dd" (dump and die). Ga nu naar de browser en gebruik <http://127.0.0.1:8000/test> of <http://localhost:8000/test> . Je krijgt een collectie van alle gegevens.

Dit is een snelle manier om even te testen of de database goed is ingesteld en je model wel werkt.  
  
  
Verwijder nu de test route en de "*use App\Models\Item;*" regel.

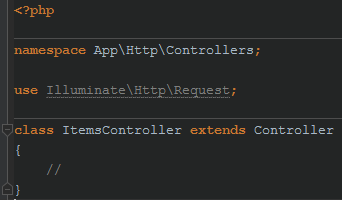
## Het MVC principe

In deze paragraaf gaan we ervoor zorgen dat de controller communiceert met de view en hierbij de juiste data doorgeeft.

**Opdracht 1:** Aanmaken en inrichten van de controller.

**Stap 1:** Maak de *ItemsController* aan (Commando: php artisan make:controller ItemsController).

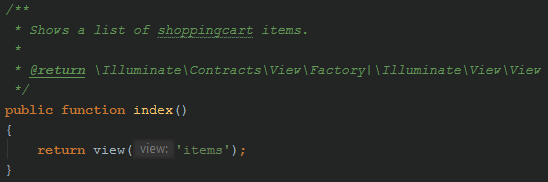
Controllers bevinden zich zoals al eerder aangegeven in de *app/Http/Controllers* map. Na het uitvoeren van bovenstaand commando is er in deze folder een *ItemsController.php* file bij gekomen. Dit bestand is gewoon een lege controller en ziet er als volgend uit:

Dit is dus een lege klasse die de Controller klasse van Laravel extends.

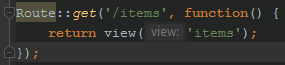
In deze klasse kunnen we methodes toevoegen.

Wij maken hier een methode aan die een lijst van Items terug gaat geven aan.

**Stap 2:** Voeg de *index* methode toe aan de *ItemsController* die de items view terug geeft.



**Stap 3:** Laat de get route van “/” verwijzen naar de *index* methode van de *ItemsController* in het *routes/web.php* bestand. Verander dus dit:



In:



De tweede parameter van de get functie is een array die bestaat uit de controller en de functie in die controller die we willen gebruiken (die 'index' functie moeten we nog maken). Als je nu naar <http://localhost/shoppingcart/public/items> (of <http://127.0.0.1:8000/items> indien je het “php artisan serve” commando gebruikt) gaat, zou je nog steeds de hard coded boodschappenlijst moeten zien.

**Stap 4:** Haal de data op vanuit het model.

Dit doen we met eloquent (het ORM (object relational mapping) van Laravel). Hier gaan we later nog verder op in met de reader. Alle items kun je ophalen met de volgende regel code:



Dit hadden we al gezien in de test route. Nu plaatsen we een variabele er voor zodat we de data kunnen hergebruiken.

Plaats deze regel in de *index* methode van de *ItemsController* (maak de index methode). Echter kent hij nu de klasse *Item* niet, dus moeten we deze even bovenaan toevoegen. Voeg deze regel bovenaan de itemsController toe bij de use statements:

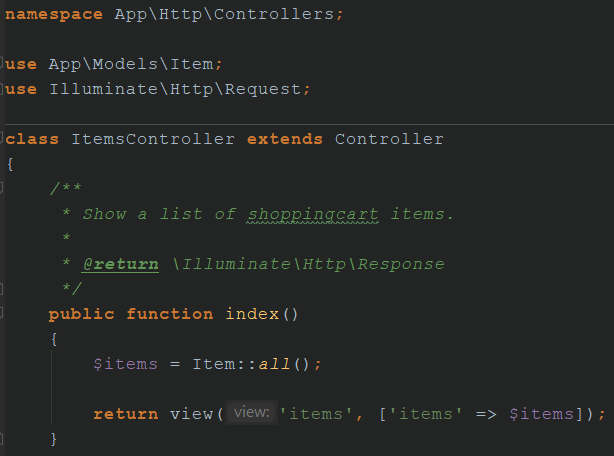


**Stap 5:** Geef de data door aan de view.

Dit kan d.m.v. een key => value array:



Als het goed is, ziet de controller er nu als volgt uit:



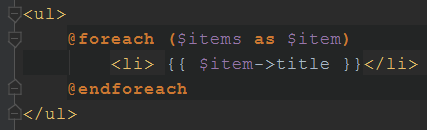
De key van de array mag ook een andere naam hebben bijv: ItemsuitDeController. De naam die je instelt als key moet je dadelijk gebruiken in de view.

**Opdracht 2:** Geef de data weer in de view.

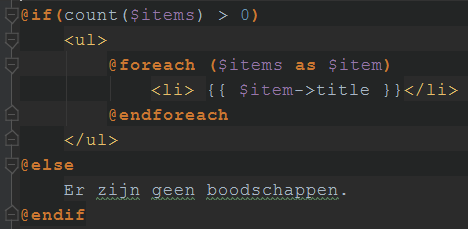
Voor het weergeven van data in een view maken we gebruik van templating engine genaamd Blade. Heel diep gaan we nu nog niet in op Blade, maar mocht je interesse hebben, dan kan je hier de documentatie van Blade vinden: <https://laravel.com/docs/8.x/blade>.

Statements zoals foreach, if, else if, else, etc. worden aangegeven met een @. Een echo wordt aangegeven door kringelhaken. Let ook in onderstaande code goed op hoe deze @ statements afgesloten worden. Ook dit kun je lezen in bovenstaande documentatielink.

**Stap 1:** Geef de boodschappenlijst weer.



**Stap 2:** Geef aan indien er geen boodschappen zijn.

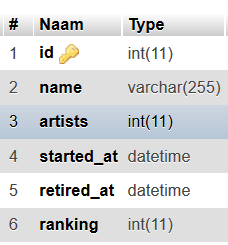


## Zelfstandige opdracht

Wat je in de afgelopen paragrafen aan technieken hebt gezien, ga je nu toepassen in een zelfstandige opdracht. Je gaat een opdracht maken die zich bezig houden met bands.

**Tip:** Voor de database kun je ook het bijgeleverd sql bestand gebruiken.

Maak de volgende database aan:

De primary key voor deze tabel is id en deze heeft een auto-increment.

Vul deze database met wat testwaarden.

Er zijn in deze applicatie 2 WEB routes, namelijk: *<de website>/bands* en   
*<de website>/bands/{id}*

**Gebruik het MVC principe!** Dus: een route verwijst naar een controller, die controller haalt vervolgens via een model de juiste data op en geeft deze door aan de view.

De eerste route geeft een overzicht van alle bands in de tabel. De kolommen in deze tabel zijn: name, artists (het aantal artiesten in de band) en ranking (1 tot 5 sterren).

De tweede route geeft alle details over een specifieke band. De gegevens van een band worden door middel van het id opgehaald. Voor meer informatie over parameters doorspelen naar een controller, zie: <https://laravel.com/docs/8.x/controllers#basic-controllers>.

Om één specifieke band met een bepaald id op te halen, kun je de find methode gebruiken van Eloquent (<https://laravel.com/docs/8.x/eloquent#retrieving-single-models>).

**In het totaal heb je dus het volgende nodig:**

* Twee routes (*<de website>/bands* en *<de website>/bands/{id}*)
* Een database zoals hierboven beschreven
* Een model genaamd *Band*
* Een controller genaamd *bandsController*
* Twee views (*bands.blade.php* en *band.blade.php*)

**Voor extra uitdaging:** Maak een tabel aan genaamd artists en zorg ervoor dat deze aan een band gekoppeld kunnen worden. Deze artiesten moeten ook in de vorm van een lijstje getoond worden in de band.blade.php pagina.

**Tip:** <https://laravel.com/docs/8.x/eloquent-relationships#one-to-many>

# Views

Views in Laravel maken gebruik van de templating engine genaamd Blade. Deze views staan gedefinieerd in de map *resources/views*. In dit hoofdstuk wordt meer uitgelegd over hoe Blade gebruikt wordt, hoe data doorgegeven kan worden aan views en hoe er tussen views geswitcht kan worden.

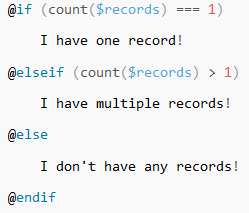
## Blade basics

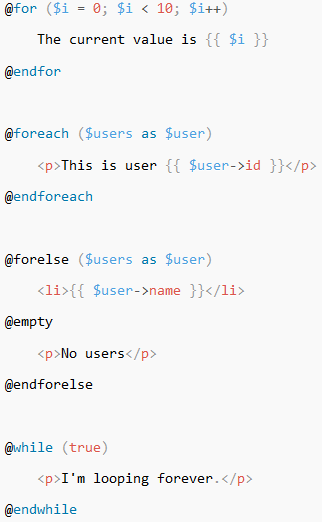
Blade gebruikt PHP, maar heeft wel zijn eigen syntax. Zo ben je in PHP gewend om een echo te gebruiken, maar hier gebruikt Blade een andere syntax voor.

**PHP echo:** echo( $something );

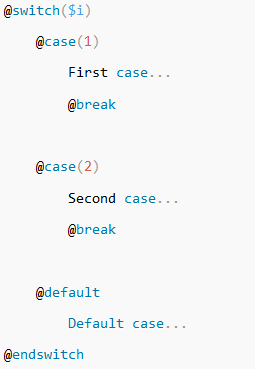
**Blade echo:** {{ $something }}

Zo zijn er nog een aantal functionaliteiten in Blade die iets anders werken, zoals:

**Loops: If | else if | else statements:**



**Switch statements:**



Blade is gelukkig meer dan alleen maar een efficiëntere syntax voor PHP. Blade heeft ook een aantal functionaliteiten toegevoegd, die soms erg handig zijn. Denk hier bijvoorbeeld eens aan de loop variabele: <https://laravel.com/docs/7.x/blade#control-structures>.

## Data doorgeven aan views

In de praktijkopdracht hebben we gezien dat er data doorgegeven kan worden aan views. Dit deden we op deze manier:



Hierboven wordt een array van data doorgegeven aan de *greetings* view. Wat hier belangrijk is om te weten, dat is dat dit een array met key/value pairs moet zijn. Oftewel in dit voorbeeld is ‘name’ de key en ‘Victoria’ de value.

Indien deze array op de bovenstaande manier doorgegeven wordt, kan er in de *greetings* view de value opgehaald worden d.m.v. de key. Bijvoorbeeld op deze manier:



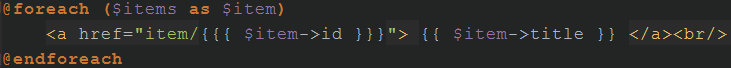
## Switchen van views

Volgens het MVC principe bepaald de controller welke view er getoond wordt en welke data er nodig is om een view te laden. Deze controller hebben we bijvoorbeeld in de praktijkopdracht benaderd via een route:

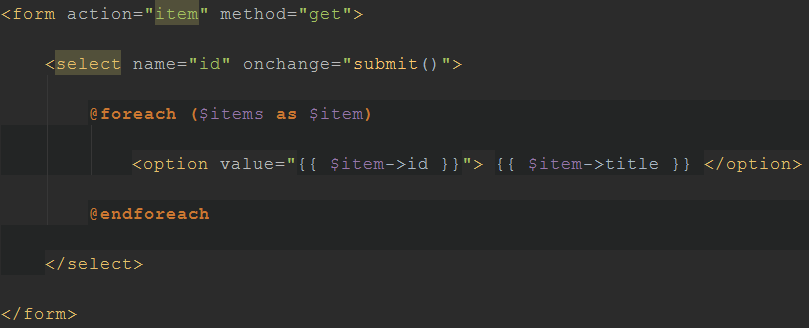


Hierboven is een route gedefinieerd genaamd: *<de website>/items*, die verwijst naar de *index* methode in de *ItemsController*.

Om een andere view te openen zullen we moeten verwijzen naar een controller actie. Dit kan via een route zoals hierboven weergegeven is. Dit kun je op meerdere manieren doen:

**Via een Link:**  


**Via een form submit:**



## Zelfstandige opdracht

We gaan verder bouwen op de opdracht uit hoofdstuk 7 (praktijkopdracht). In deze opdracht hadden we een boodschappenlijstje gerealiseerd. Deze gaan we nu uitbreiden met een extra view.

**De opdracht:** Zorg ervoor dat het boodschappenlijstje uit de praktijkopdracht van hoofdstuk 7 nu ook een detailpagina van een item uit het boodschappenlijstje toont. Deze *detail* pagina moet meer informatie geven over het specifieke product.

* De detail pagina heet *item.blade.php*.
* Breid de database uit met extra detailgegevens van producten (extra kolommen toevoegen aan de shoppingcart tabel, bijvoorbeeld: aantal gram koolhydraten, aantal gram suikers, etc.).
* De detailpagina moet benaderd kunnen worden door op een item uit de boodschappenlijst te klikken.
* De detailpagina toont de extra gegevens van het product waar op geklikt is.

# Forms

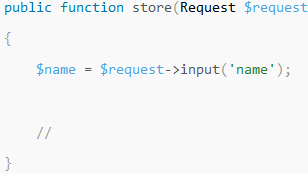
In de afgelopen schooljaren hebben jullie tijdens webdevelopment al een aantal keer formulieren moeten gebruiken. Deze formulieren kunnen ook gebruikt worden in Laravel.

Echter om deze formulieren werkend te krijgen, moet er bij Laravel wel een CSRF veld gezet worden. Forms kunnen op de volgende manier gebruikt worden:



Hierboven staat een formulier weergegeven die een POST uitvoert naar de */profile* route. Deze route zou weer verwijzen naar een bepaalde action van een controller. Op de … zou je jouw input velden kunnen plaatsen die via de POST doorgestuurd moeten worden naar de */profile* route.

Op het moment dat een formulier gesubmit wordt, dan wordt er een HTTP request uitgevoerd. Dit request is van het type *\illuminate\http\request*. Deze request wordt ook als parameter meegegeven aan de controller actie die de form uiteindelijk zal benaderen. Dit ziet er bijvoorbeeld zo uit:



Hier wordt bijvoorbeeld de name opgehaald uit de request. Echter kan je nog op meerdere manieren data uit een request halen. Meer informatie hierover vind je hier: <https://laravel.com/docs/8.x/requests#retrieving-input>.

## Zelfstandige opdrachten

Voor deze opdracht gaan we verder werken aan de praktijkopdracht uit hoofdstuk 7. Als het goed is heb je deze al uitgebreid in hoofdstuk 8, zo niet: maak eerst de zelfstandige opdracht uit hoofdstuk 8.

**Opdracht 1:** Bereid het *Shoppingcart* project uit, zodat het mogelijk wordt om items toe te voegen aan de lijst.

* Maak hiervoor een extra view *new-item.blade.php*.
* Indien een item toegevoegd is aan de lijst, wordt je doorverwezen naar de *items* view.

**Opdracht 2:** Bereid het *Shoppingcart* project uit, zodat het mogelijk wordt om items te verwijderen uit de lijst.

* Voeg een knop toe per item in de boodschappenlijst. Indien er op deze knop gedrukt wordt, wordt het desbetreffende item uit de lijst verwijderd.
* Zorg ervoor dat de view ook geüpdatet wordt.

**Opdracht 2:** Bereid het *Shoppingcart* project uit, zodat het mogelijk wordt om de details van een item te wijzigen.

* Maak hiervoor een extra view *update-item.blade.php*.
* Indien een item gewijzigd is, wordt de gebruiker doorverwezen naar de *item* view.

# Form request validation

Met een form request worden allerlei gegevens meegestuurd vanuit velden in de form. Deze velden kun je met Laravel valideren. Hierbij kan er bijvoorbeeld gevalideerd worden op: het ingevoerde gegevenstype, is een veld ingevuld, maar ook op een foreign key.

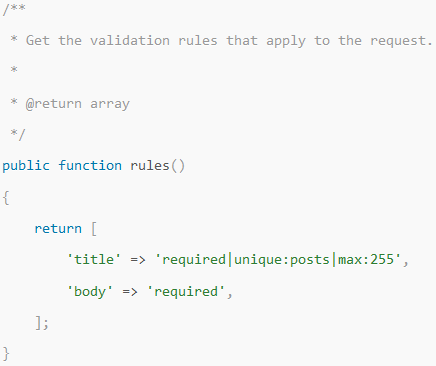
Het valideren gaat door middel van een form request class.

**Het aanmaken van een form request klasse.**

Een form request klasse kan aangemaakt worden door het volgende commando:   
php artisan make:request <RequestNaam>

**Wat doet deze request klasse?**

Deze artisan call maakt een request klasse aan met de naam *RequestNaam* en plaatst deze in de map *app/Http/Requests*. Deze klasse bevat een functie genaamd *rules* die een array van regels teruggeeft waar de specifieke request aan moet voldoen. Een voorbeeld van deze *rules* functie:



Bij deze request is het dus noodzakelijk om een *title* en een *body* veld op te geven.

Verder moet de *title* ook nog uniek zijn in de *posts* tabel en mag deze *title* niet langer zijn dan 255 karakters.

Regels scheid je dus met een |.

Er zijn echter nog veel meer regels die je kunt gebruiken bij het valideren van data. Kijk eens naar: <https://laravel.com/docs/8.x/validation#available-validation-rules>.

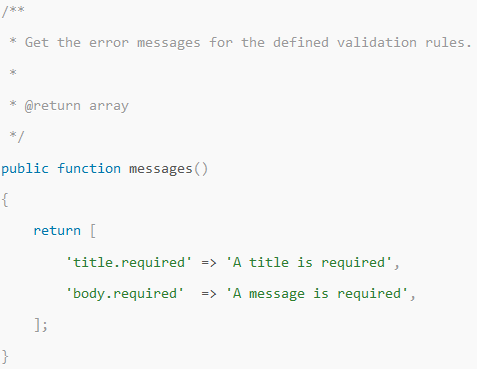
**Het gebruiken van de request klasse**

Voor het gebruiken van de request klasse moet je deze in de controller als parameter opgeven. Voorheen gebruikte je als parameter: Request $request. Hier kun je echter ook jouw eigen request klasse voor gebruiken. Dit ziet er zo uit:

## Error messages

In de vorige bladzijde heb je gezien hoe je requests kunt valideren. Nu is het wel zo netjes om ook een nette foutafhandeling te geven bij deze niet geslaagde requests. Dit kan je met Laravel doen door middel van de *messages* functie.

Deze *messages* functie zit in de *Request* klasse en ziet er als volgt uit:

Hierbij kun je per gestelde regel in de *rules* methode een passende foutmelding geven.

De *messages* functie geeft een array terug van foutmeldingen. Deze lijst is een key-value array. Met als key de validation rule en als value de error message.

Hoe je foutmeldingen op kunt halen lees je hier: <https://laravel.com/docs/8.x/validation#working-with-error-messages>.

## Zelfstandige opdracht

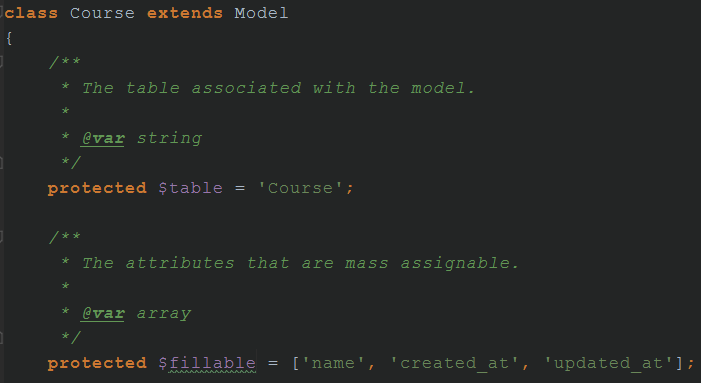
Zorg ervoor dat de requests die je gebruikt hebt bij de zelfstandige opdracht uit hoofdstuk 9 een eigen custom *Request* klasse krijgen. In deze klasse moet er gevalideerd worden op de volgende input:

* Gegevens mogen niet ontbreken (nieuw artikel toevoegen en artikel bewerken)
* Er mag geen zelfde artikel in de boodschappenlijst komen (nieuw artikel toevoegen)
* Lengte van ingevoerde velden mogen niet langer zijn dan de maximale lengte van het gegevenstype in de database (nieuw artikel toevoegen en artikel bewerken).

Ook moet ervoor gezorgd worden dat op het moment dat er een form request als niet valide beschouwt wordt deze een eigen custom foutmelding geeft die correspondeert met de regel of regels die geschonden worden.

# CRUD met Eloquent

Eloquent is Laravel’s ORM (Object-Relational Mapping). Een ORM zorgt ervoor dat data gekoppeld wordt aan bestaande klassen en dat queries wat meer object georiënteerd geschreven kunnen worden.

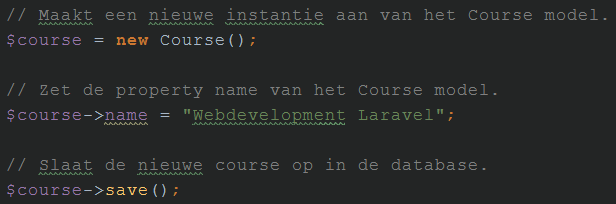
In onderstaande voorbeelden wordt een voorbeeld model genaamd Course gebruikt: 

## Create

Een nieuw record toevoegen aan de database kan op verschillende manieren, namelijk met en zonder mass assignment. Zonder mass assignment kan altijd zonder extra dingen in jouw model in te stellen.

**Zonder mass assignment:**

Hieronder staat uitgelegd hoe een insert gedaan kan worden zonder mass assignment. Wat belangrijk is om te weten, dat is dat er pas iets in de database opgeslagen wordt indien de *save* functie uitgevoerd wordt.

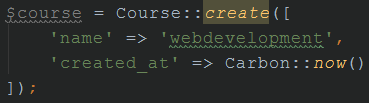


**Met mass assignment:**

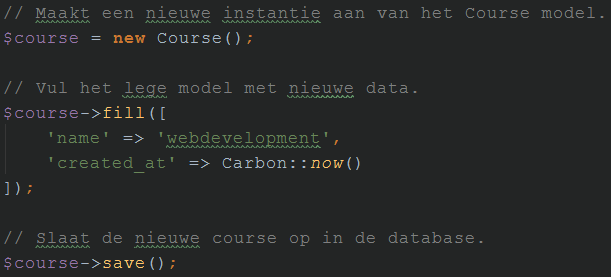
Mass assignment geeft eigenlijk al aan dat je een “massa aan data” wil inserten. Dit is gemakkelijk als je bijvoorbeeld een array van data op wil slaan in de database. Om

**Let op:** om mass assignment toe te kunnen passen dienen alle velden waarop dit van toepassing is aangegeven te worden in de *$fillable* property van het desbetreffende model. Zie het voorbeeld *Course* model op de vorige pagina.

Voor het gebruiken van mass assignment gebruiken we de *Create* methode van een model.



Als je eerst een lege instantie van een model aan wil maken, om deze vervolgens te vullen, kan dit ook. Hiervoor gebruik je de *fill* methode. Ook hier moet dan daarna nog de *Save* methode uitgevoerd worden om deze daadwerkelijk in de database op te slaan.



## Read

Om één enkel model op te halen voor een specifieke primary key, kan de *find* methode gebruikt worden binnen Laravel. Standaard zoekt Laravel dan naar een veld genaamd *id*. Is jouw veld dat als primary key dient anders dan id, dan moet je de aangeven in de property *$primaryKey* aangeven in het model.

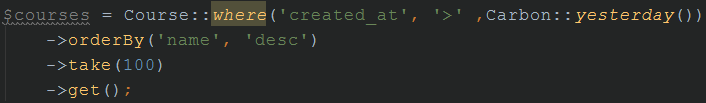


Wil je meerdere alle resultaten ophalen van een bepaald model, dan gebruik je de *all* methode.



Wil je een selectie van een bepaald model ophalen, dan gebruik je de *get* methode. Bij deze methode is het ook mogelijk om filtering toe te passen door bijvoorbeeld *where* clauses.

Hier worden bijvoorbeeld de top 100 Courses opgehaald die aangemaakt zijn sinds gisteren gesorteerd op het *name* veld. Deze query is opgebouwd met behulp van de querybuilder.



**Voor meer informatie over wat je kan doen met de querybuilder:**

<https://laravel.com/docs/8.x/queries#introduction>

### Collections

Belangrijk om te weten is dat elke database call met de querybuilder die meerdere records terug kan geven een *collection* terug geeft. Deze collection is van het type *Illuminate\Database\Eloquent\Collection*. Deze klasse is door Laravel ontworpen en is eigenlijk niets meer dan een uitgebreidere *array*.

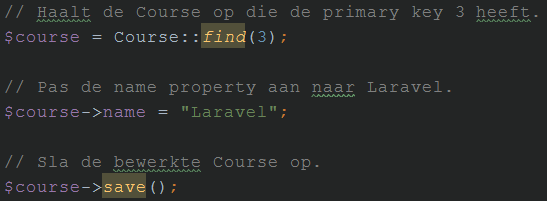
Collections zijn zeer fijn om data te mappen, filteren, sorteren, berekeningen op uit te voeren, etc. Voor alle mogelijkheden van collections zie: <https://laravel.com/docs/8.x/eloquent-collections#available-methods>

**Good practice:** probeer resultaten zoveel mogelijk in de query te filteren. In een query filteren is namelijk efficiënter dan filteren met een collection. Sterker nog: sinds Laravel 8.x zijn veel functionaliteiten van een collection alleen nog maar toegankelijk via de querybuilder.

## Update

Ook de update kent ook de twee opties: met mass assignment en zonder mass assignment.

**Zonder mass assignment:**



**Met mass assignment:**

Deze methode wordt gebruikt indien je meerdere records wilt updaten. Hiervoor gebruik je niet de *save* methode van het model, maar de *update* methode van het model. Deze update methode voert achter de schermen een *fill* methode uit op het model gevolgd door een *save* methode.



## Delete

Het is mogelijk om in Laravel een model voor een specifieke primary key te verwijderen of een zwik aan models.

Om een specifiek model te verwijderen voor een specifieke primaray key gebruik je de *destroy* methode.



Om een specifieke selectie van models te verwijderen kan de *delete* methode gebruikt worden.



### Softdeleting

Bij Laravel is het ook mogelijk om records te softdeleten. Dit houdt in dat records niet verwijderd worden uit de database, maar dat een deleted\_at timestamp gezet wordt.

Alle records die opgehaald worden negeren standaard deze regels. Mocht je deze records toch op willen halen, dan kan dit ook.

Voor meer informatie over softdeleting: <https://laravel.com/docs/8.x/eloquent#deleting-models>

## Timestamps

Indien de *$timestamps* property op *true* wordt gezet in een model, dan zorgt het model ervoor dat er een created\_at timestamp in de database gezet wordt op het moment dat er een nieuw model opgeslagen wordt in de database.

Verder wordt er ook een updated\_at timestamp in de database gezet voor het moment dat een model bewerkt wordt in de database.

Om dit voor elkaar te krijgen moet je voor de tabel in de database die het model gebruikt een created\_at en updated\_at veld plaatsen. Deze velden zijn van het gegevenstype *DateTime*.

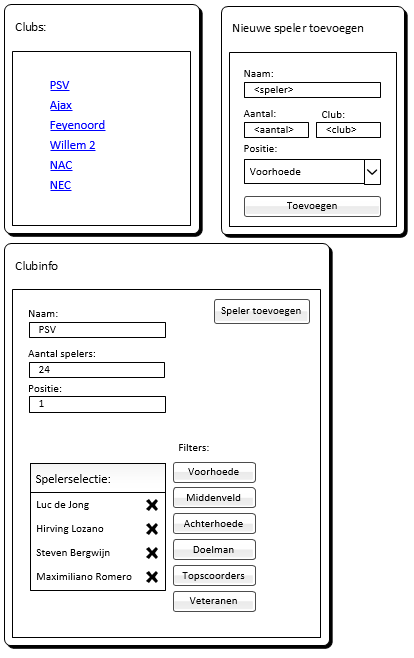
## Zelfstandige opdracht

Maak een CRUD applicatie om deze cursussen te onderhouden. Met de applicatie kun je cursussen tonen, veranderen en verwijderen.

* Gebruik school\_of\_sciences.sql
* Pas soft deleting toe

## Zelfstandige opdracht

Maak een CRUD applicatie om een voetbalelftal samen te stellen. De pagina’s moeten er zo uit komen te zien (rode pijlen geven de navigatie tussen schermen weer):



**Het aanmaken van de database**

1. Maak een database aan genaamd *Voetbalclubs*.
2. Maak een tabel aan genaamd clubs en geef deze de volgende velden: (naam (PK, varchar(255)), aantal\_spelers (integer), positie (integer))
3. Maak een tabel aan genaamd spelers en geef deze de volgende velden: (id (PK, auto increment), club\_naam (varchar(255)), doelpunten (integer), positie (varchar (255))

**Meer uitleg over de werking:**

* De spelerstabel werkt met softdeleting (indien een model verwijderd is, wordt deze als een veteraan beschouwt).
* Een speler verwijderen kan door middel van op het kruisje te klikken naast de speler
* De spelerlijst moet gefilterd kunnen worden op positie, veteranen en topscoorders.
  + Deze filtering moet via collections gefilterd worden.
  + Bij de topscoorders moeten de 3 spelers met de meeste doelpunten getoond worden.

**Extra opdracht:**

* Zorg ervoor dat spelers ook bewerkt kunnen worden.
* Zorg dat er een CRUD gemaakt wordt voor de clubs