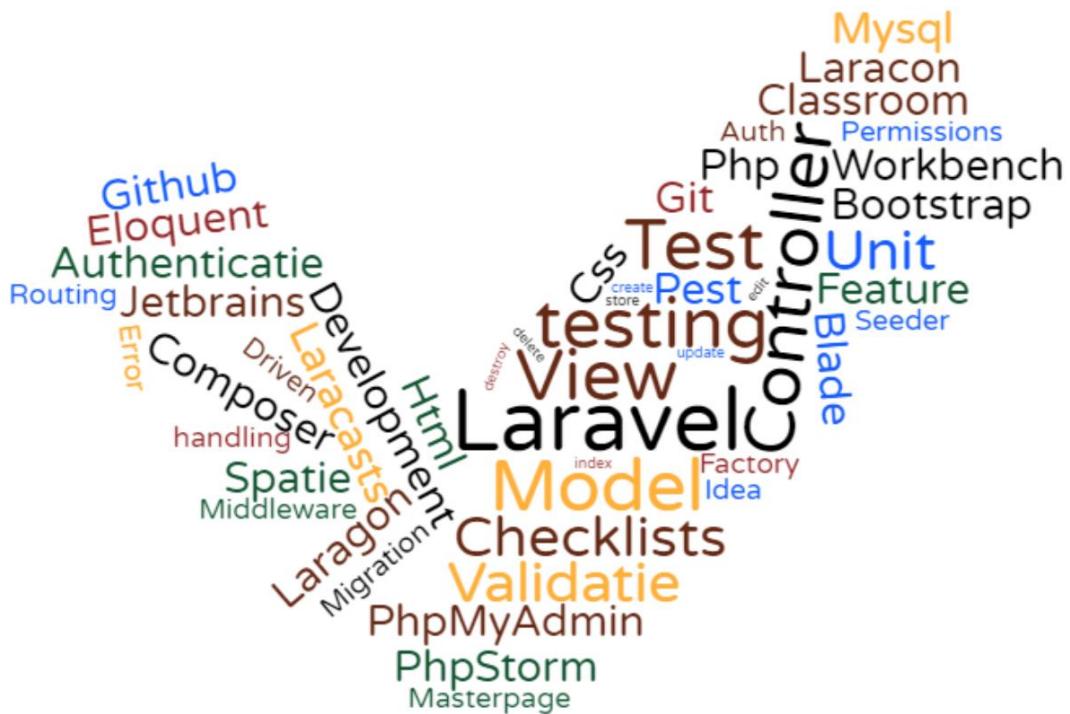


# Laravel Project



# Laravel



# Marcel Koningstein

## Techniek College Rotterdam locatie Spijkenisse

# Inhoud

1.	Inleiding .....	6
1.1.	Versiebeheer.....	7
1.2.	Wishlist.....	7
1.3.	Omgeving & Installatie .....	8
1.4.	Casus beschrijving.....	10
1.5.	Opdrachten.....	11
2.	Categories .....	12
2.1.	Model.....	12
2.2.	Migration.....	14
2.3.	Opdracht 1: model en migration.....	22
2.4.	Factory.....	23
2.5.	Seeder .....	26
2.6.	Opdracht 2: factory & seed .....	29
2.7.	Lay-out met Breeze .....	30
2.8.	Opdracht 3: layout met Tailwind.....	34
3.	Resource Controller .....	35
3.1.	Aanmaken van de resource controller.....	35
3.2.	Routing .....	38
3.3.	Index in controller .....	40
3.4.	Index in View .....	42
3.4.1.	Van url naar scherm .....	44
3.5.	Opdracht 4: index .....	46
3.6.	Create .....	47
3.7.	Store .....	51
3.8.	Opdracht 5: create & store .....	54
3.9.	Validatie in de store .....	54
3.10.	Opdracht 6: validatie .....	60
3.11.	Show .....	61
3.12.	Opdracht 7: show .....	64
3.13.	Edit .....	65
3.14.	Update .....	68
3.15.	Opdracht 8: edit & update .....	71
3.16.	Delete.....	72
3.17.	Destroy .....	76
3.19.	Opdracht 9: delete & destroy .....	77
4.	Rollen en Permissies .....	78

4.1.	Role and Permission Seeder .....	81
4.2.	User Seeder .....	83
4.3.	Opdracht 10: Roles & Users.....	87
4.4.	Middleware gebruiken in Routes.....	88
4.5.	Authenticatie .....	90
4.6.	Permissies in de Controller.....	95
4.7.	Opdracht 11: Permissions .....	99
5.	Product & Price .....	100
5.1.	Models .....	101
5.2.	Migrations .....	104
5.3.	Opdracht 12: Models en Migrations .....	107
5.4.	Factory.....	108
5.5.	Seed .....	110
5.6.	Opdracht 13: Factory.....	113
5.7.	Product Controller .....	114
5.7.1.	Index .....	114
5.7.2.	Installeren debugbar .....	118
5.7.3.	Uitbreiden van de index .....	121
5.7.4.	Oplossen Seed probleem .....	126
5.7.5.	Opdracht 14: Seeder .....	128
5.7.6.	Opdracht 15: Index van Tasks.....	128
5.7.7.	Permissies voor product controller .....	129
5.7.8.	Opdracht 16: Permissies voor tasks .....	130
5.7.9.	Create.....	131
5.7.10.	Store .....	133
5.7.11.	Opdracht 17: Create & store voor tasks met validatie .....	138
5.7.12.	Show .....	139
5.7.13.	Opdracht 18: Show voor task.....	141
5.7.14.	Edit .....	142
5.7.15.	Update .....	145
5.7.16.	Opdracht 19: Edit & update voor tasks .....	149
5.7.18.	Delete .....	150
5.7.19.	Destroy .....	152
5.7.20.	Opdracht 20: Delete & destroy voor tasks.....	153
5.7.21.	Category Test .....	154
6.	Unit & Feature testing.....	158
6.1.	Omgeving .....	158

6.2.	Gebruik van Pest / Phpunit .....	160
6.3.	Response Status .....	163
6.4.	Schrijven van een test .....	164
6.5.	Feature test.....	167
6.6.	Opdracht 21: Schrijven van een Featuretest .....	169
6.7.	Dataset.....	170
6.8.	Gebruik van een database.....	172
6.9.	Naamgeving en Setup .....	176
7.	Product Tests.....	178
7.1.	Product Index.....	179
7.2.	Product Create.....	186
7.3.	Product Store .....	188
7.4.	ProductStoreCheck .....	195
7.5.	ProductStoreCheck met debugbar.....	200
7.6.	Verder met ProductStoreCheck.....	204
7.7.	Product edit.....	207
7.8.	Product update.....	209
7.9.	ProductUpdateCheck .....	212
7.10.	Product Delete.....	216
7.11.	Product Destroy .....	218
7.12.	Unit Testing.....	222
8.	TDD met orders.....	225
8.1.	Maken van testen voor Order Index.....	225
8.2.	Starten met test van Order Index .....	227
8.3.	Order Unit Test .....	238
8.4.	Verder met TDD van Index.....	239
8.5.	Create .....	246
8.6.	Store .....	253
8.7.	Edit .....	261
8.8.	Update .....	268
8.9.	Delete.....	273
8.10.	Destroy .....	280
9.	Omgeving.....	284
9.1.	Wamp.net installatie met Laravel .....	284
9.2.	Laragon gebruiksklaar maken .....	292
9.3.	Installatie met laravel installer binnen laragon .....	294
9.4.	Installatie met quickapp binnen laragon .....	298

10.	Tips & Tricks .....	302
10.1.	Gebruik van phpstorm's terminal met laragon.....	302
10.2.	Composer update .....	305
10.3.	Database in PhpStorm .....	306
10.4.	NPM errors met Laragon .....	308

# 1. Inleiding

Dit is een lessenserie om stap voor stap de basis van MVC te leren door middel van het framework Laravel. Bij het schrijven van deze lessenserie is de huidige versie van Laravel 8. In de lessenserie leer je hoe je de omgeving kan opzetten, de basisvaardigheden van Laravel en hoe je een werkelijk project kan maken.

Aan het einde van deze lessen kan/weet de student het volgende:

- De student weet wat een MVC is
- De student kan een werkomgeving realiseren die geschikt is om te werken met een MVC
- De student kan views aanpassen
- De student kan werken met Migrations, Seeds en Factory
- De student kan werken met Controllers
- De student kan werken met Models
- De student kan een database aanspreken via Laravel
- De student weet wat hidden files zijn
- De student kan werken met relaties
- De student kan werken met authenticatie
- De student kan werken met validatie
- De student kan werken met phpunit en pest
- De student kan feature tests schrijven
- De student kan unit tests schrijven
- De student weet wat response statussen zijn
- De student kan ontwikkelen volgens test driven development
- De student kan een simpele website bouwen in Laravel.

Als je deze lessenserie heb gevolgd en je hebt er wat aan gehad, voeg me dan toe op LinkedIn:

<https://www.linkedin.com/in/marcel-koningstein/>

Deze lessenserie heb ik gemaakt door de lessen die ik heb gegeven aan de applicatie klas in Spijkenisse. Mijn dank voor de prettige lessen die ik kon geven en de feedback die ik heb gekregen.

Er zijn een aantal (oud) studenten die ik wil bedanken, voor de feedback die ik heb gekregen en voor de hulp die ze hebben gegeven om deze lessenserie te maken:

- Bart van Venrooij (TNO)
- Evert Kruis
- Miranda van Otichem
- Jordy van Domselaar (DIJ)

## 1.1. Versiebeheer

Versie	Beschrijving	Datum	Aanpassing
1.0	Initieel document	15-9-2018	
1.1	Aanpassing	9-11-2018	Aanpassingen gemaakt op basis van feedback
1.2	Aanpassing	15-4-2019	Update naar Laravel 5.8 Toevoeging van Docker + Lando Toevoeging voor html forms zonder collective
1.3	Aanpassing	3-9-2019	Update naar Laravel 6.0
1.4	Aanpassing	14-9-2019	Toevoegen van Laravel/ui vanaf begin
1.5	Aanpassing	31-10-2019	Aanpassingen voor gebruik named routes. Kleine stukjes toegevoegd voor duidelijkheid
2.0	Geheel aangepast	8-5-2020	Workshop basis en intermediate nu samengevoegd, waarin in 1 keer het meteen goed staat. Update naar Laravel 7 Geen Docker meer i.v.m. moeilijkheidsgraad en problemen met unit testing
2.1	Uitbreiding	29-5-2020	Onderdeel toegevoegd met unit & feature testing
2.2	Uitbreiding	22-6-2020	Onderdeel toegevoegd met test driven development
3.0	Geheel aangepast	25-9-2020	Update naar Laravel 8.0. Ontwikkelomgeving in bijlage ivm verschillende mogelijkheden.
3.1	Aanpassing	8-2-2021	Update naar Laravel 8 zonder Jetstream
3.2	Grote aanpassing	8-3-2021	Verschillende installaties erbij gezet. Standaard omgeving met wamp.net ivm nieuwste versies van nginx, php en mariadb. Installatie verplaatst naar bijlage zodat je zelf kan kiezen welke installatie je gebruikt.
3.3	Aanpassing	5-4-2021	Hoofdstuknummers toegevoegd Opdrachten toegevoegd. Permissies in eigen lay-out veranderd. Eerst alleen Blade, later Components.
3.4	Aanpassing	23-4-2021	Checklists weggehaald Mogelijk uitbreidingen weggehaald, komt er pas in als het klaar is. Wishlist toegevoegd
3.5	Uitbreiding	6-5-2021	Voor opdrachten link naar repository met tests toegevoegd. Per opdracht staat bij welke test uitgevoerd kan worden
3.6	Aanpassing	13-5-2021	Opdracht 1 t/m 20 gewijzigd met de testopdracht erbij.

## 1.2. Wishlist

Hieronder onderdelen die nog niet in de lessenserie staan, maar wel een wens is om er nog in te verwerken.

1	Per validatie aparte message
2	Prefix voor resource controllers
3	Pagination
4	Permissies in eigen layout
5	Components
6	API calls
7	Installatie op een mac
8	Redis
9	Browser tests with Dusk

### 1.3. Omgeving & Installatie

Omdat je op zoveel verschillende manieren je omgeving kan hebben, heb ik dit nu niet hier staan. In de bijlage kan je allerlei verschillende manieren vinden hoe je Laravel kan krijgen op verschillende omgevingen.

Omgevingen die in de bijlage staan zijn:

- [Wamp.net, met Nginx, Php 8 en MariaDB](#)
- [Laragon met Laravel installer](#)
- [Laragon met quick install](#)

Je moet zelf kijken wat je prettig vindt om te gebruiken. Zit je op een Mac, kan je bijvoorbeeld Valet gebruiken (<https://laravel.com/docs/8.x/valet#introduction>)

Op de website van Laravel staat ook nog een andere installatie, namelijk met Docker en WSL2.

(<https://laravel.com/docs/8.x/installation#getting-started-on-windows>)

Gebruik de omgeving met docker alleen als je echt je werkomgeving in een container wilt hebben. De installatie hiervan is niet zo simpel als het lijkt, dus voor studenten raad ik het aan om hiermee niet je eerste Laravel project te starten.

Mocht je een andere omgeving hebben, je kan altijd Laravel binnen halen met composer.

(<https://laravel.com/docs/8.x/installation#installation-via-composer>)

Ik verwacht dat de volgende programma's al geïnstalleerd zijn:

Editor:

- PhpStorm: <https://www.jetbrains.com/phpstorm/>
- VS Code: <https://code.visualstudio.com/>

Versiebeheer:

- Git: <https://git-scm.com/>
- Gitkraken: <https://www.gitkraken.com/>

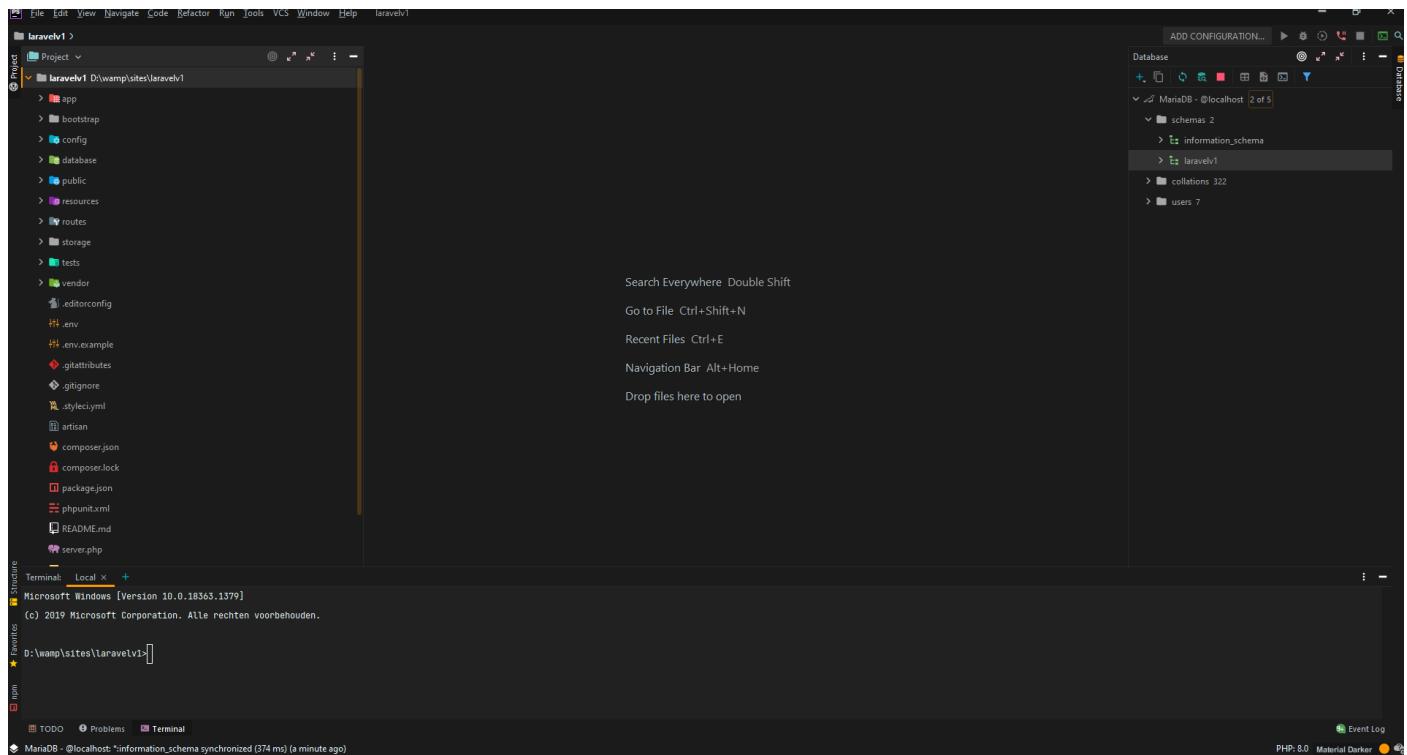
Andere tools:

- Composer: <https://getcomposer.org/>
- Npm: <https://nodejs.org/en/>

Zelf heb ik gekozen voor de omgeving met:

- Wamp.net
- PhpStorm
  - Material Theme UI
  - Material Icons
  - Laravel-idea ( <https://laravel-idea.com/> )
  - Database Tools en SQL

De editor ziet er dan als volgt uit:



Op deze manier zie je wat er gebeurt met mappen en bestanden, de database is zichtbaar in de editor en de terminal staat klaar voor gebruik.

## 1.4. Casus beschrijving

Als we naar de lessenserie verder gaan kijken, zullen we iets moeten gaan maken. Dan is het handig om een casus/opdracht te hebben. Bij deze de beschrijving.

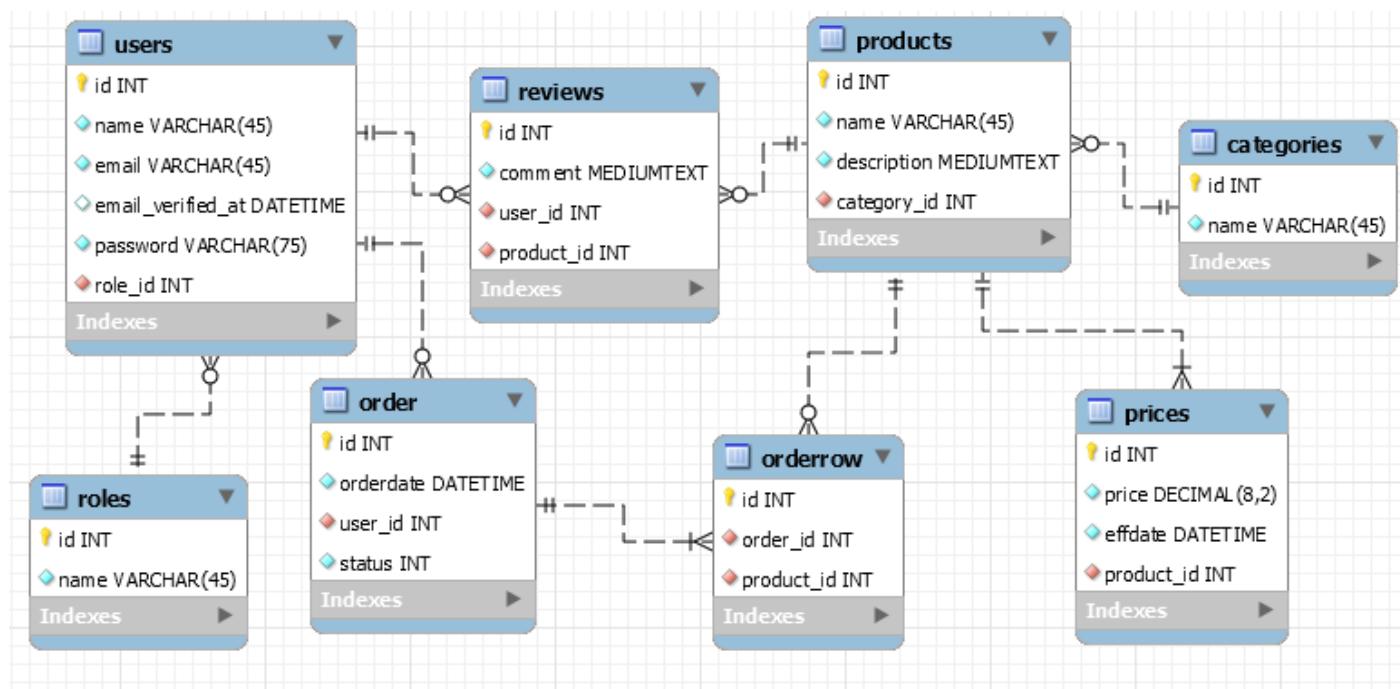
We gaan een simpele webshop maken voor spellen. Hierbij kunnen gebruikers:

- Spellen bekijken
- Details van een spel bekijken
- Review achterlaten over een spel
- Een spel kopen

Een beheerder zal alles moeten kunnen toevoegen, wijzigen en verwijderen. Hiervoor zijn de volgende admins:

- Categorie admin
- Product admin
- Price admin
- Order admin
- Review admin
- User admin

De volgende opzet van een concept database is gemaakt om dit voor elkaar te krijgen.



Om ervoor te zorgen dat je alle principes geleerd krijgt gaan we de website stap voor stap maken. De users worden door Laravel al voor een gedeelte geregeld. Daarnaast gaan we gebruik maken van een package die de rollen en permissions gaat regelen.

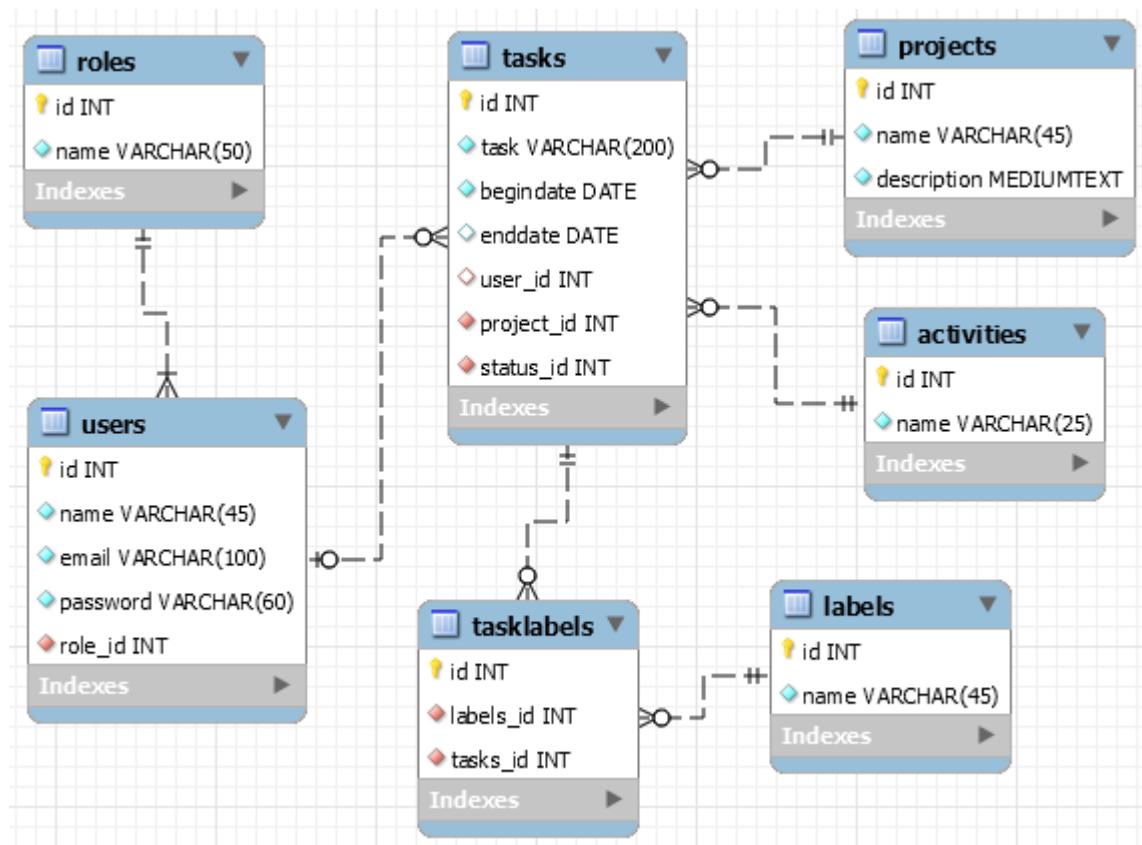
## 1.5. Opdrachten

Naast het project wat stap voor stap zal worden uitgelegd, is het ook handig dat je zelf met een klein projecten kan oefenen dat niet wordt voorgedaan in het document. Laravel leer je niet door middel van alleen kijken, je zal het echt zelf moeten uitvoeren en ervaren. Voor dit zal er dan ook tussen alle informatie oefenopdrachten zijn, waarbij de opdrachten gekoppeld zijn met wat je net hebt geleerd.

Voor de opdracht zal je aan de slag gaan met een project tool. Dit is om bij te houden hoeveer een project staat, welke taken er zijn en wie de taken moeten uitvoeren. De functionaliteit bestaat uit het volgende:

- Studenten kunnen een project aanmaken
- Bij een project kunnen allerlei taken horen
- Een taak heeft een status van een activity, bijvoorbeeld: Todo, Doing, Testing, Verify, Done
- Een taak kan verschillende labels hebben, zoals: front-end, backend, documentation, bug, feature

En ja, het ziet er misschien lastig uit, maar de opdrachten zullen steeds een klein stapje zijn. De opdrachten zullen ook heel precies zijn, er zijn namelijk automatische testen beschikbaar waar op alle details wordt gelet.



Om ervoor te zorgen dat je de automatische tests kan uitvoeren, kan je de repository clonen waarin de tests staan. Deze kan je vinden op:

- <https://gitlab.com/koningstein/laravel-8-opdrachten>

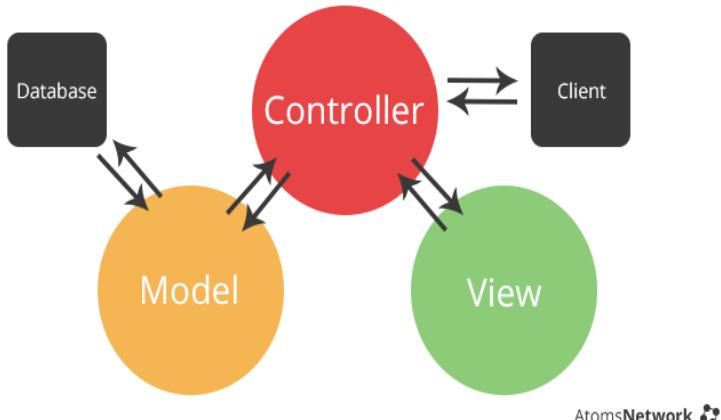
Voor nu nog geen opdracht verder, want je zal eerst wat moeten leren over Laravel voordat je een opdracht uit kan voeren.

## 2. Categories

We beginnen met de categorie. Dit is een van de makkelijkste gedeeltes, omdat er geen foreign key in deze tabel zit. Om hier mee te beginnen zal je eerst een klein stukje van het design pattern gaan leren.

### 2.1. Model

Laravel gebruikt het mvc principe. Hierbij is er een verdeling van verantwoordelijkheden binnen het systeem.



De Model zorgt dat alles met Data geregeld wordt. Let erop dat dit niet standaard alleen met een database is. In het begin zullen we het wel daar veel voor gebruiken, maar onthoud dat het voor alle data verantwoordelijk is.

Omdat Laravel nog een principe gebruikt, namelijk het ORM-principe. Als we gaan kijken wat Laravel hierover zegt vinden we dit:

### # Introduction

The Eloquent ORM included with Laravel provides a beautiful, simple ActiveRecord implementation for working with your database. Each database table has a corresponding "Model" which is used to interact with that table. Models allow you to query for data in your tables, as well as insert new records into the table.

Wat hier dus staat is dat voor elke tabel in de database een Model verantwoordelijk is. Deze gebruiken we dan ook om met de tabel te communiceren.

Nu heb je waarschijnlijk bij database lessen geleerd dat tabelnamen standaard enkelvoud zijn. In Laravel zijn regels die hier juist van afwijken.

Omdat de models dusdanig belangrijk zijn, worden deze in enkelvoud gezet. Om niet met de models en tabellen in verwarring te raken, zal elke tabel in meervoud in de database moeten staan.

Dit lijstje regels zal steeds iets verder uitgebreid worden zodat je elke functionaliteit die standaard in Laravel zit kan gebruiken.

- Model in enkelvoud (UpperCamelCase): voorbeeld = Student
- Tabel in meervoud (lowercase): voorbeeld = students

Als je straks de tabel en model hebt gemaakt zal je het langzamerhand gaan begrijpen. Het is belangrijk dat je deze Laravel conventies volgt om alle ingebouwde onderdelen te gebruiken.

Het aanmaken van de model is eigenlijk heel simpel. We gaan hiervoor Artisan gebruiken, wat veel gebruikt wordt binnen Laravel. Als je in de terminal intypt '`php artisan`' zal je ook zien dat je heel veel opties hebt. Een van die opties in het maken van een model.

Je kan altijd de help functie gebruiken om meer informatie te krijgen wat je allemaal met een commando kan. Hieronder zie je bijvoorbeeld wat allemaal mogelijk is als je `php artisan make:model` wilt gebruiken.

```
D:\wamp\sites\laravelv1>php artisan help make:model

Description:
Create a new Eloquent model class

Usage:
make:model [options] [--] <name>

Arguments:
name           The name of the class

Options:
-a, --all      Generate a migration, seeder, factory, and resource controller for the model
-c, --controller Create a new controller for the model
-f, --factory   Create a new factory for the model
--force         Create the class even if the model already exists
-m, --migration Create a new migration file for the model
-s, --seed       Create a new seeder file for the model
-p, --pivot     Indicates if the generated model should be a custom intermediate table model
-r, --resource  Indicates if the generated controller should be a resource controller
--api           Indicates if the generated controller should be an API controller
-h, --help       Display help for the given command. When no command is given display help for the list command
```

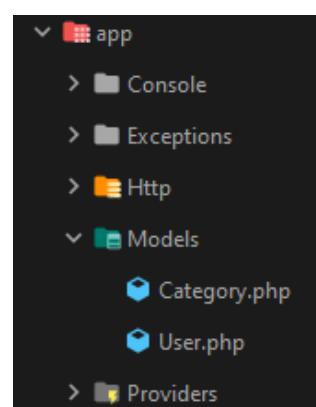
Nu kan je de makkelijkste optie gebruiken, namelijk de `--all` optie erbij zetten. Deze zorgt dat je alles krijgt wat je nodig hebt voor een CRUD. Nu in het begin gebruiken we deze nog even niet om juist even alle onderdelen los te leren, maar later in de lessenserie, bij de andere tabellen, gaan we deze optie wel gebruiken.

Voor nu gaan we dus eerst alleen de model maken. Dit doen we met: `php artisan make:model Category`

```
D:\wamp\sites\laravelv1>php artisan make:model Category
Model created successfully.

D:\wamp\sites\laravelv1>
```

Het bestand `Category.php` is gemaakt in de map `app/Models`. In deze map komen standaard alle models terecht. We hoeven voor nu nog even niks in de model te doen.



## 2.2. Migration

Laravel heeft, zoals vele frameworks, een onderdeel om migrations te maken. Een migration is voor het aanmaken van je database, maar dan in code. Maar waarom zou je dit doen?

Heel simpel, stel je voor dat je in een team van 5 mensen werkt. Jij update een tabel. Hoe krijgen de andere teamleden deze update? Ga je steeds een SQL-dump maken?

Hiervoor hebben ze dus een systeem bedacht dat je door middel van code een tabel kan aanmaken, maar ook weer kan aanpassen en verwijderen. Dit zodat je door middel van Git gewoon versiebeheer kan toepassen.

Meer info kan je hier vinden: <https://laravel.com/docs/8.x/migrations>

We hebben nu de commando `php artisan make:migration` nodig. Met de help gaan we kijken wat er verder nog mee kan.

```
D:\wamp\sites\laravelv1>php artisan help make:migration

Description:
Create a new migration file

Usage:
make:migration [options] [--] <name>

Arguments:
name          The name of the migration

Options:
--create[=CREATE]  The table to be created
--table[=TABLE]    The table to migrate
--path[=PATH]      The location where the migration file should be created
--realpath        Indicate any provided migration file paths are pre-resolved absolute paths
--fullpath        Output the full path of the migration
```

Als je voor het eerst kijkt is dit misschien nog een beetje verwarring. Maar op zich valt het best mee als je even weet hoe het werkt. Kijk eerst naar het onderdeel Usage.

`Make:migration [options] [--] <name>`

Dit betekent dus dat we eerst `make:migration` moeten hebben, daarna opties mee kunnen geven en op het einde de naam van de migration. Bij de migration die we willen maken willen we een tabel aanmaken. Bij de opties staat dan dat we `--create` mee moeten geven. Hoe het commando dan uiteindelijk eruitziet is zo:

`Php artisan make:migration --create=categories create_categories_table`

```
D:\wamp\sites\laravelv1>php artisan make:migration --create=categories create_categories_table
Created Migration: 2021_03_12_155613_create_categories_table
```

Bij dit commando maken we dus een migration aan, waar de tabel categories gaat heten.

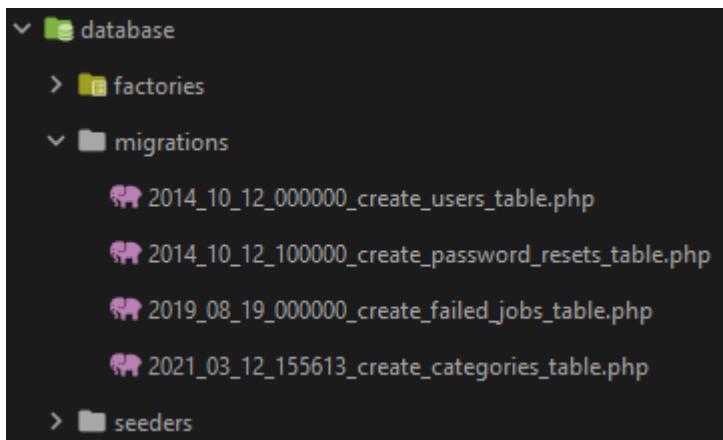
Omdat we met een migration niet alleen een tabel kunnen aanmaken maar ook veranderen, is het handig om de

naam van de migration duidelijk te houden. Hierdoor de benaming `create_categories_table`. (Let je op, de tabel is meervoud, dus de benaming van de migration is ook meervoud bij `categories`)

In Laravel 8 zit ook dat je de `--create=categories` kan weglaten, maar dan moet de naam wel duidelijk genoeg zijn.

Laravel will use the name of the migration to attempt to guess the name of the table and whether or not the migration will be creating a new table. If Laravel is able to determine the table name from the migration name, Laravel will pre-fill the generated migration file with the specified table. Otherwise, you may simply specify the table in the migration file manually.

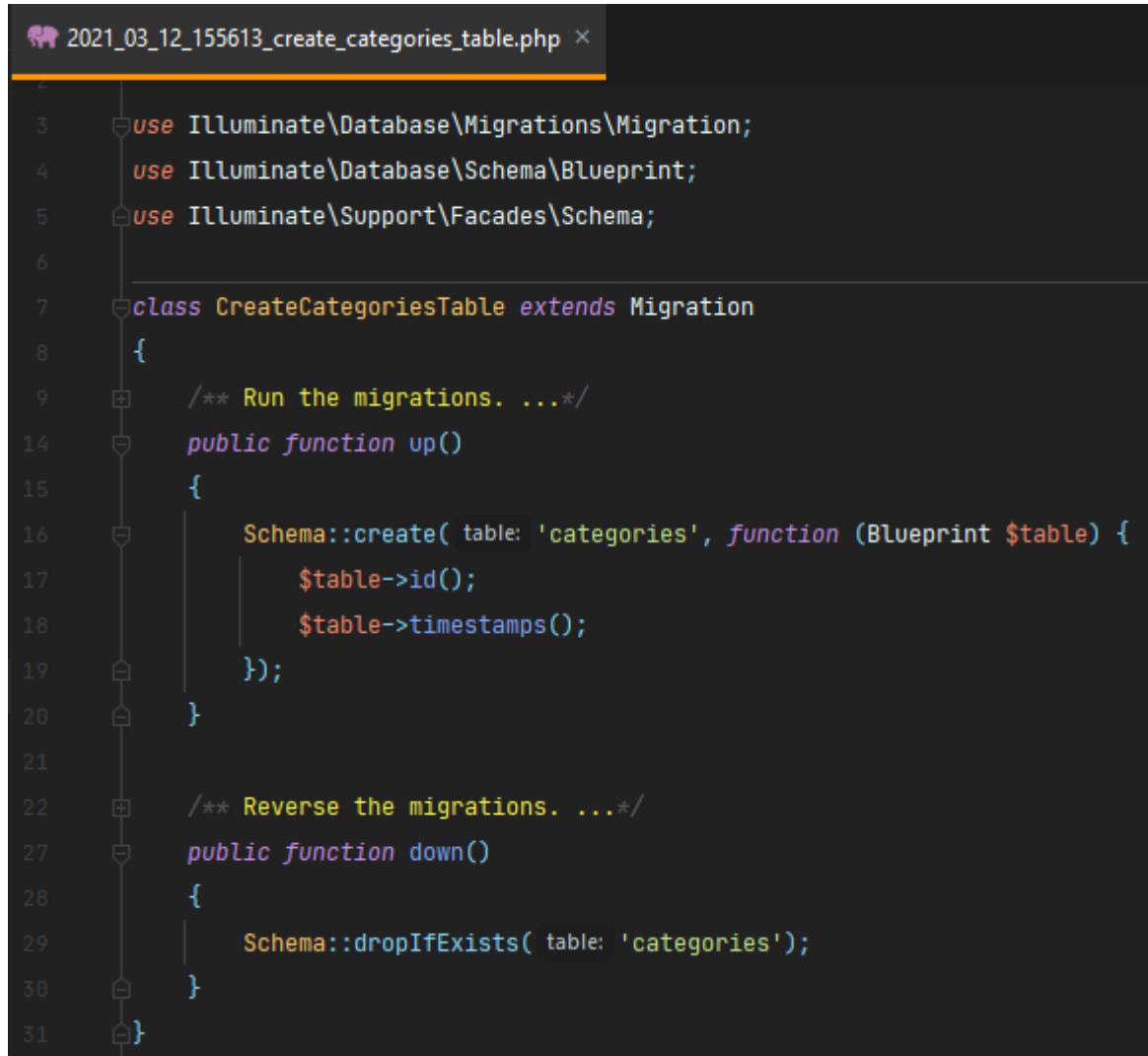
Toch gebruiken we de `--create` wel, want dan weet je zeker dat het goed gaat.



Nu we dit commando hebben uitgevoerd is er een bestand aangemaakt voor de migration. Deze vind je in de map van database/migrations.

Zoals je ziet hebben we ook nog 3 andere migrations. Deze worden meegeleverd door Laravel. Later gaan we deze nog nodig hebben als we met Laravel Auth aan de slag gaan.

Laten we kijken naar de categories migration.



```
2021_03_12_155613_create_categories_table.php ×

1
2
3     use Illuminate\Database\Migrations\Migration;
4
5     use Illuminate\Database\Schema\Blueprint;
6
7     use Illuminate\Support\Facades\Schema;
8
9
10    class CreateCategoriesTable extends Migration
11    {
12
13        /**
14         * Run the migrations.
15         */
16
17        public function up()
18        {
19
20            Schema::create('categories', function (Blueprint $table) {
21
22                $table->id();
23
24                $table->timestamps();
25            });
26
27        }
28
29        /**
30         * Reverse the migrations.
31         */
32
33        public function down()
34        {
35
36            Schema::dropIfExists('categories');
37        }
38    }

```

Je maakt gebruik van een aantal classes die in Laravel zijn ingebouwd, dat zie je op regel 3 t/m 5.

De class die je hebt aangemaakt heet CreateCategoriesTable, wat een uitbreiding is op de class Migration. Er staan 2 methodes in de class, een methode up() en een methode down(). Dit zijn de methodes die ervoor zorgen dat de tabel wordt aangemaakt in de database of wordt verwijderd.

Je ziet ook al netjes dat het om de tabel **categories** gaat (meervoud!)

In de up methode staat al een klein stukje waaronder de id en timestamps.

Let op dat de primary key in Laravel altijd id is. Ga dit niet in de tabel veranderen, anders ga je later in de problemen raken. Een foreign key zal je later tegenkomen, maar je zal dan zien dat deze ook een standaard format heeft voor de naam, namelijk tabelnaam\_id

De methode up heb ik aangepast om de attributen binnen de tabel te benoemen. Volgens de ERD hebben we de attribuut name nodig.

```
public function up()
{
    Schema::create('categories', function (Blueprint $table) {
        $table->id();
        $table->string('name', 100);
        $table->timestamps();
    });
}
```

Ik krijg hierdoor straks in mijn tabel:

- Een primary key: id (int)
- name (varchar(100))
- de timestamps, die ervoor zorgen dat ik de attributen created\_at en updated\_at krijg, wat allebei timestamps zijn. (datum+tijd)

De down methode hoeven we niet te veranderen. Als we een tabel willen verwijderen is alleen de tabelnaam voldoende.

De migration kunnen we nu uitvoeren met *php artisan migrate*

```
D:\Wamp\sites\laravelv1>php artisan migrate
Migration table created successfully.

Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (17.28ms)

Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (14.79ms)

Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (16.75ms)

Migrating: 2021_03_12_195928_create_categories_table
Migrated: 2021_03_12_195928_create_categories_table (7.81ms)
```

Er worden 4 tabellen aangemaakt, users, password reset, failed\_jobs en categories. Dit klopt want dit zijn ook de migrations die in de map staan.

Voor de zekerheid even controleren in de database:

Tabel	Actie	Rijen	Type
categories	Verkennen  Structuur  Zoeken  Invoegen  Legen  Verwijderen	0	InnoDB
failed_jobs	Verkennen  Structuur  Zoeken  Invoegen  Legen  Verwijderen	0	InnoDB
migrations	Verkennen  Structuur  Zoeken  Invoegen  Legen  Verwijderen	4	InnoDB
password_resets	Verkennen  Structuur  Zoeken  Invoegen  Legen  Verwijderen	0	InnoDB
users	Verkennen  Structuur  Zoeken  Invoegen  Legen  Verwijderen	0	InnoDB
5 tabellen	Som	4	InnoDB

Laravel zorgt zelf voor het bijhouden van migrations. Dit doen ze in de migrations tabel. Als je naar de inhoud kijkt zal je dit ook zien. Ga je later aanpassingen maken zal de batch gaan veranderen.

		id	migration	batch
	Wijzigen	Kopiëren	Verwijderen	1
	Wijzigen	Kopiëren	Verwijderen	1
	Wijzigen	Kopiëren	Verwijderen	1
	Wijzigen	Kopiëren	Verwijderen	1

Je kan altijd ook checken in PhpStorm met de database tool. Configuratie ervan vind je in de [bijlage](#). Hiermee kan je zien dat wat we net hebben gezien ook daarin staat.

Nu denk je vast, waarom gebruiken we nu migrations en hebben we die van categories genoemd met een create. Laten we nu een update doen, dat de name niet 100 characters maar 75 characters lang mag zijn.

Als eerst gaan we een nieuwe migration aanmaken. (Niet de oude gaan bewerken!!)  
Wat hebben we nu nodig? Hiervoor eerst de help gebruiken: `php artisan help make:migration`

```
D:\wamp\sites\laravelv1>php artisan help make:migration

Description:
  Create a new migration file


Usage:
  make:migration [options] [--] <name>

Arguments:
  name          The name of the migration


Options:
  --create[=CREATE]  The table to be created
  --table[=TABLE]    The table to migrate
```

Met `--table` gaan we geen tabel maken, maar een tabel wijzigen. Deze hebben we dus nodig. Het commando wordt dan ook: `php artisan make:migration --table=categories updateNameLength_categories_table`

```
D:\wamp\sites\laravelv1>php artisan make:migration --table=categories updateNameLength_categories_table
Created Migration: 2021_03_15_132625_update_name_length_categories_table
```

Je zal zien, dat automatisch meerdere underscores komen vanwege het hoofdlettergebruik.

Het resultaat zie je hieronder. Ik heb alvast regel 17 en 29 erin zelf ingezet.

```
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  class UpdateNameLengthCategoriesTable extends Migration
8  {
9      /**
10     * Run the migrations.
11     */
12    public function up()
13    {
14        Schema::table('categories', function (Blueprint $table) {
15            $table->string('name', 75)->change();
16        });
17    }
18
19
20
21    /**
22     * Reverse the migrations.
23     */
24    public function down()
25    {
26        Schema::table('categories', function (Blueprint $table) {
27            $table->string('name', 100)->change();
28        });
29    }
30
31
32 }
```

Bij de up methode moeten we de wijziging van 100 (1<sup>e</sup> migration) naar 75 zetten. Bij de downmethode moeten we weer terug kunnen, dus naar 100. Nu denk je dat we klaar zijn om het door te voeren. Laten we het controleren. We voeren de migration uit met `php artisan migrate`

```
D:\wamp\sites\laravelv1>php artisan migrate
Migrating: 2021_03_15_132625_update_name_length_categories_table

 RuntimeException

Changing columns for table "categories" requires Doctrine DBAL. Please install the doctrine/dbal package.
```

Helaas werkt het nog niet. We hebben doctrine/dbal package nodig.

Een package installeren we altijd door middel van de composer.json, die in de root map staat van het project. Hierin gaan we dan de doctrine package zetten. In de editor kan je zien, dat versie 4.0.x-dev bestaat als laatste versie. Die kunnen we alleen niet gebruiken, omdat deze nog niet compatible is. De laatste versie is dan 3.1.\* die we wel kunnen gebruiken.

```
"require-dev": {
    "facade/ignition": "^2.5", 2.5.14
    "fakerphp/faker": "^1.9.1", v1.13.0
    "laravel/sail": "^1.0.1", v1.4.7
    "mockery/mockery": "^1.4.2", 1.4.3
    "nunomaduro/collision": "^5.0", v5.3.0
    "phpunit/phpunit": "^9.3.3", 9.5.2
    "doctrine/dbal": "3.1.*" 3.1.x-dev
},
```

Als je dan composer update gebruikt, zie je dat hij doctrine gaat installeren.

```
D:\Wamp\sites\laravelv1>composer update
Loading composer repositories with package information
Updating dependencies
Lock file operations: 4 installs, 19 updates, 0 removals
```

Als de installatie klaar is, kunnen we de migration uitvoeren.

```
D:\wamp\sites\laravelv1>php artisan migrate
Migrating: 2021_03_15_132625_update_name_length_categories_table
Migrated: 2021_03_15_132625_update_name_length_categories_table (490.66ms)
```

Als we in phpMyAdmin kijken bij de tabel categories zien we dat de name 75 characters is

#	Naam	Type	C	id	migration	batch
1	<b>id</b> 	bigint(20)		1	2014_10_12_000000_create_users_table	1
2	<b>name</b>	varchar(75)	u	2	2014_10_12_100000_create_password_resets_table	1
3	<b>created_at</b>	timestamp		3	2019_08_19_000000_create_failed_jobs_table	1
4	<b>updated_at</b>	timestamp		4	2021_03_12_155613_create_categories_table	1
				5	2021_03_15_132625_update_name_length_categories_ta...	2

Verder zie je in de migration tabel dat de update erbij is gekomen. Je ziet dan ook dat bij batch er een 2 staat. Dit betekent dat bij de eerste migration nr 1 t/m 4 zijn gemigrated, en bij de 2e migration alleen nr 5.

De database structuur staat er nu. Het volgende onderdeel zal over het vullen van de database gaan.

## 2.3. Opdracht 1: model en migration

Maak een model en migration voor de tabel projects:

projects	
!	id INT
◆	name VARCHAR(45)
◆	description MEDIUMTEXT
Indexes	

Zorg dat bij de migration de standaard id en timestamps gebruikt worden.

## 2.4. Factory

Om de tabel categories te vullen met data, gaan we gebruik maken van 3 onderdelen. Ieder met zijn eigen rol:

- Factory: Structuur van 1 insert
- Faker: genereren van data
- Seeder: hoeveelheid rijen genereren

De reden dat we ook de database al met data vullen is, zodat elk teamlid op dezelfde wijze testdata in de database kan krijgen. Je zal merken dat dit straks heel erg gemakkelijk gaat.

Als eerst gaan we de factory aanmaken met Artisan. Als we naar de opties kijken vind je dit.

```
D:\Wamp\sites\laravelv1>php artisan help make:factory

Description:
Create a new model factory

Usage:
make:factory [options] [--] <name>

Arguments:
name           The name of the class

Options:
-m, --model[=MODEL]  The name of the model
-h, --help          Display help for the given command. When no command is given display help for the list command
-q, --quiet         Do not output any message
-V, --version       Display this application version
```

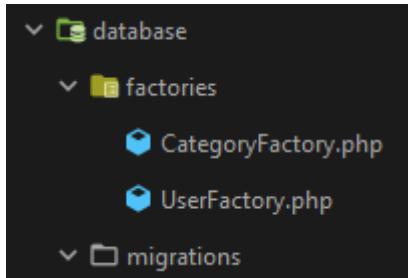
Als we de model meegeven bij het aanmaken van de factory zal er meteen een stuk extra code voor je gemaakt worden. Handig! Dit gaan we dan ook doen met `php artisan make:factory --model=Category CategoryFactory`

```
D:\Wamp\sites\laravelv1>php artisan make:factory --model=Category CategoryFactory
Factory created successfully.
```

Even de standaard notatie voor een Factory: ModelnaamFactory (UpperCamelCase). Dit voegen we toe aan de regels.

- Model in enkelvoud (UpperCamelCase): voorbeeld = Student
- Tabel in meervoud (lowercase): voorbeeld = students
- Factory in enkelvoud (UpperCamelCase): voorbeeld = StudentFactory

De factory die is aangemaakt vind je in de map database/factories:



Als we dan de CategoryFactory openen zien we dit.

```
<?php

namespace Database\Factories;

use App\Models\Category;
use Illuminate\Database\Eloquent\Factories\Factory;

class CategoryFactory extends Factory
{
    /**
     * The name of the factory's corresponding model. ...
     */
    protected $model = Category::class;

    /**
     * Define the model's default state. ...
     */
    public function definition()
    {
        return [
            //
        ];
    }
}
```

Mocht je al met een eerdere versie van Laravel gewerkt hebben, merk je dat dit anders is geworden.

Voor de factory gaan we gebruik maken van Faker. Dit zit al standaard in Laravel.

Op GitHub kan je de meest actuele documentatie over Faker vinden: <https://github.com/fzaninotto/Faker>

In de tabel products willen we een aantal attributen gaan vullen:

- name
- created\_at
- updated\_at

De return in de factory moet ingevuld worden met de velden die je wilt vullen.

```
public function definition()
{
    return [
        'name' => $this->faker->name
    ];
}
```

Het kan zijn dat je geen lijntje onder definition methode krijgt. Dit is een instelling van mijn editor dat bij methodes ook datatypes meegegeven moeten worden ivm de nieuwe php regels. Laravel heeft dit nog niet allemaal erin staan.

De created\_at en updated\_at worden automatisch gevuld. Je kan er dus voor kiezen om alleen de name te regelen. Alleen kies ik ervoor om hier datums wat specifieker te bepalen.

```
public function definition()
{
    return [
        'name' => $this->faker->name,
        'created_at' => $this->faker->dateTimeThisDecade( max: 'now', timezone: 'Europe/Amsterdam'),
        'updated_at' => $this->faker->dateTimeThisDecade( max: 'now', timezone: 'Europe/Amsterdam')
    ];
}
```

Alle onderdelen van Faker die ik heb gebruikt vind je in de documentatie hiervan. De structuur van 1 insert is nu bepaald.

## 2.5. Seeder

De volgende stap is om de seeder te maken. Dit doen we weer met Artisan.

```
D:\Wamp\sites\laravelv1>php artisan help make:seeder

Description:
  Create a new seeder class

Usage:
  make:seeder <name>

Arguments:
  name          The name of the class

Options:
  -h, --help      Display help for the given command. When no command is given display help for the list command
  -q, --quiet     Do not output any message
  -V, --version   Display this application version
  --ansi         Force ANSI output
```

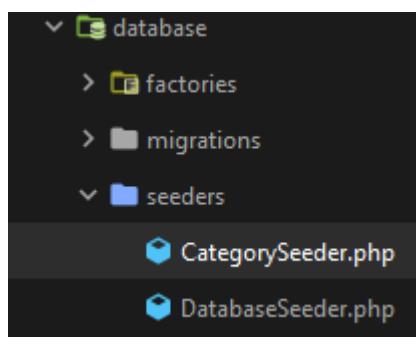
We kunnen nu dus niet echt wat extra's doen bij de seeder. We gaan dus de seeder voor Categorie nu maken.

```
D:\Wamp\sites\laravelv1>php artisan make:seeder CategorySeeder
Seeder created successfully.
```

Format van de naam van de seeder is TabelnaamSeeder. Dit voegen we dan ook toe aan de regels:

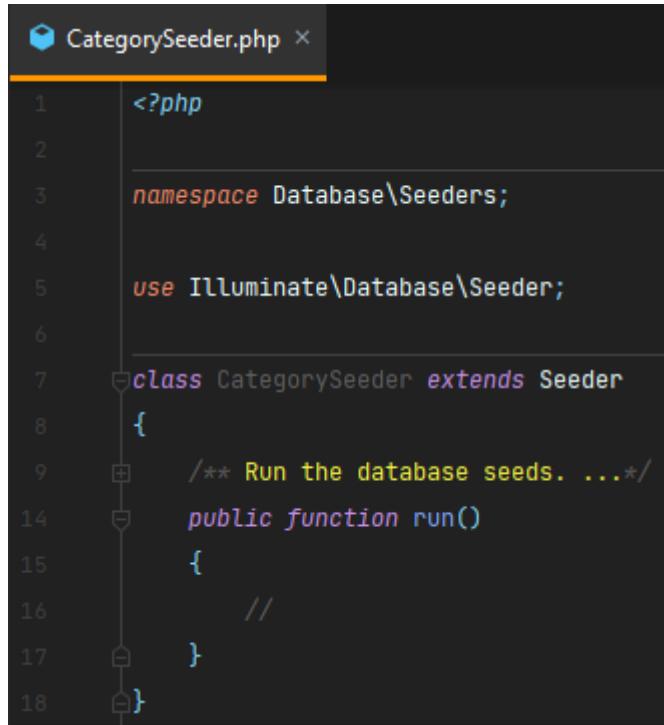
- Model in enkelvoud (UpperCamelCase): voorbeeld = Student
- Tabel in meervoud (lowercase): voorbeeld = students
- Factory in enkelvoud (UpperCamelCase): voorbeeld = StudentFactory
- Seeder in enkelvoud (UpperCamelCase): voorbeeld = StudentSeeder

De seeder kunnen we in de map database/seeders vinden



Als we dan gaan kijken naar de seeder zien we een class met 1 methode, de run methode.

Deze wordt uitgevoerd als deze seeder wordt aangeroepen.



```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6
7 class CategorySeeder extends Seeder
8 {
9     /**
10      * Run the database seeds.
11      */
12
13     public function run()
14     {
15         //
16     }
17 }
18 }
```

Om ervoor te zorgen dat we nu vulling krijgen in de database moeten we regel 17-19 toevoegen aan de code. Je geeft aan van welke model je de factory wil gebruiken, en hoeveel regels je in je tabel wilt hebben.

We gebruiken de Model van Category, dus op regel 5 is ook deze toegevoegd zodat de class CategorySeeder weet dat we deze gebruiken.



```
3 namespace Database\Seeders;
4
5 use App\Models\Category;
6 use Illuminate\Database\Seeder;
7
8 class CategorySeeder extends Seeder
9 {
10     /**
11      * Run the database seeds.
12      *
13      * @return void
14      */
15     public function run()
16     {
17         Category::factory()
18             ->times(3)
19             ->create();
20     }
21 }
```

Als laatst gaan we dan de CategorySeeder toevoegen in de DatabaseSeeder.

De DatabaseSeeder wordt door `php artisan db:seed` gestart.

Elke seed die in dit bestand wordt opgeroepen wordt uitgevoerd in de volgorde die in het bestand staat. Er staat een voorbeeld in hoe je dit doet. We voegen dan ook de seeder toe van de CategorySeeder op regel 17. We maken er alvast een array van, omdat we later andere seeders dan makkelijker kunnen toevoegen.

```
3  namespace Database\Seeders;
4
5  use Illuminate\Database\Seeder;
6
7  class DatabaseSeeder extends Seeder
8  {
9      /** Seed the application's database. ... */
10     public function run()
11     {
12         $this->call([
13             CategorySeeder::class
14         ]);
15     }
16 }
17
18 }
```

Nu kunnen we de tabel eindelijk vullen met `php artisan db:seed`

```
D:\Wamp\sites\laravelv1>php artisan db:seed
Seeding: Database\Seeders\CategorySeeder
Seeded: Database\Seeders\CategorySeeder (31.99ms)
Database seeding completed successfully.
```

Als we dan naar het resultaat gaan kijken in de phpMyAdmin zien we dit

<b>id</b>	<b>name</b>	<b>created_at</b>	<b>updated_at</b>
1	Mrs. Nina McLaughlin	2014-01-06 18:41:48	2017-01-17 11:19:25
2	Isabel Shields	2011-12-19 10:22:00	2018-05-05 13:28:14
3	Dr. Jessika Torp	2019-12-23 17:54:26	2015-06-02 05:25:12

De database is nu netjes gevuld. Ok, misschien zijn de namen van de categorieën niet zoals je het zou willen, maar dit is gewoon testdata.

Let er even op dat het proces iets lastiger wordt als je straks meerdere tabellen in je database wilt met relaties ertussen. Dit omdat de primary key – foreign key relaties wel moeten kloppen. Dit zal later worden besproken.

## 2.6. Opdracht 2: factory & seed

Zorg dat de tabel projects gevuld wordt met data, door middel van een factory en een seeder. Bij opdracht 1 heb je de migration gemaakt voor deze tabel.

projects	
!	id INT
◆	name VARCHAR(45)
◆	description MEDIUMTEXT
Indexes	

Daarnaast heb je timestamps erbij gedaan. De timestamps worden automatisch gevuld als je deze niet invult in je factory. Gebruik in de factory faker om random data erin te zetten.

De seeder die zal de volgende naam moeten hebben:

- ProjectSeeder

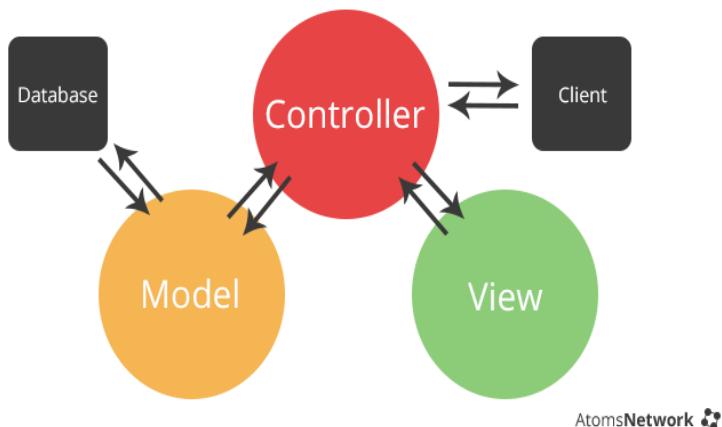
### Opdracht controle

Als je de opdracht af hebt en gebruik hebt gemaakt van de opdrachten repository, kan je het volgende commando gebruiken om te controleren of alles goed werkt:

```
.\vendor\bin\pest --group=Opdracht1_2
```

## 2.7. Lay-out met Breeze

Voordat we beginnen met het programmeren, is het makkelijk om alvast de basis lay-out te hebben. Hiervoor gebruiken we in Laravel een masterpage. Dit betekent dat alles wat op elke pagina steeds hetzelfde is in de masterpage staat, en wat er verandert in de subpage.



Als we het over de lay-out hebben gaan we het Views hebben. Views zijn verantwoordelijk het uiterlijk van de site, dus eigenlijk gewoon de html.

Bij de oudere versies van Laravel had je een ui package waarmee je bootstrap kon inladen. In Laravel 8 hebben ze allerlei vernieuwingen aangebracht voor de basis layout. Dit is zo uitgebreid geworden, dat je allerlei opties hebt. De package die we gaan gebruiken is Laravel Breeze. (<https://laravel.com/docs/8.x/starter-kits#laravel-breeze>)

Als eerst gaan we ervoor zorgen dat we laravel/breeze binnenhalen met composer met: `composer require laravel/breeze --dev`

```
D:\Wamp\sites\laravelv1>composer require laravel/breeze --dev
Using version ^1.1 for laravel/breeze
./composer.json has been updated
Running composer update laravel/breeze
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
- Locking laravel/breeze (v1.1.4)

Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
- Downloading laravel/breeze (v1.1.4)
- Installing laravel/breeze (v1.1.4): Extracting archive
Generating optimized autoload files
composer/package-versions-deprecated: Generating version class...
composer/package-versions-deprecated: ...done generating version class
```

In Laravel Breeze wordt er gebruik gemaakt van Blade templates en Tailwind CSS. Alle simpele onderdelen voor authenticatie zitten erin, zoals login, registratie, password reset en wachtwoord verificatie.

Om breeze te installeren gebruiken we: `php artisan breeze:install`

```
D:\Wamp\sites\laravelv1>php artisan breeze:install
Breeze scaffolding installed successfully.

Please execute the "npm install && npm run dev" command to build your assets.
```

Je ziet dat we npm moeten installeren en runnen om het scaffolden te voltooien. Dus eerst: `Npm install && npm run dev`

```
D:\Wamp\sites\laravelv1>npm install && npm run dev
npm [WARN] deprecated fsevents@1.2.13: fsevents 1 will break on node v14+ and could be using insecure binaries. Upgrade to fsevents 2.
npm [WARN] deprecated resolve-url@0.2.1: https://github.com/lydell/resolve-url#deprecated
npm [WARN] deprecated urix@0.1.0: Please see https://github.com/lydell/urix#deprecated
npm [WARN] deprecated chokidar@2.1.8: Chokidar 2 will break on node v14+. Upgrade to chokidar 3 with 15x less dependencies.
[          ] \ reify:source-map: http fetch GET 200 https://registry.npmjs.org/source-map/-/source-map-0.6.1.tgz 9410ms
```

File	Size
/js/app.js	673 Kib
css/app.css	3.75 MiB

Dat er wat meldingen komen maakt voor ons nu niet uit. Pas als je wat online gaat zetten maakt dit uit. Waarschijnlijk heb ik een wat oude npm geïnstalleerd staan. Het resultaat is dat we allerlei bestanden erbij hebben gekregen. Dit zijn node modules.

```
> database
> node_modules library root
> public
```

We hebben een css bestand en een js bestand gekregen voor onze standaard layout. Je ziet ook dat het is geïnstalleerd met Laravel Mix. Als je op de site van Tailwind kijken, kan je daar ook lezen hoe het met de configuratie zit.

Omdat we Laravel Mix al hebben gebruikt hoeft je de volgende stap niet te doen. Wel even info waar de JS en CSS bestanden staan. De config staat namelijk in resources map, maar de werkelijke css en js bestanden staan in de public map.

## Configure Tailwind with Laravel Mix

In your `webpack.mix.js`, add `tailwindcss` as a PostCSS plugin:

```
// webpack.mix.js
mix.js("resources/js/app.js", "public/js")
    .postCss("resources/css/app.css", "public/css", [
+        require("tailwindcss"),
    ]);
```

Je kan verder nog zien welke componenten zijn geïnstalleerd. Volgens de website zijn deze onderdelen de basis

```
/* ./resources/css/app.css */
@tailwind base;
@tailwind components;
@tailwind utilities;
```

```
1  @import 'tailwindcss/base';
2  @import 'tailwindcss/components';
3  @import 'tailwindcss/utilities';
```

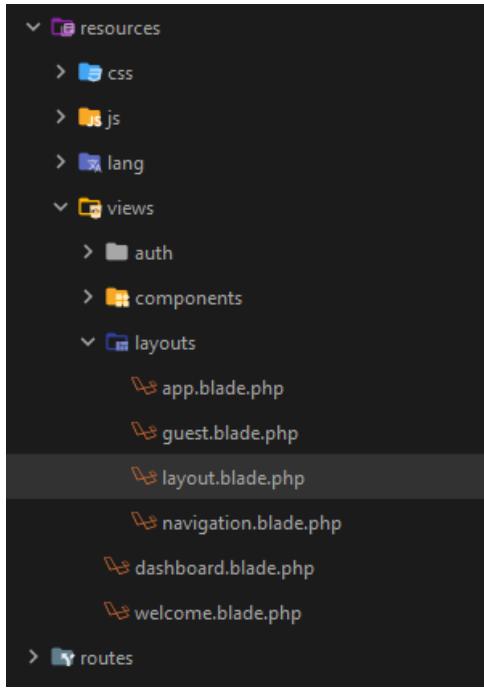
Als je dan kijkt in resources zie je een soortgelijke notatie, alleen dan geen @tailwind maar @import.

Nu wordt het tijd om onze lay-out te kiezen. Omdat we met Tailwind CSS gaan werken, heb ik een basis layout gezocht die zo gebruikt kan worden. (<https://tailwindcomponents.com/component/tailwind-users-dashboard>)

Name	Login	Project	Role	Status	Date Added
Carmen Beltran	Carmen.Bel	Aero Treck Search 2 More...	Designer	None In Work	Final Date 20.02.2020 07.02.2020 11:00
Enoshima Junko	Zetsbuo	Aero Treck	Front-End	In Work	Final Date 20.02.2020 11:00

Natuurlijk krijgen we een html pagina waarin gebruik wordt gemaakt van Tailwind. De layout moet nog geschikt worden gemaakt om binnen Laravel te gebruiken. Als je dan naar de broncode gaat en dat allemaal kopieert naar een nieuw bestand in het project: `resources/views/layouts/`

De map bestaat al, en er staat een `app.blade.php` in. Hier staat wat code in die we later in onze lay-out moeten gaan gebruiken, dus hierdoor kies ik ervoor om de masterpage die we gaan maken layout.blade.php te noemen.

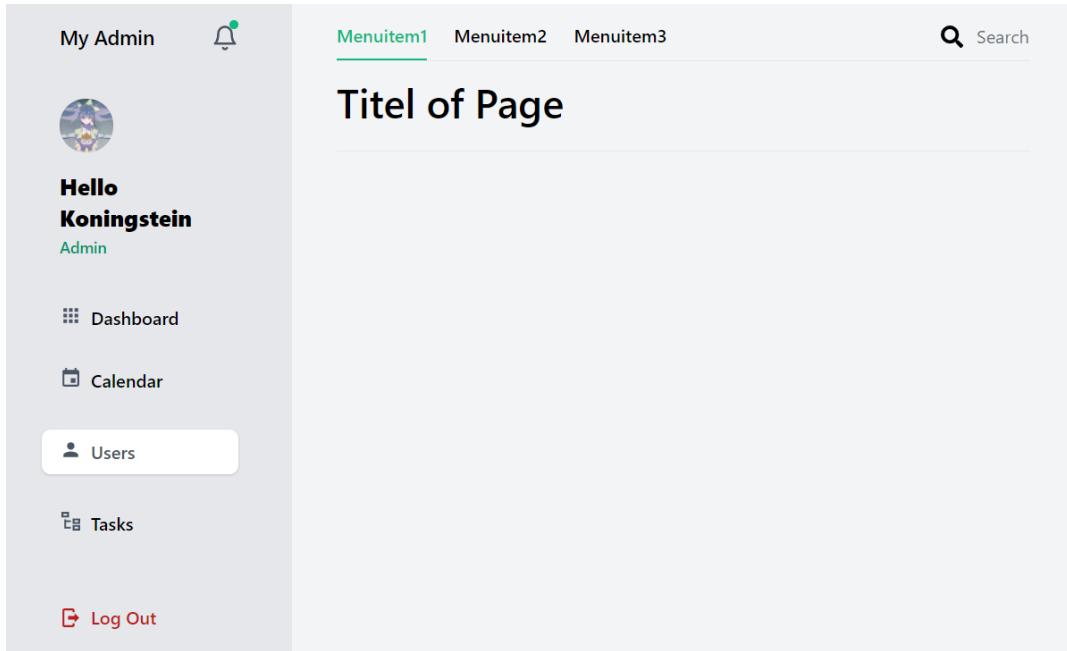


Als eerst gaan we de <head> aanpassen voor wat we nodig hebben. De inhoud van component.html heb ik in layout.blade.php gezet en de <head> aangepast zodat WindTail CSS gebruikt gaat worden.

```
layout.blade.php
1 <!doctype html>
2 <html lang="en">
3 <head>
4     <meta charset="utf-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1">
7     <meta name="csrf-token" content="{{ csrf_token() }}>
8     <title>{{ config('app.name', 'Laravel') }}</title>
9     <link href="{{ asset('css/app.css') }}" rel="stylesheet">
10
11    <!-- Door middel van defer wordt de javascript pas ingeladen als de pagina is ingeladen
12    https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script -->
13    <script src="{{ asset('js/app.js') }}" defer></script>
14 </head>
```

We gebruiken {{ }} om te zorgen dat de Blade template engine zijn werk gaat doen. De asset functie zorgt ervoor dat er steeds vanuit de map public wordt gekeken. Hierdoor zal de css steeds blijven werken. Doe je dit niet zal je merken dat als de url langer wordt vanwege dat je in mappen zit, je lay-out niet meer werkt. Let er even op dat de bootstrap.min.css in Laravel staat in app.css

De rest van de invulling heb ik voornamelijk onderdelen weggehaald. Dit kan je doen zoals je dat zelf wilt. Uiteindelijk kan je er zo iets van maken:



De content is weg, waardoor we daar straks de invulling aan kunnen geven. Aan de bovenkant komen menuitems van de actieve pagina. Om naar andere pagina's te gaan zal ik het menu aan de linkerkant gaan gebruiken.

De masterpage is bijna klaar. Alleen de plek waar onze content komt moet nog bepaald worden. Dit doen we binnen onze layout.blade redelijk op het eind van de code.

```
130      </nav>
131
132      <div
133          class="pb-2 flex items-center justify-between text-gray-600
134              dark:text-gray-400 border-b dark:border-gray-600">
135
136          @yield('content')
137
138      </div>
139      </div>
140      </main>
141      </div>
142      </body>
143      </html>
```

De @yield zorgt ervoor dat we straks de inhoud van de sub page op die plek kunnen zetten.

We zijn nu voorlopig klaar met de masterpage. Zodra we de sub pagina's gaan maken, die we nodig hebben bij de controller, zal je precies zien hoe we dit gaan gebruiken. Er zullen nog wel aanpassingen komen, maar die gaan we er later inzetten zodat je ook ziet waarvoor dat nodig is.

## 2.8. Opdracht 3: layout met Tailwind

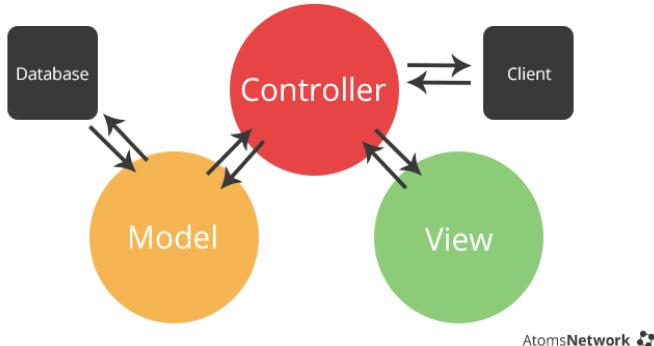
Maak een eigen masterpage op basis van Tailwind. Probeer hiervoor een template te gebruiken zodat het niet al te veel tijd kost om dit aan te passen.

# 3. Resource Controller

## 3.1. Aanmaken van de resource controller

In deze lessenserie gaan we, anders dan de meeste tutorials, niet geheel beginnen bij de basis, maar meteen kijken hoe je functionaliteiten kan gaan maken. Ondertussen zullen er allerlei basisprincipes voorbijkomen.

Om de tabel die we gemaakt hebben te kunnen bewerken, hebben we een CRUD nodig. Een crud staat voor Create Read Update Delete. Nu heeft Laravel iets handigs hiervoor en dat is een resource controller. Om even te kijken wat in een controller gebeurt kijken we even naar het MVC-principe.



AtomsNetwork

Zoals je op het plaatje kan zien werkt de controller met het model samen en kan vanuit de controller een view worden aangestuurd. Eigenlijk is dus de controller de regelneef die alles gaat aansturen.

Om te kijken hoe dan zo'n controller eruit ziet gaan we de resource controller maken.

Dit doen we met `php artisan help make:controller`

```
D:\Wamp\sites\laravelv1>php artisan help make:controller

Description:
Create a new controller class

Usage:
make:controller [options] [--] <name>

Arguments:
name          The name of the class

Options:
--api          Exclude the create and edit methods from the controller.
--force        Create the class even if the controller already exists
-i, --invokable Generate a single method, invokable controller class.
-m, --model[=MODEL] Generate a resource controller for the given model.
-p, --parent[=PARENT] Generate a nested resource controller class.
-r, --resource Generate a resource controller class.
-h, --help      Display help for the given command. When no command is given display help for the list command
```

Bij een controller zijn er heel veel opties zoals je ziet. Op dit moment hebben we een controller nodig waarmee we onze CRUD kunnen maken. Dit is een resource controller. Zonder opties krijg je een lege controller.

Nu kan je een resource controller maken met `--resource`, maar als je de model meegeeft maakt die juist voor die model een resource controller, wat dus voor ons de beste optie is.

Omdat we aan de slag gaan met een CRUD, wat een onderdeel van de admin wordt, is de bedoeling om straks alle crud's voor de admin netjes in een mapje te zetten. Hierdoor voegen we 'Admin/' voor de naam van de controller, zodat deze in de map komt.

Voor de optie met het meegeven van de model gebruiken we dus dit:

```
php artisan make:controller --model=Category Admin/CategoryController
```

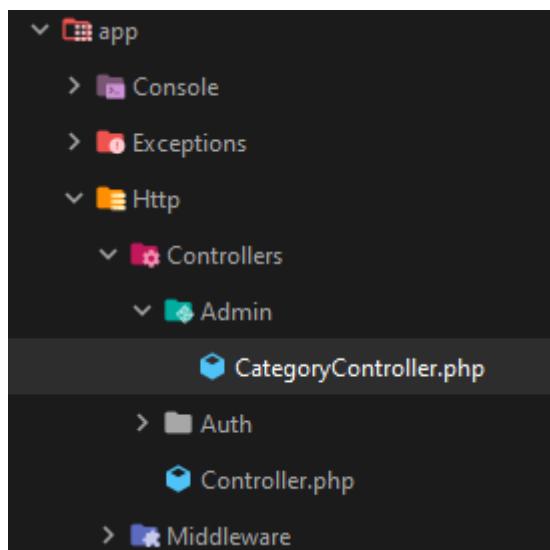
Format van de naam van de controller is ModelnaamController. Dit voegen we dan ook toe aan de regels:

- Model in enkelvoud (UpperCamelCase): voorbeeld = Student
- Tabel in meervoud (lowercase): voorbeeld = students
- Factory in enkelvoud (UpperCamelCase): voorbeeld = StudentFactory
- Seeder in enkelvoud (UpperCamelCase): voorbeeld = StudentSeeder
- Controller in enkelvoud (UpperCamelCase): voorbeeld = StudentController

Als we dit doorvoeren wordt de controller gemaakt.

```
D:\Wamp\sites\laravelv1>php artisan make:controller --model=Category Admin/CategoryController
Controller created successfully.
```

Binnen app/Http/Controllers/Admin kunnen we de CategoryController vinden.



De methodes die erin staan hebben een specifieke functie

- |           |   |
|-----------|---|
| • index   | overzicht van categorie                                   |
| • create  | formulier om een nieuwe categorie toe te voegen           |
| • store   | opslaan van een categorie                                 |
| • show    | een categorie laten zien                                  |
| • edit    | een ingevuld formulier tonen om een categorie te wijzigen |
| • update  | opvangen van editformulier en wijzigen van categorie      |
| • destroy | het verwijderen van een categorie                         |

Met de resource controller hebben we dus alles wat we nodig hebben voor de crud.

Als je naar de onderdelen gaat kijken zien we dit. (Commentaar en regels kleiner gemaakt zodat het in 1 overzicht te zien is)

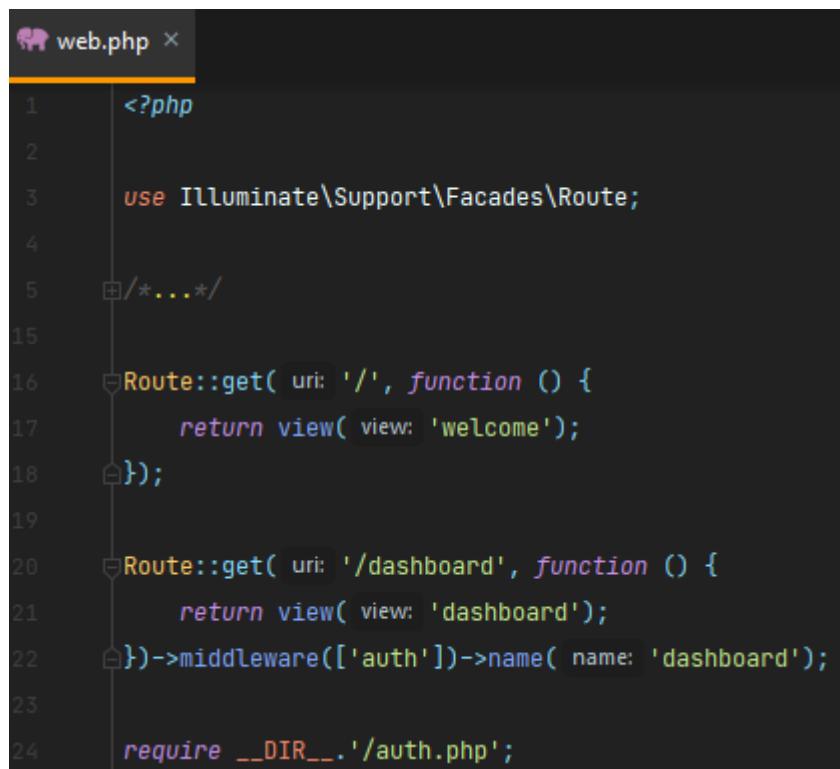
```
3  namespace App\Http\Controllers\Admin;
4
5  use App\Http\Controllers\Controller;
6  use App\Models\Category;
7  use Illuminate\Http\Request;
8
9  class CategoryController extends Controller
10 {
11
12     /** Display a listing of the resource. ...*/
13     public function index(){...}
14
15
16     /** Show the form for creating a new resource. ...*/
17     public function create(){...}
18
19
20     /** Store a newly created resource in storage. ...*/
21     public function store(Request $request){...}
22
23
24     /** Display the specified resource. ...*/
25     public function show(Category $category){...}
26
27
28     /** Show the form for editing the specified resource. ...*/
29     public function edit(Category $category){...}
30
31
32     /** Update the specified resource in storage. ...*/
33     public function update(Request $request, Category $category){...}
34
35
36     /** Remove the specified resource from storage. ...*/
37     public function destroy(Category $category){...}
38 }
```

Je ziet bovenin dat we een hoofdcontroller hebben die gebruikt wordt. Elke controller gebruikt deze. Je ziet ook dat we App\Models\Category gebruiken, want de model is die zelf hebben gemaakt. Je ziet ook Request erbij staan, dit is straks voor alles wat we met bijv een formulier ontvangen.

Er staan bij sommige methodes ook al argumenten in de methodes. Deze moet je laten staan, want die zorgen ervoor dat een aantal standaard functionaliteiten van Laravel werken.

### 3.2. Routing

Om ervoor te zorgen dat de resource controller straks gaan werken, moeten we ervoor zorgen dat we vanuit de site bij de controller kunnen komen. Dit doen we door middel van routing. Laravel wordt door middel van de url gestuurd. Als je ook nog even kijkt gaan we vanuit de client naar een controller. Hiervoor hebben we de web.php nodig die in de map routes staat.



```
<?php

use Illuminate\Support\Facades\Route;

/*...*/

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

Route::get('/dashboard', function () {
    return view('dashboard');
})->middleware(['auth'])->name('dashboard');

require __DIR__.'/auth.php';
```

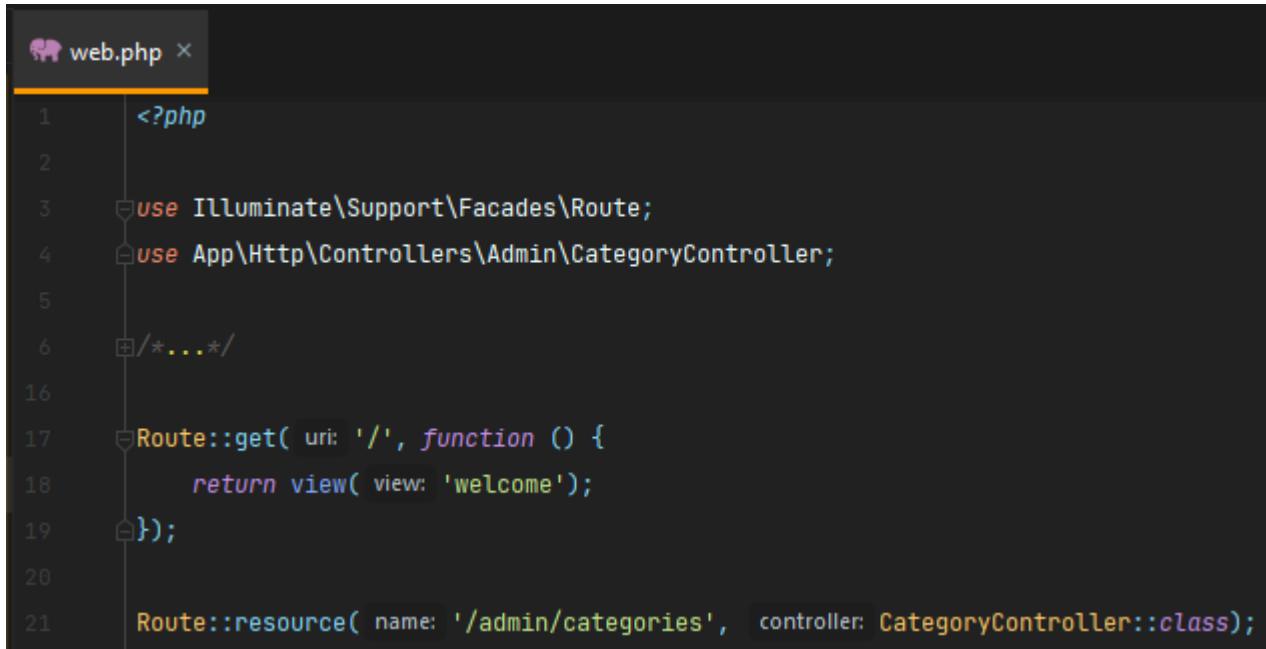
Wat op regel 16-18 nu staat is, als je op de website komt bij de beginmap /, laat dan een stuk lay-out zien van welcome. Deze view vind je in de map resources/views/layouts/welcome.blade.php

In deze view vind je html, terwijl het een .php extensie heeft. Laravel heeft een Blade template systeem, dus doordat het .blade.php is herkend het dat het een stuk lay-out is. Er staan eronder nog wat andere onderdelen in de web.php, maar dat bekijken we later.

Om ervoor te zorgen dat we straks naar onze resource controller komen, moeten we hier dus wat bij gaan zetten. We gaan meteen voor een resource routing, zodat alles meteen geregeld is voor alle methodes in ons bestand. Dit zou eigenlijk op de volgende manier geschreven moeten worden.

```
Route::resource('admin/categories', \App\Http\Controllers\Admin\CategoryController::class);
```

Enige is, we weten dat we straks meerdere resource routes zullen krijgen bij het gehele project. Verder staat het hele \App\Http\Controllers\Admin\ gewoon niet mooi. Hierdoor gaan we de controller in de use zetten.



```

1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\Admin\CategoryController;
5
6 /*...*/
17 Route::get('/', function () {
18     return view('welcome');
19 });
21 Route::resource('admin/categories', CategoryController::class);

```

De vraag dat is, wat krijgen we nu te zien. We hebben nog de controller niet uitgewerkt, maar normaal gesproken zullen we als we naar /admin/categories gaan de index methode uitvoeren.

Een overzicht hoe de resource routing werkt zie je hieronder. Wat ook meteen meegenomen is, is dat we named routes erbij krijgen voor elke methode. Die kunnen we gebruiken voor links, redirects en actions.

Dit lijstje zal je behoorlijk veel nodig hebben. Het is goed om even een shortcut te hebben naar de online versie.  
<https://laravel.com/docs/8.x/controllers#actions-handled-by-resource-controller>

### **Actions Handled By Resource Controller**

<b>Verb</b>	<b>URI</b>	<b>Action</b>	<b>Route Name</b>
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	destroy	photos.destroy

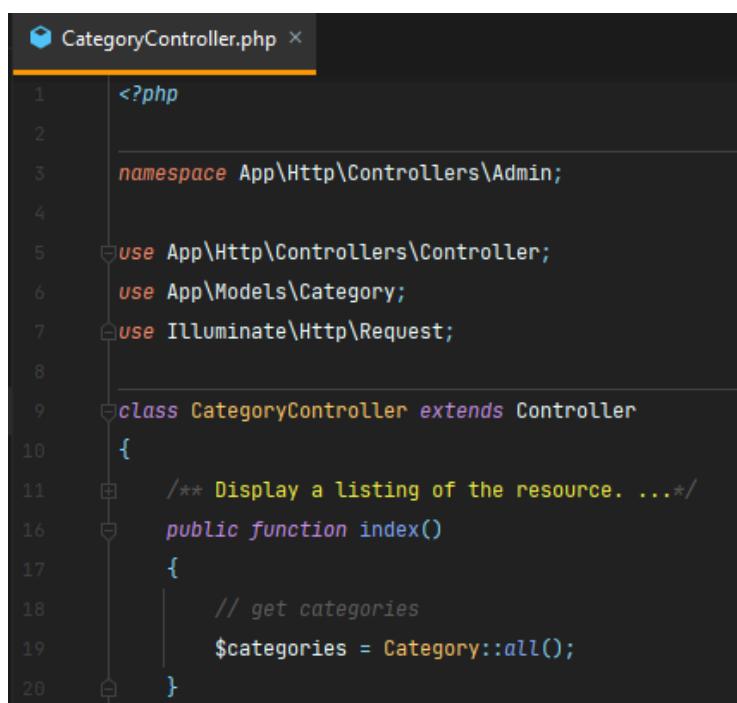
### 3.3. Index in controller

Bij de index in de resource controller moeten we ervoor zorgen dat we een overzicht krijgen van alle categorieën. Deze zullen getoond worden in een view. Als eerst gaan we alle categorieën ophalen. Nu gaan we even terug naar waarom we een model hebben:

## # Introduction

The Eloquent ORM included with Laravel provides a beautiful, simple ActiveRecord implementation for working with your database. Each database table has a corresponding "Model" which is used to interact with that table.

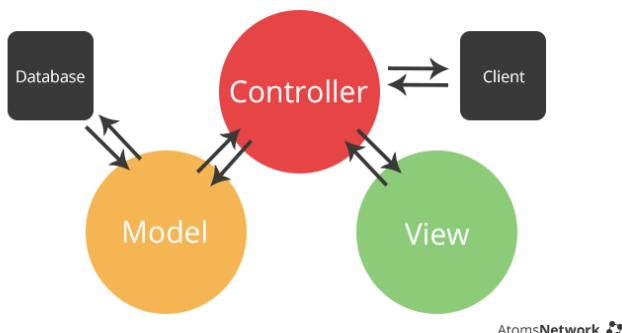
Omdat Laravel Eloquent heeft, kunnen we door middel van die functie heel makkelijk communiceren met de database. Zelf SQL-query's schrijven hoeft dus niet. Dit kan allemaal via de model. De model die we hebben gemaakt is Category. Bij de index() beginnen we dan ook om alle producten op te halen met de volgende code.



```
CategoryController.php
1 <?php
2
3 namespace App\Http\Controllers\Admin;
4
5 use App\Http\Controllers\Controller;
6 use App\Models\Category;
7 use Illuminate\Http\Request;
8
9 class CategoryController extends Controller
10 {
11     /**
12      * Display a listing of the resource. ...
13     */
14     public function index()
15     {
16         // get categories
17         $categories = Category::all();
18     }
19 }
```

Let even op dat we Category op regel 19 kunnen gebruiken, omdat we op regel 6 aangeven de model hiervan te gebruiken. Dit omdat we in de namespace van de controllers op dit moment zitten.

Nu we alle categorieën hebben opgehaald is natuurlijk de bedoeling om deze te laten zien. Dit regelen we verder in de lay-out. Wat we moeten doen is de gegevens doorsturen naar de view. De view is namelijk verantwoordelijk om alles te laten zien.

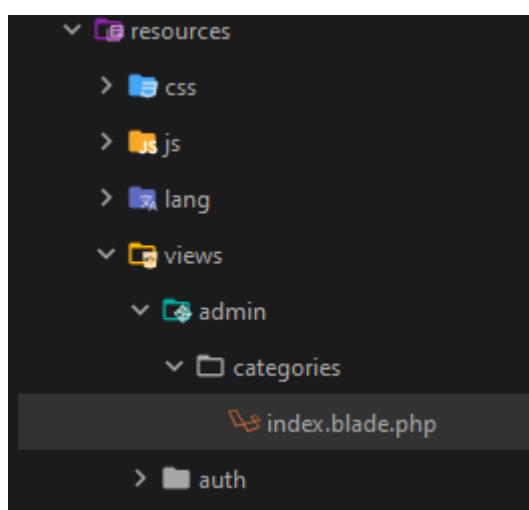


De index methode komt er dan zo uit te zien.

```
/** Display a listing of the resource. ...*/
public function index()
{
    // get categories
    $categories = Category::all();

    // return a view with $categories
    return view( view: 'admin.categories.index', compact( var_name: 'categories'));
}
```

Zoals je hierboven ziet hebben we een return, waar we een view openen. De view die we openen zal in een directory staan: `resources/views/admin/categories/index.blade.php`



De map admin en het bestand index.blade.php zullen we nog wel moeten aanmaken. Bij het aangeven van welke view we willen zien gebruiken we in de controller een punt notatie. De slash werkt ook gewoon, maar de punt notatie is gewoon net iets netter. De view hebben we op dit moment nog niet gemaakt, maar gaan als volgende onderdeel doen.

Met compact kan je variabelen meegeven richting de view. Je ziet geen \$ er staan, maar \$categories wordt op deze manier meegegeven. Als je wil kan je meerdere variabelen meegeven. Op dit moment nog niet nodig, maar dit gaan we later een keer gebruiken.

Je zou ook zonder compact variabelen kunnen sturen, maar dan zal je een array moeten gebruiken.

Verder gaan we in de views map allerlei mappen aanmaken om de views overzichtelijk te houden. Omdat we voor de categories allerlei schermen krijgen die ieder een eigen view krijgen, maken we een map categories aan, waarin we dan de index.blade.php inzetten. Verder zijn we nu bezig met een CRUD. Dit is om het beheer te doen, wat dus ook in een admin moet gaan komen.

### 3.4. Index in View

In de index.blade.php gaan we ervoor zorgen dat we een klein stukje html hebben, wat straks op de plek zal komen in de masterpage bij yield('content'). Zorgen dat de masterpage gebruikt wordt doen we met @extends

```
@extends('layouts.layout')

@section('content')

    <div class="container mx-1">
        <div class="ml-2 flex flex-col">
            <h2 class="my-4 text-4xl font-semibold text-gray-600 dark:text-gray-400">
                Category Admin
            </h2>
        </div>
    </div>

@endsection
```

Met @section zorgen we ervoor dat wat hierbinnen staat dan in een yield komt. Dus @section('content') komt terecht in yield('content')

In onze index willen we verder een tabel hebben waar we alle producten laten zien. Dit doen we op de volgende manier: (je ziet niet alle html code)

```
<tbody class="bg-white divide-y divide-gray-200">
@foreach($categories as $category)
<tr>
    <td class="px-6 py-4 whitespace nowrap text-sm font-medium text-gray-900">
        {{ $category->name }}
    </td>
    <td class="px-6 py-4 whitespace nowrap text-sm text-gray-500">
        Details
    </td>
    <td class="px-6 py-4 whitespace nowrap text-sm text-gray-500">
        Edit
    </td>
    <td class="px-6 py-4 whitespace nowrap text-sm text-gray-500">
        Delete
    </td>
</tr>
@endforeach

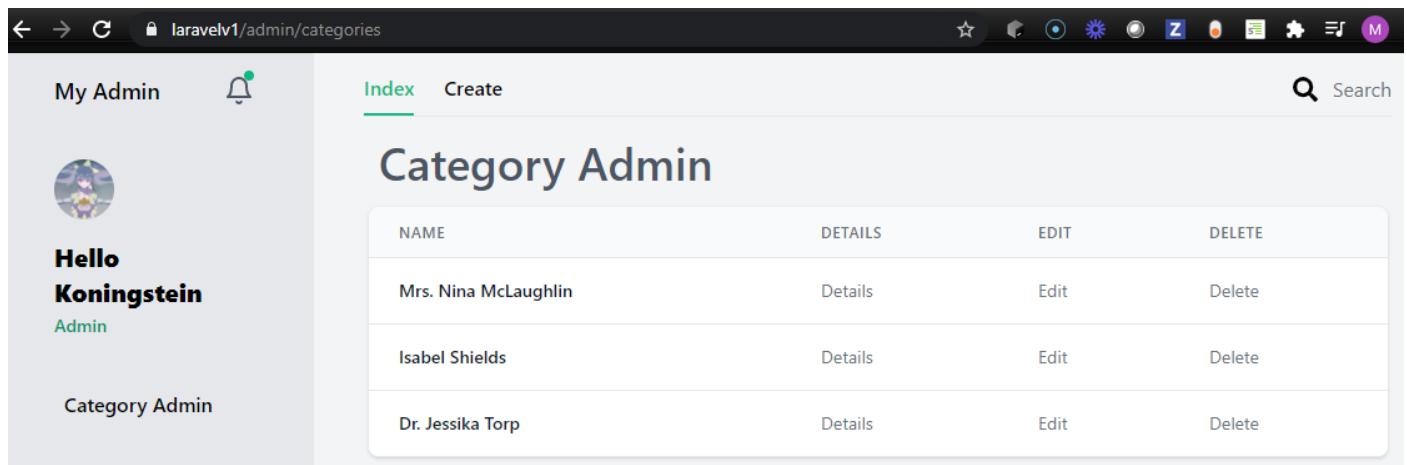
<!-- More items... -->
</tbody>
```

In de controller hebben we met compact('categories') de variabel \$categories meegegeven aan de lay-out. Deze is dat ook als \$categories beschikbaar.

Deze zetten we in een @foreach, wat hetzelfde werkt als de foreach in php. De variabel \$category is een object waar we dan ook de info vandaan kunnen halen. Met de {{ }} printen we de variabel op de plek.

In onze database hebben we bij category een id en een name. Dit zijn dus ook de properties van het object wat ik nu kan gebruiken.

Als we dan naar onze pagina gaan, moeten we naar `localhost/admin/categories` gaan om dit te zien. Als het goed is krijg je dan een overzicht te zien van alle producten. Waarschijnlijk gaan we later de url nog veranderen, maar zolang we in de controller met named routes werken is het aanpassen van urls niet veel werk.



The screenshot shows a web browser window with the URL `laravelv1/admin/categories`. On the left, there's a sidebar with a user profile picture, the text "Hello Koningstein Admin", and a "Category Admin" menu item. The main content area has a header "Category Admin". Below it is a table with four rows, each representing a category. The columns are labeled "NAME", "DETAILS", "EDIT", and "DELETE". The data in the table is as follows:

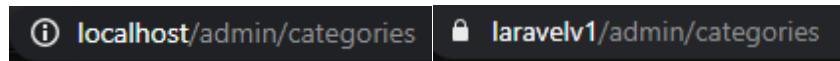
NAME	DETAILS	EDIT	DELETE
Mrs. Nina McLaughlin	Details	Edit	Delete
Isabel Shields	Details	Edit	Delete
Dr. Jessika Torp	Details	Edit	Delete

We zien netjes een lijst met categories verschijnen. Dit overzicht is nog niet af, maar dit komt later als we gaan werken aan het toevoegen, wijzigen en verwijderen. Het gaat alleen nu niet om uiterlijk maar om de technieken die erachter zitten. Verder is het misschien straks handig om de menu items bovenin niet in de master layout te hebben, zodat deze voor elke admin makkelijk anders kunnen worden.

### 3.4.1. Van url naar scherm

Nog even nadenken hoe nu deze pagina tot stand komt. Dit zijn de globale stappen die gebeuren zodra je naar een pagina gaat.

Je gaat naar de url localhost/admin/categories (als je werkt met virtual hosts, kan het bijv ook net als bij mij laravelv1/admin/categories zijn).



In de routes/web.php verwijst deze url naar de CategoryController

```
20 | Route::resource( 'name: '/admin/categories', controller: CategoryController::class);
```

Doordat we een Resource controller en een Resource route hebben gemaakt, gaan we met een GET en deze url naar de index methode van de controller.

#### Actions Handled By Resource Controller

Verb	URI	Action	Route Name
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	destroy	photos.destroy

In de index halen we door middel van de Model de data op uit de categoriestabel en zorgen we dat de data naar de admin.categories.index view gaat.

```
9 class CategoryController extends Controller
10 {
11     /**
12      * Display a listing of the resource. ...
13     */
14     public function index()
15     {
16         // get categories
17         $categories = Category::all();
18
19         // return a view with $categories
20         return view( view: 'admin.categories.index', compact( varname: 'categories'));
21     }
22 }
23 }
```

In de index.blade.php extenden we de lay-out, waardoor de masterpage wordt ingeladen.

```
@extends('layouts.layout')

@section('content')

    <div class="container mx-1">
        <div class="ml-2 flex flex-col">
            <h2 class="my-4 text-4xl font-semibold text-gray-600 dark:text-gray-400">
                Category Admin
            </h2>
        </div>
    </div>

@endsection
```

In de masterpage staat een @yield('content'), wat ervoor zorgt dat alles in de section content van index.blade.php daar zal komen.

```
<div
    class="pb-2 flex items-center justify-between text-gray-600
    dark:text-gray-400 border-b dark:border-gray-600">

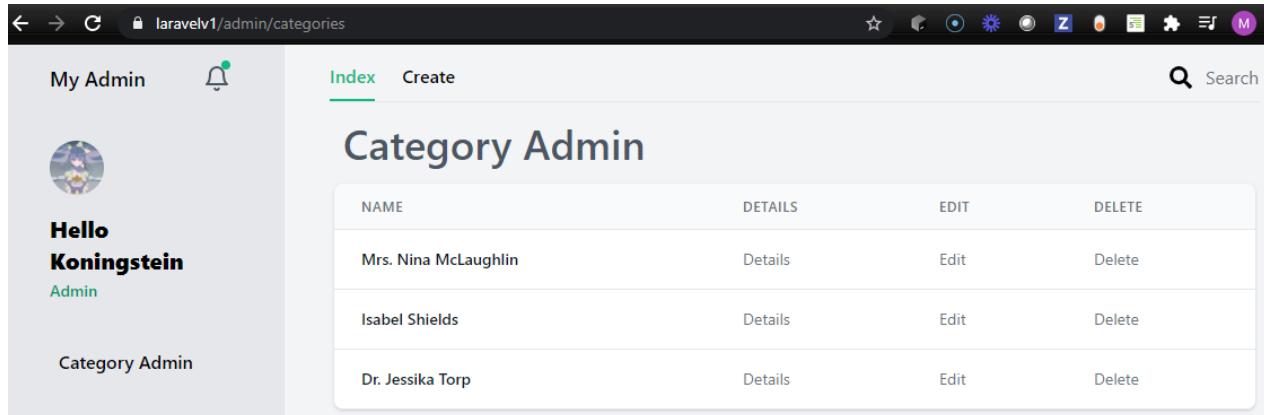
    @yield('content')

</div>

<tbody class="bg-white divide-y divide-gray-200">
    @foreach($categories as $category)
        <tr>
            <td class="px-6 py-4 whitespace nowrap text-sm" >
                {{ $category->name }}
            </td>
            <td class="px-6 py-4 whitespace nowrap text-sm" >
                Details
            </td>
            <td class="px-6 py-4 whitespace nowrap text-sm" >
                Edit
            </td>
            <td class="px-6 py-4 whitespace nowrap text-sm" >
                Delete
            </td>
        </tr>
    @endforeach

```

Wat er in de browser wordt getoond is een overzicht van producten, netjes in een Tailwind lay-out



The screenshot shows a web browser window with the URL 'laravelv1/admin/categories'. The page has a dark header with navigation icons. On the left, there's a sidebar with a user profile picture, the name 'Hello Koningstein Admin', and a 'Category Admin' link. The main content area has a title 'Category Admin' and a table with four rows of data. The table columns are 'NAME', 'DETAILS', 'EDIT', and 'DELETE'. The data rows are: 'Mrs. Nina McLaughlin' with 'Details', 'Edit', and 'Delete' links; 'Isabel Shields' with 'Details', 'Edit', and 'Delete' links; and 'Dr. Jessika Torp' with 'Details', 'Edit', and 'Delete' links.

NAME	DETAILS	EDIT	DELETE
Mrs. Nina McLaughlin	Details	Edit	Delete
Isabel Shields	Details	Edit	Delete
Dr. Jessika Torp	Details	Edit	Delete

### 3.5. Opdracht 4: index

Maak voor de tabel projects een admin door middel van een resource controller. Zorg dat je de index ziet binnen je eigen masterpage. Op de index moeten de volgende onderdelen te zien zijn:

- Project id
- Naam van het project

De view van de index moet in de volgende map zitten en ook de correcte naam hebben:

- Resources/views/admin/projects/index.blade.php

Je moet gebruik maken van de named routes die bij een resource controller aan gemaakt worden.

#### Opdracht controle

Als je de opdracht af hebt en gebruik hebt gemaakt van de opdrachten repository, kan je het volgende commando gebruiken om te controleren of alles goed werkt:

```
.\vendor\bin\pest --group=Opdracht3_4
```

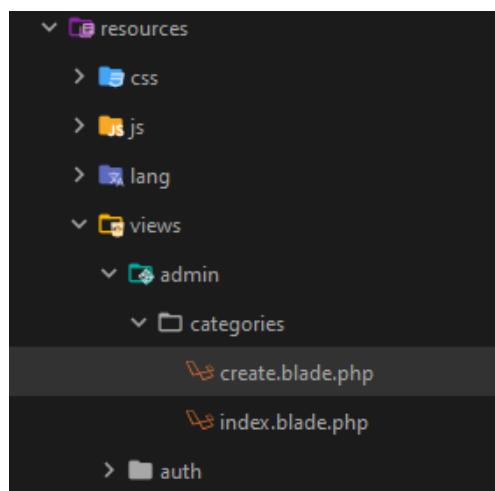
### 3.6. Create

Nu de index functionaliteit klaar is gaan we met de create beginnen. Bij de create moet een formulier worden getoond die ervoor gaat zorgen dat we straks een categorie kunnen toevoegen. Het opvangen gebeurt in de store methode, dus daar hoeven we nog niet aan te denken.

Dit betekent dus, dat we eigenlijk alleen maar ervoor hoeven te zorgen dat je een formulier ziet en deze komt in een view. In de create methode zetten we dus alleen maar:

```
/** Show the form for creating a new resource. ...*/
public function create()
{
    // return a view with a form to create a category
    return view( view: 'admin.categories.create');
}
```

Bij de views map gaan we in het mapje admin/categories een create.blade.php aanmaken.



En in de create zorgen we meteen ervoor dat de masterpage ingeladen wordt en onze section er staat.

```
@extends('layouts.layout')

@section('content')



<div class="ml-2 flex flex-col">
        <h2 class="my-4 text-4xl font-semibold text-gray-600 dark:text-gray-400">
            Category Admin
        </h2>
    </div>

</div>
@endsection


```

Nu gaan we met html een formulier maken. Er zijn allerlei voorbeelden op internet te vinden om met Tailwind een formulier te maken. Zelf heb ik dit nu ook gedaan zodat simpel blijft.

```
@section('content')
    <div class="container mx-1">
        <div class="ml-2 flex flex-col">
            <h2 class="my-4 text-4xl font-semibold text-gray-600 dark:text-gray-400">
                Category Admin
            </h2>
        </div>

        <form id="form" class="bg-white shadow-md rounded-lg px-8 pt-6 pb-8 mb-4"
            action="{{ route('categories.store') }}" method="POST">
            @csrf
            <div class="mb-4">
                <label class="block text-gray-700 text-sm font-bold mb-2" for="name">
                    Name
                </label>
                <input
                    class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
                    focus:outline-none focus:shadow-outline" name="name" id="name"
                    type="text" required>
            </div>
            <div class="flex items-center justify-between">
                <button id="submit"
                    class="bg-green-500 hover:bg-green-700 text-white font-bold py-2 px-4 rounded
                    focus:outline-none focus:shadow-outline" type="submit">Submit
                </button>
            </div>
        </form>
    </div>
@endsection
```

Als we een POST stuurt naar /admin/categories gaan we naar de store methode in onze controller. Maar omdat je soms bestanden kan willen verschuiven is het beter om met named routes te werken zoals in de laatste kolom staat. Voor ons wordt het dus route('categories.store')

POST	/photos	store	photos.store
------	---------	-------	--------------

Let er wel op dat je @csrf erbij zet. Dit zorgt voor een stuk beveiliging.

```
<form id="form" class="bg-white shadow-md rounded px-8 pt-6 pb-8 mb-4"
    action="{{ route('categories.store') }}" method="POST">
    @csrf
```

In de documentatie kan je dit over @csrf vinden:

Laravel makes it easy to protect your application from [cross-site request forgery](#) (CSRF) attacks. Cross-site request forgeries are a type of malicious exploit whereby unauthorized commands are performed on behalf of an authenticated user.

Laravel automatically generates a CSRF "token" for each active user session managed by the application. This token is used to verify that the authenticated user is the one actually making the requests to the application.

Anytime you define a HTML form in your application, you should include a hidden CSRF token field in the form so that the CSRF protection middleware can validate the request. You may use the `@csrf` Blade directive to generate the token field:

Als gebruiker van de site krijg je dus een CSRF token. Bij de opvang van een formulier wordt naar de CSRF gekeken, of je wel dezelfde gebruiker bent. Je kan dit zien in de bron:

```
<form id="form" class="bg-white shadow-md rounded px-8 pt-6 pb-8 mb-4"
    action="http://localhost/admin/categories" method="POST">
    <input type="hidden" name="_token" value="Q2S3vU8RdKYZ2uVWkNaCq2QoJKEhQ0MF0cMkKiIq">
```

Als we nu naar `localhost/admin/categories/create` gaan zien we dit:

The screenshot shows a web browser window with the address bar containing 'localhost/admin/categories/create'. The page itself is titled 'Category Admin'. On the left side, there's a sidebar with a user profile picture, the name 'Hello Koningstein', and the word 'Admin'. The main content area has a form with a single input field labeled 'Name' and a green 'Submit' button below it.

De create is nu tot zover klaar, want het werkt. Alleen is de gebruiksvriendelijkheid ook van belang, waardoor we nog wat kleine aanpassingen zullen maken.

Het zal nu wel handiger worden als we in onze lay-out niet steeds de url's in hoeven te typen. Het Topmenu moet nu gaan werken. Op dit moment staat het topmenu in de master layout. Deze haal ik eruit en plaats ik in de childen. Nu staat in de master layout dit:

```
<nav  
    class="flex flex-row justify-between border-b  
    dark:border-gray-600 dark:text-gray-400 transition duration-500  
    ease-in-out">  
    <div class="flex">  
        <!-- Top NavBar -->  
        @yield('topmenu_items')  
    </div>
```

Met yield zorg ik dat we nu met section('topmenu\_items') aan de slag kunnen in onze layout. Dat ziet er dan zo uit:

```
@section('topmenu_items')  
    <!-- Top NavBar -->  
    <a href="{{ route('categories.index') }}">  
        <button  
            class="py-2 block border-b-2 border-transparent  
            focus:outline-none font-medium capitalize text-center  
            focus:text-green-500 focus:border-green-500  
            dark-focus:text-green-200 dark-focus:border-green-200  
            transition duration-500 ease-in-out">  
            Index  
        </button></a>  
    <a  
        href="{{ route('categories.create') }}"  
        class="ml-6 py-2 block text-green-500 border-green-500  
        dark:text-green-200 dark:border-green-200  
        focus:outline-none border-b-2 font-medium capitalize  
        transition duration-500 ease-in-out">  
        Create  
    </a>  
@endsection
```

Dit betekent dus, dat in onze index en create in de map categories, deze section toegevoegd moet worden zodat het top menu te zien is. Ook hier maar ik weer gebruik van de named routes binnen Laravel.

Tot zover is de create nu klaar. Extra toevoegingen zullen later nog komen zodra we met validatie bezig zijn

### 3.7. Store

Voor het opslaan van gegevens vanuit een formulier gebruiken we de store methode. Alleen als we iets gaan opslaan zullen we wel wat over de beveiliging moeten weten. Hier gaat het bijv. over mass assignment:

## # Mass Assignment

You may also use the `create` method to save a new model in a single line. The inserted model instance will be returned to you from the method. However, before doing so, you will need to specify either a `fillable` or `guarded` attribute on the model, as all Eloquent models protect against mass-assignment by default.

A mass-assignment vulnerability occurs when a user passes an unexpected HTTP parameter through a request, and that parameter changes a column in your database you did not expect. For example, a malicious user might send an `is_admin` parameter through an HTTP request, which is then passed into your model's `create` method, allowing the user to escalate themselves to an administrator.

Wat hier dus staat is, dat we met fillable kunnen zeggen welke attributen gevuld mogen worden. Verder bestaat er ook nog guarded:

### Guarding Attributes

While `$fillable` serves as a "white list" of attributes that should be mass assignable, you may also choose to use `$guarded`. The `$guarded` property should contain an array of attributes that you do not want to be mass assignable. All other attributes not in the array will be mass assignable. So, `$guarded` functions like a "black list". Importantly, you should use either `$fillable` or `$guarded` - not both. In the example below, all attributes **except for** `price` will be mass assignable:

Zoals hier beschreven staat kunnen we dus **of \$fillable** gebruiken **of \$guarded** gebruiken in onze controller. \$fillable is een white list, dus welke attributen mogen we met mass assignment vullen. \$guarded is een blacklist, welke attributen we niet met mass assignment mogen vullen.

Nu heb je verschillende manieren om iets op te slaan in Laravel. Waarschijnlijk is het verhaal over fillable en guarded nog een beetje vaag, dus laten we gewoon is kijken wat er gebeurt als we onze store methode gaan maken.

Als je naar de store methode gaat kijken kunnen we deze op de volgende manier maken:

```
36     /** Store a newly created resource in storage. ...*/
42     public function store(Request $request)
43     {
44         $category = Category::create(['name' => $request->name]);
45     }
```

In de \$request wordt alle Postdata bewaart.

Als we dan de create formulier invullen en submitten krijgen we een foutmelding.

D:\Wamp\sites\laravelv1\

Illuminate\Database\Eloquent\MassAssignmentException

Add [name] to fillable property to allow mass assignment on [App\Models\Category].

<https://laravelv1/admin/categories>

Je ziet dus dat het nu niet is toegestaan. Als we de onderdelen die we verwachten in \$fillable zetten zal dat anders zijn. Dit moet in de Model gebeuren. De model kan je vinden in *app/Models/Category.php*

```
Category.php
<?php
namespace App\Models;
use ...
class Category extends Model
{
    use HasFactory;
    protected $fillable = ['name'];
}
```

We zetten de \$fillable hierin, met wat we verwachten vanuit het formulier, en dat is een name.

Als je nu weer probeert om het formulier te sturen zal je een wit scherm krijgen. Dit komt omdat we in de methode store verder niks doen.

De bedoeling is, dat als we het formulier hebben verstuurd, we weer terugkomen op de index pagina. Je zou dus misschien zeggen dat we de index view moeten returnen.

```
/** Store a newly created resource in storage. ...*/
public function store(Request $request)
{
    $category = Category::create(['name' => $request->name]);

    return view('admin.categories.index');
}
```

We krijgen dan een andere foutmelding

D:\Wamp\sites\laravelv1\

ErrorException

Undefined variable \$categories (View:

D:\Wamp\sites\laravelv1\resources\views\admin\categories\index.blade.php)

<https://laravelv1/admin/categories>

Hide solutions

Possible typo \$categories

Did you mean \$errors ?

We zeggen namelijk in onze store methode, return view('admin.categories.index'). Het enige is dat we vergeten de data op te halen zoals bij de index methode.

Je zou zeggen, haal dan de data eerst op en return het dan, maar dan doen we exact hetzelfde als wat er bij de index methode gedaan wordt. Het is dus beter om te verwijzen naar de index methode. Dit kunnen we doen met een redirect. In de redirect gebruiken we weer de named route. (Let even op dat je bij de named route niet de admin directory erbij mag zetten)

```
/** Store a newly created resource in storage. ...*/
public function store(Request $request)
{
    $category = Category::create(['name' => $request->name]);

    return redirect()->route('categories.index');
```

De gegevens staan nu in de database en die zie je ook in het overzicht.

Er is ook een andere manier die wat meer lijkt op wat je met OOP hebt gehad ziet er zo uit. Zelf vind ik dit prettiger. Op deze manier kan sowieso geen mass assignment dus zou je fillable niet nodig hebben om het te laten werken. Het kan nooit kwaad om het toch te laten staan....

```
/** Store a newly created resource in storage. ...*/
public function store(Request $request)
{
    $category = new Category();
    $category->name = $request->name;
    $category->save();

    return redirect()->route('categories.index');
```

Je ziet hier dat we eerst een object aanmaken van Category. Daarna vullen we de property name, en saven het daarna in de database. Duidelijke stappen in je code zoals je ziet.

Ik zal verder gaan met deze versie van de store methode.

### 3.8. Opdracht 5: create & store

Maak binnen de resource controller van project de create en store methode en zorg dat het binnen je eigen masterpage werkt.

Op de create zal een formulier beschikbaar moeten zijn die post naar de store methode.

De view van de create moet in de volgende map zitten en ook de correcte naam hebben:

- Resources/views/admin/projects/create.blade.php

Als je het formulier hebt verstuurd kom je bij de store methode. Hier zal de data worden opgeslagen in de tabel projects. Er wordt gecontroleerd op een name en description. Nadat het is opgeslagen wordt je automatisch naar de index gestuurd met de melding dat het project is toegevoegd.

Je moet gebruik maken van de named routes die bij een resource controller aan gemaakt worden.

### Opdracht controle

Als je de opdracht af hebt en gebruik hebt gemaakt van de opdrachten repository, kan je het volgende commando gebruiken om te controleren of alles goed werkt:

```
.\vendor\bin\pest --group=Opdracht5
```

### 3.9. Validatie in de store

Om ervoor te zorgen dat je in de database opslaat wat je wilt, moet er een controle komen waarbij we de data valideren. Je zal veel voorbeelden op internet tegenkomen waar dat gebeurt in de controller, alleen is dit niet de plek. Als je een groot formulier gaat krijgen, moet er meer data gevalideerd worden en wordt eigenlijk de methode te groot.

Laravel heeft hier de Request voor. We kunnen een eigen request aanmaken waar we de validatie inzetten. Eerst is even kijken hoe dit gaat met php artisan help make:request

```
D:\Wamp\sites\laravelv1>php artisan help make:request
Description:
Create a new form request class

Usage:
make:request <name>

Arguments:
name           The name of the class

Options:
-h, --help      Display help for the given command. When no command is given display help for the list command
-q, --quiet     Do not output any message
```

Je ziet dat we alleen een name mee kunnen geven, de rest maakt voor ons niet echt veel uit. Dus gebruiken we `php artisan make:request CategoryStoreRequest`

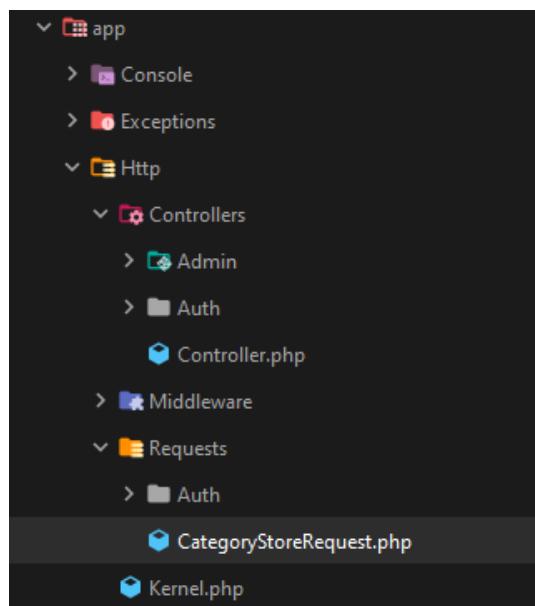
```
D:\Wamp\sites\laravelv1>php artisan make:request CategoryStoreRequest  
Request created successfully.
```

Om ervoor te zorgen dat hier een duidelijke naam voor is zetten we deze ook in onze regels. Als je kijkt naar de naam zit het zo in elkaar: ControllerMethodRequest

Deze voegen we toe aan de regels:

- Model in enkelvoud (UpperCamelCase): voorbeeld = Student
- Tabel in meervoud (lowercase): voorbeeld = students
- Factory in enkelvoud (UpperCamelCase): voorbeeld = StudentFactory
- Seeder in enkelvoud (UpperCamelCase): voorbeeld = StudentSeeder
- Controller in enkelvoud (UpperCamelCase): voorbeeld = StudentController
- Request met ControllerMethodRequest (UpperCamelCase): voorbeeld = StudentStoreRequest

In de app/Http/ map is een map voor Requests, waar nu onze *CategoryStoreRequest.php* staat.



Als we gaan kijken naar de request ziet deze er zo uit

```
3  namespace App\Http\Requests;
4
5  use Illuminate\Foundation\Http\FormRequest;
6
7  class CategoryStoreRequest extends FormRequest
8  {
9      /**
10      * Determine if the user is authorized to make this request. ...
11      */
12      public function authorize()
13      {
14          return false;
15      }
16
17      /**
18      * Get the validation rules that apply to the request. ...
19      */
20      public function rules()
21      {
22          return [
23              //
24          ];
25      }
26  }
```

Deze request gaan we gebruiken voor de validatie.

De methode authorize() is om ervoor te zorgen dat alleen gebruikers die rechten hebben de methode kunnen gebruiken. Omdat we nog helemaal niks met een login of rechten gedaan hebben, zullen we deze voor nu even op true moeten zetten.

```
public function authorize()
{
    return true;
}
```

In de methode rules() staat een return met een array. In deze array kunnen we onze regels neerzetten die we nodig hebben voor de name. Alle mogelijkheden kan je hier vinden:

<https://laravel.com/docs/8.x/validation#available-validation-rules>

Als je iets wil maar dit zit niet in de standaard kan je zelf deze toevoegen. Dit gaan we nu niet doen.

De validatie kan er bijvoorbeeld zo uitzien.

```
    /** Get the validation rules that apply to the request. ...*/
    public function rules()
    {
        return [
            'name' => 'required|string|unique:categories|min:5|max:75'
        ];
    }
```

- Bij name is het verplicht om wat in te vullen
- De name moet een string zijn
- De name mag niet eerder gebruikt zijn (dus nog niet in de database staan)
- De name moet minimaal 5 characters bevatten.
- De name mag maximaal 75 characters bevatten. We gebruiken 75 characters omdat ons veld in de database die grootte heeft. (we hadden met de update migration dit namelijk veranderd)

In de controller moeten we nu aangeven dat we onze eigen validatie request willen gebruiken. Hiervoor zullen we dus de Request moeten wijzigen in CategoryStoreRequest. De eigen gemaakte request is een uitbreiding op de standaard request.

```
    /** Store a newly created resource in storage. ...*/
    public function store(CategoryStoreRequest $request)
    {
        $category = new Category();
        $category->name = $request->name;
        $category->save();

        return redirect()->route('categories.index');
    }
```

Sommige editors zorgen vanzelf dat de Request class dan ook bij use komt. Zoniet, zorg dat je boven de class deze toevoegt.

```
3  namespace App\Http\Controllers\Admin;
4
5  use App\Http\Controllers\Controller;
6  use App\Http\Requests\CategoryStoreRequest;
7  use App\Models\Category;
8  use Illuminate\Http\Request;
9
10 class CategoryController extends Controller
11 {
```

Omdat we onze Request als argument meegeven hoeven we niet meer de validatie verder te activeren. Automatisch zal de request door de StoreProductsRequest nu gaan. Je ziet soms ook validatie in de controller, wat niet netjes is. We geven de Request dus netjes mee als argument.

```
/** Store a newly created resource in storage. ...*/
public function store(CategoryStoreRequest $request)
{
    $category = new Category();
    $category->name = $request->name;
    $category->save();

    return redirect()->route('categories.index');
}
```

Als het formulier verstuurd wordt maar het komt niet door de validatie heen zal je automatisch weer het formulier zien met de data die je erin hebt gezet. Let even op dat je niet standaard de informatie ziet als je teruggaat naar het formulier.

Wanneer we wel door de validatie heen komen wordt de data in de tabel toegevoegd.

```
</div>

@if($errors->any())
    <div class="bg-red-200 text-red-900 rounded-lg shadow-md p-6 pr-10 mb-8" style="...">
        <ul class="mt-3 list-disc list-inside text-sm text-red-600">
            @foreach($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif

<form id="form" class="bg-white shadow-md rounded-lg px-8 pt-6 pb-8 mb-4" action="{{ route('categories.store') }}" method="POST">
```

Het is prettig voor gebruikers dat als er iets is toegevoegd om nog even een melding te krijgen. Ook is het handig om als we niet door de validatie heen komen te weten wat er mis is. Hiervoor gaan we de views aanpassen van de index en de create.

In de create.blade.php ga ik de errors opvangen. Ik kies ervoor om alle errors / messages op dezelfde plek te tonen, namelijk onder de titel van de pagina. Ik gebruik hiervoor de standaard soort van Laravel. Je kan deze hier vinden:

<https://laravel.com/docs/8.x/validation#quick-displaying-the-validation-errors>

Nu zie je hier dat in Blade je een if net zoals in php kan gebruiken. Erg handig omdat we dit niet willen zien als er geen error is. We bekijken of er errors zijn, zo ja, gaan we door de array heen van errors en laten deze zien.

Verder kan het veld waar de input ook nog duidelijk worden gemaakt met @error. We geven dan de input een rode border op deze manier.

```
</label>
<input
    class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
    focus:outline-none focus:shadow-outline @error('name') border-red-500 @enderror" name="name" id="name"
    value="{{ old('name') }}" type="text" required>
</div>
```

Het is wel slim om ook de oude waarde te tonen, zodat de gebruiker ziet wat er was ingetypt. Dit kan je doen met `value="{{old('name')}}"`. Dit zorgt ervoor dat de oude waarde wordt getoond. Het ziet er dan zo uit:

The screenshot shows a web page titled "Category Admin". At the top, there are two navigation links: "Index" and "Create", with "Create" being underlined. Below the title, there is a red callout box containing the text: "• The name must be at least 5 characters.". Underneath the callout box is a form field labeled "Name" with the value "1234" entered. At the bottom of the form is a green "Submit" button.

In de index.blade.php gaan we een flash message gebruiken, om ervoor te zorgen dat als de categorie netjes wordt opgeslagen we hier een bericht over krijgen. Laravel gebruikt hiervoor de session.

<https://laravel.com/docs/8.x/responses#redirecting-with-flashed-session-data>

Ook deze zet ik onder de titel.

```
Category Admin
</h2>
</div>

@if(session('status'))
    <div class="bg-green-200 text-green-900 rounded-lg shadow-md p-6 pr-10 mb-8"
        style="...">
        {{ session('status') }}
    </div>
@endif

<div class="flex flex-col">
```

Nu moeten we alleen nog zorgen dat met de redirect in de store() methode we de message meegeven die hier getoond moet worden.

```
/** Store a newly created resource in storage. ...*/
public function store(CategoryStoreRequest $request)
{
    $category = new Category();
    $category->name = $request->name;
    $category->save();

    return redirect()->route('categories.index')->with('status', 'Categorie aangemaakt');
}
```

Als het wel goed gaat, je bent door de validatie heen gekomen, wordt het product toegevoegd en zie je een melding als je bij de index komt.

Category Admin			
Categorie aangemaakt			
NAME	DETAILS	EDIT	DELETE
Mrs. Nina McLaughlin	Details	Edit	Delete
Isabel Shields	Details	Edit	Delete

De validatie voor het opslaan van categorie is nu klaar.

### 3.10. Opdracht 6: validatie

Voor het opslaan van projects wordt nog alles geaccepteerd. De validatie moet worden toegevoegd.

Bij de validatie wordt gecontroleerd op de volgende voorwaarden:

- Naam van het project is ingevuld
- De naam is minimaal 5 karakters, maximaal 45 karakters
- De projectnaam moet uniek zijn
- De beschrijving moet ingevuld zijn. Hier is geen minimale of maximale waarde

Als er niet aan de validatie wordt voldaan, moet het zichtbaar in het formulier zijn welke onderdelen fout zijn. De foutmeldingen moeten binnen de lay-out op het scherm komen.

Als het aan de validatie voldoet moet het worden opgeslagen in de tabel projects. Er zal een redirect moeten komen naar de index met een melding dat het project is opgeslagen.

### Opdracht controle

Als je de opdracht af hebt en gebruik hebt gemaakt van de opdrachten repository, kan je het volgende commando gebruiken om te controleren of alles goed werkt:

```
.\vendor\bin\pest --group=Opdracht6
```

### 3.11. Show

De show methode lijkt een beetje op de index. Bij de index halen we alle rijen op en zorgen we dat die naar de view worden gestuurd. Bij de show gaat het om 1 rij. Waar we nu wel voor het eerst mee te maken krijgen is Model Binding. Laravel zegt het volgende over Model Binding:

## # Route Model Binding

When injecting a model ID to a route or controller action, you will often query to retrieve the model that corresponds to that ID. Laravel route model binding provides a convenient way to automatically inject the model instances directly into your routes. For example, instead of injecting a user's ID, you can inject the entire `User` model instance that matches the given ID.

Dit betekent, dat als we een id in de URL hebben, automatisch de gegevens van de id opgehaald worden. Bij de resource controller gaan we dan meteen even kijken hoe dat zit.

### Actions Handled By Resource Controller

Verb	URI	Action	Route Name
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	destroy	photos.destroy

Zoals je hierboven ziet zitten we in de controller map. Daarna zie je {photo} staan. Hier zal een id komen te staan van de foto. Voor onze controller met categories komt daar dus een id te staan van categories, ofwel:

`admin/categories/{category}`

Als we dan kijken naar de show methode zoals die nu is, gaan we dus straks een id meekrijgen vanuit de URL. Hiermee wordt automatisch door Laravel wat meegegeven. In de methode wordt namelijk meteen \$category meegegeven, met als type Category (de model class).

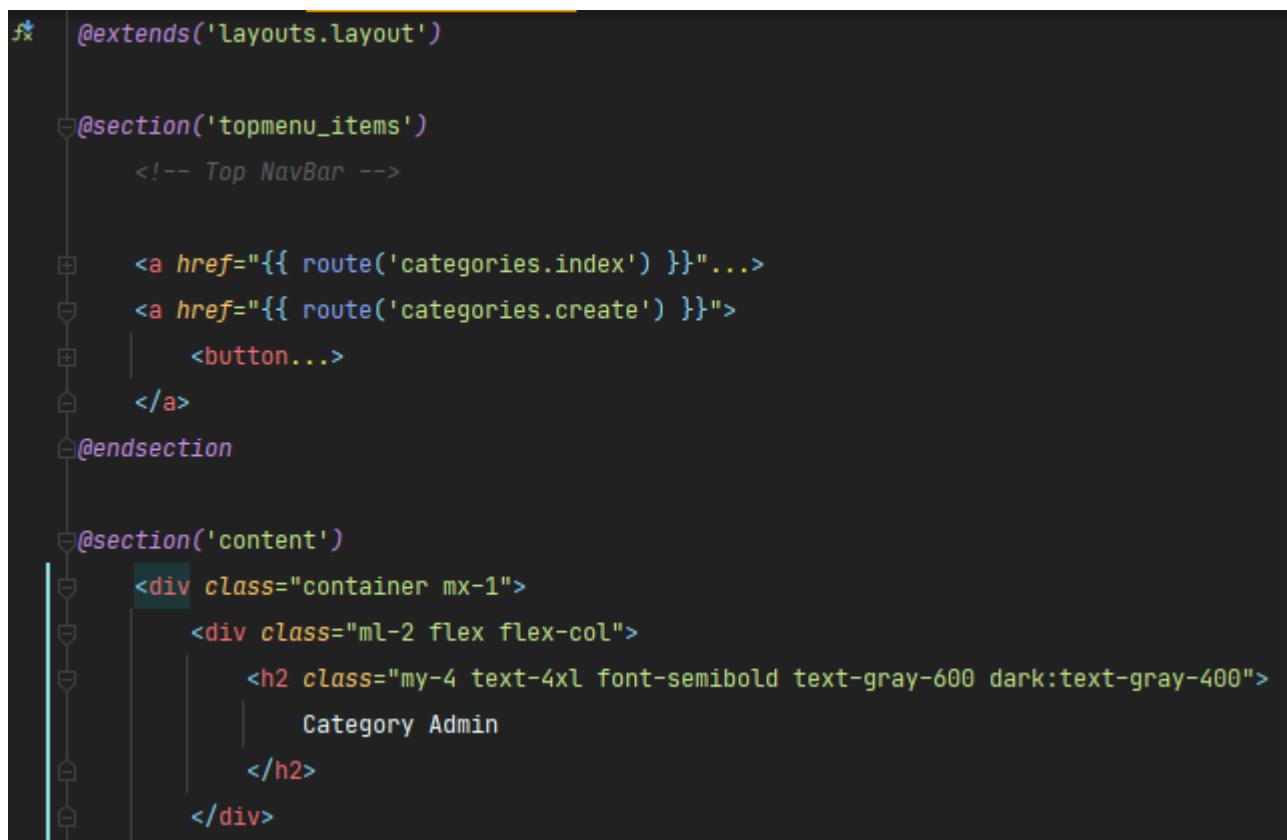
```
/** Display the specified resource. ...*/
public function show(Category $category)
{
    //
}
```

Wat Laravel doet, is dus de categorie met de id in de URL uit de database ophalen en in \$category zetten. Dit betekent dat we dus de categorie niet meer hoeven op te halen maar dat we dit meteen richting de view mee kunnen sturen.

```
/** Display the specified resource. ...*/
public function show(Category $category)
{
    return view('admin.categories.show', compact('var_name: 'category'));
}
```

Voor de view maken we een show.blade.php aan in de map categories (mag ondertussen verwachten dat je weet waar dit is). Om ervoor te zorgen dat het laten zien van 1 categorie een beetje netjes is, gebruik ik een card voorbeeld van tailwind. (<https://tailwindcomponents.com/component/user-card>)

Je ziet, dat ik vaak voorbeelden van internet gebruik met Tailwind en ze iets aan pas om te gebruiken bij een project. Ga niet opnieuw het wiel uitvinden met Tailwind als de focus ligt bij het leren van Laravel !



```
@extends('layouts.layout')

@section('topmenu_items')
    <!-- Top NavBar -->

    <a href="{{ route('categories.index') }}">...
        <a href="{{ route('categories.create') }}>
            <button...>
        </a>
    </a>
@endsection

@section('content')
    <div class="container mx-1">
        <div class="ml-2 flex flex-col">
            <h2 class="my-4 text-4xl font-semibold text-gray-600 dark:text-gray-400">
                Category Admin
            </h2>
        </div>
    </div>

```

De layout wordt extended, de section topmenu\_items moet er nog steeds bij, en daarna de section content. Daar binnen voeg ik de card toe, al heb ik het nu al redelijk uitgekleed in code.

```
33      <!-- component -->
34      <div class="max-w-sm bg-white shadow-lg rounded-lg overflow-hidden my-4">
35          
37              <h1 class="mx-3 text-white font-semibold text-lg">Category</h1>
38          </div>
39          <div class="py-4 px-6">
40              <h1 class="text-2xl font-semibold text-gray-800">{{ $category->name }}</h1>
41              <p class="py-2 text-lg text-gray-700"></p>
42          </div>
43      </div>
44
45      </div>
46  @endsection
```

We hebben \$category mee gestuurd met de layout, dus de naam van de categorie kunnen we netjes in de card plaatsen. Let wel op dat je netjes je section afsluit. Als we het dan uitproberen in de browser (url even handmatig gedaan) zien we dit:

The screenshot shows a Laravel application's admin interface. The URL in the browser is `laravelv1/admin/categories/1`. On the left, there's a sidebar with a user profile picture, the text "Hello Koningstein Admin", and a "Category Admin" link. The main content area has a header "Category Admin". Below it is a card with a blue-tinted image of a woman wearing a hoodie and glasses. A dark overlay on the card displays the word "Category" and the name "Mrs. Nina McLaughlin". Above the card, there are navigation links "Index" and "Create".

Op zich voor nu even een net resultaat. Natuurlijk gaan we straks dingen krijgen die uitgebreider moeten in een show, waardoor zo'n card al handig is om te hebben. Het belangrijkste nu was om te zien hoe we door de model binding al de informatie binnen hebben en meteen kunnen gebruiken.

Nu is het alleen nog netjes om vanuit de index een link te maken naar de show. Want nu moesten we zelf de id in de url zetten. Natuurlijk gebruiken we weer de named route hiervoor.

```
<tbody class="bg-white divide-y divide-gray-200">
@foreach($categories as $category)
<tr>
    <td class="px-6 py-4 whitespace nowrap text-sm font-medium text-gray-900">
        {{ $category->name }}
    </td>
    <td class="px-6 py-4 whitespace nowrap text-sm text-gray-500">
        <a href="{{ route('categories.show', ['category' => $category->id]) }}">Details</a>
    </td>
    <td class="px-6 py-4 whitespace nowrap text-sm text-gray-500">
        Edit
    </td>
    <td class="px-6 py-4 whitespace nowrap text-sm text-gray-500">
        Delete
    </td>
</tr>
@endforeach

<!-- More items... -->
</tbody>
```

Als je naar de index gaat in de browser merk je dat je nu op details kan klikken en naar de show gaat van die categorie. De show is dan nu ook klaar.

### 3.12. Opdracht 7: show

Maak de show voor projects. Zorg dat je de show ziet binnen je eigen masterpage. Op de show moeten de volgende onderdelen te zien zijn:

- Project id
- Naam van het project
- Beschrijving van het project
- Datum van het aanmaken van het project

De view van de show moet in de volgende map zitten en ook de correcte naam hebben:

- Resources/views/admin/projects/show.blade.php

Je moet gebruik maken van de named routes die bij een resource controller aan gemaakt worden.

#### Opdracht controle

Als je de opdracht af hebt en gebruik hebt gemaakt van de opdrachten repository, kan je het volgende commando gebruiken om te controleren of alles goed werkt:

```
.\vendor\bin\pest --group=Opdracht7
```

### 3.13. Edit

Het doel van de edit methode is een ingevuld formulier te hebben die richting de update gestuurd kan worden. Een formulier hebben we al is gemaakt bij de create. Deze code zullen we ook kopiëren om in de edit view te gaan gebruiken. Maar eerst de methode zelf.

Als je goed kijkt zie je bij de edit weer een Model Binding. Dit betekent dat het product id als het goed is in de url staat. Dit klopt ook:

GET	/photos/{photo}/edit	edit	photos.edit
-----	----------------------	------	-------------

We hebben dus de informatie van het product beschikbaar en hoeven daardoor alleen de informatie naar de view te sturen.

```
/** Show the form for editing the specified resource. ...*/
public function edit(Category $category)
{
    return view('admin.categories.edit', compact('category'));
}
```

We maken dan een *edit.blade.php* in de categories map aan en kopiëren wat we bij de *create.blade* hebben naar de edit. Dit betekent dat de lay-out daarvan hetzelfde is. De error messages al in de lay-out staan en er ook een menu in zit voor categories.

Deze code gaan we dan ook aanpassen voor de edit pagina. De error message hoeft niet aangepast te worden. Het menu is wel prettig dat dit constant blijft. Het formulier wordt ook op verschillende plekken aangepast. De action moeten we goed op letten. Als we kijken naar de resource controller moet het formulier naar de updatemethode gaan.

PUT/PATCH	/photos/{photo}	update	photos.update
-----------	-----------------	--------	---------------

Bij de action moet dus nog de id meegegeven worden. Verder moeten we het versturen met een put of patch.

```
<form id="form" class="bg-white shadow-md rounded-lg px-8 pt-6 pb-8 mb-4"
      action="{{ route('categories.update', ['category' => $category->id]) }}" method="POST">
    @method('PUT')
    @csrf
    <div class="mb-4">
        <label class="block text-gray-700 text-sm font-bold mb-2" for="name">
            Name
        </label>
```

Als je goed kijkt sturen we het formulier nu met de method PUT. Deze methode heb je nog nooit gezien, maar wordt door Laravel gebruikt om de verschillen duidelijk te maken. Als je naar de bron van het formulier gaat kijken in je browser zal je dit zien.

```

<form id="form" class="bg-white shadow-md rounded-lg px-8 pt-6 pb-8 mb-4"
    action="https://laravelv1/admin/categories/1" method="POST">
    <input type="hidden" name="_method" value="PUT">           <input type="hidden" name="_token" value="EEtRbv1X7VcqKcuDKfRkprlW0apg18LvDMUDJ08A2">
        <label class="block text-gray-700 text-sm font-bold mb-2" for="name">
            Name
        </label>

```

Je ziet dus dat de method in de form POST is, maar dat er een input is met name \_method met de waarde PUT. Dit zorgt er dan voor dat het niet naar de method store gaat, maar naar de update.

De input pakken we ook aan. Bij de value zorgen we ervoor dat de als er een oude waarde (gewijzigd, maar niet door validatie heen komt) laten zien. Mocht er geen wijziging (nog) zijn, laten we de waarde die in de database staan zien.

```

<input
    class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
    focus:outline-none focus:shadow-outline @error('name') border-red-500 @enderror" name="name" id="name"
    value="{{ old('name', $category->name) }}" type="text" required>
</div>
<div class="flex items-center justify-between">
    <button id="submit"
        class="bg-green-500 hover:bg-green-700 text-white font-bold py-2 px-4 rounded
        focus:outline-none focus:shadow-outline" type="submit">Edit
    </button>
</div>

```

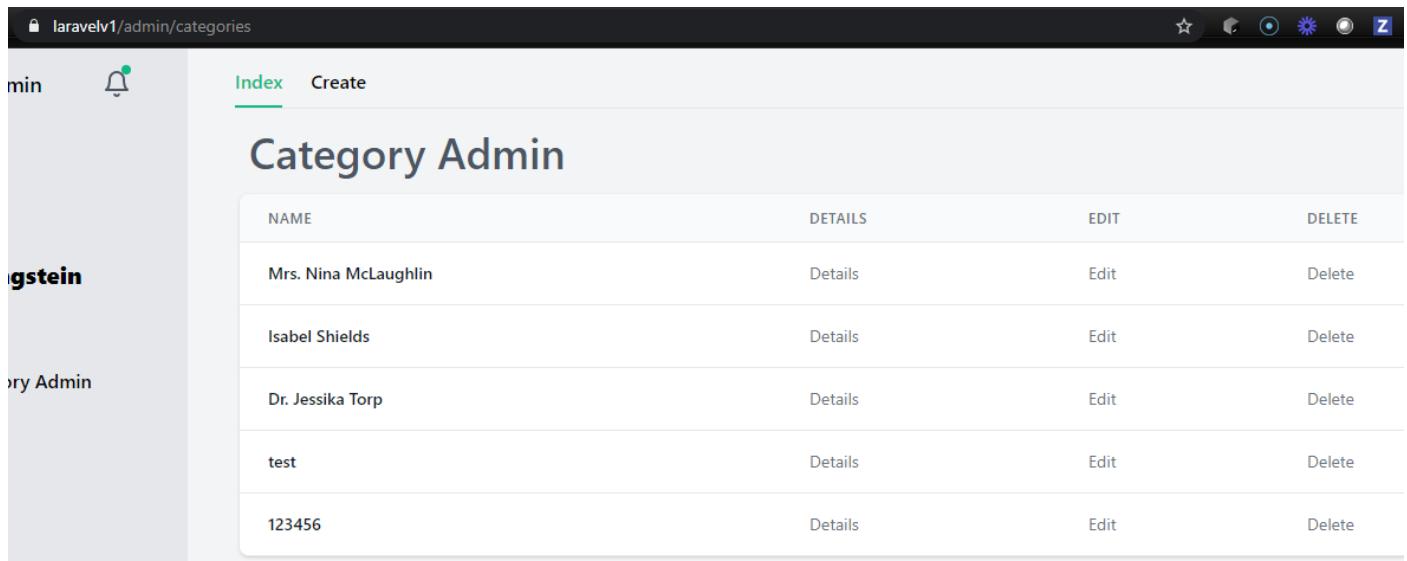
Als we handmatig de url in typen om een edit te doen zien we dit:

Een ingevuld formulier waarbij al hetgeen wat in rij 1 staat in de database ingevuld is, omdat we id 1 in de url hebben staan. Het werkt !

Om dit te zien zou je handmatig de url moeten veranderen nu. We zullen dus vanuit de index iets moeten maken om naar de edit toe te gaan. Dit zal net als bij de show natuurlijk moeten, alleen dan met een andere named route.

```
<tbody class="bg-white divide-y divide-gray-200">
@foreach($categories as $category)
    <tr>
        <td class="px-6 py-4 whitespace nowrap text-sm font-medium text-gray-900">
            {{ $category->name }}
        </td>
        <td class="px-6 py-4 whitespace nowrap text-sm text-gray-500">
            <a href="{{ route('categories.show', ['category' => $category->id]) }}>Details</a>
        </td>
        <td class="px-6 py-4 whitespace nowrap text-sm text-gray-500">
            <a href="{{ route('categories.edit', ['category' => $category->id]) }}>Edit</a>
        </td>
        <td class="px-6 py-4 whitespace nowrap text-sm text-gray-500">
            Delete
        </td>
    </tr>
@endforeach
```

Het overzicht ziet er nog steeds netjes uit:



The screenshot shows a web browser window with the URL 'laravelv1/admin/categories'. On the left, there's a sidebar with a user profile picture, a notification bell icon, and menu items 'min', 'Category Admin', and 'Category Admin'. The main content area has a header 'Category Admin' with 'Index' and 'Create' buttons. Below is a table with the following data:

NAME	DETAILS	EDIT	DELETE
Mrs. Nina McLaughlin	Details	Edit	Delete
Isabel Shields	Details	Edit	Delete
Dr. Jessika Torp	Details	Edit	Delete
test	Details	Edit	Delete
123456	Details	Edit	Delete

De gehele edit is nu klaar. We kunnen vanuit de index een categorie kiezen om te wijzigen. Als we een categorie hebben gekozen zien we een netjes ingevuld formulier. Let wel op, dat het in de gekozen layout niet goed te zien is dat het linkjes zijn geworden. Voor nu geeft dit helemaal niet, maar voor een gebruiker zal dit wel straks duidelijk moeten zijn.

### 3.14. Update

Als je gaat kijken naar de update method lijkt deze heel erg op de store method. We vangen een formulier op, moeten een validatie doen, en dan doen we een update.

Dit is op dit moment de update method:

```
/** Update the specified resource in storage. ...*/
public function update(Request $request, Category $category)
{
    //
}
```

Als argumenten in de methode zien we 2 dingen staan. Je ziet de Request \$request staan, wat hetzelfde is al bij de store. Je zou nu moeten weten dat dit voor validatie is die we ook hier weer gaan gebruiken. Daarnaast zie je Category \$category staan. Dit is hetzelfde als bij de show en edit, waar we dus al van een bepaalde id gegevens over de categorie ophalen.

We krijgen een request binnen vanuit het formulier. Tevens staat er nog steeds een id in de url, dus hierdoor hebben we met behulp van de Model Binding ook meteen het product tot onze beschikking. Denk nog even aan de edit form die al gemaakt is.

```
<form id="form" class="bg-white shadow-md rounded-lg px-8 pt-6 pb-8 mb-4"
    action="{{ route('categories.update', ['category' => $category->id]) }}" method="POST">
    @method('PUT')
    @csrf
```

Hier geven we die id in de url mee, waarvoor we dus de categorie krijgen in \$category.

Als eerst gaan we dan ook de update uitvoeren.

```
/** Update the specified resource in storage. ...*/
public function update(CategoryUpdateRequest $request, Category $category)
{
    $category->name = $request->name;
    $category->save();
}
```

Let er even op dat we dus de categorie al hebben in \$category. Wat we bij de store moesten doen met \$category = new Category(); hoeft nu dus niet, omdat we een bestaande aanpassen.

De rest is hetzelfde, al voeren we nu dus geen insert uit maar een update.

Voor de validatie moeten we even goed kijken. Soms kan je voor de store en update methodes dezelfde validatie gebruiken. Voor de store hadden we dit:

```
/** Get the validation rules that apply to the request. ...*/
public function rules()
{
    return [
        'name' => 'required|string|unique:categories|min:5|max:75'
    ];
}
```

Hier staat in dat elke categorie unique moet zijn. Dit betekent, dat als we bij het edit formulier zitten en niks wijzigen maar wel op de update knop drukken, we een error gaan krijgen. Dat hoort niet, dus hebben we een andere Request nodig, de CategoryUpdateRequest.

```
D:\Wamp\sites\laravelv1>php artisan make:request CategoryUpdateRequest
Request created successfully.
```

Bij de nieuwe request er weer even voor zorgen dat authorize op true staat. Verder kunnen we de rules bijna overnemen van de store. De uitbreiding zit hem bij de validatie van unique. Op deze manier zorg je ervoor dat als de name gelijk blijft, deze uitgezonderd wordt waardoor je door de validatie heen komt.

We halen vanuit de route op met welke categorie we bezig zijn. Dan geven we bij unique aan bij welke id de naam van de categorie overeen mag komen, zodat het niet erg is als we op edit klikken terwijl dezelfde naam er nog steeds staat.

```
class CategoryUpdateRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request. ...
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request. ...
     */
    public function rules()
    {
        $category = $this->route('param', 'category');

        return [
            'name' => 'required|string|min:5|max:75|unique:categories,name,' . $category->id
        ];
    }
}
```

In de update method zorgen we er dan voor dat de juiste Request er staat in het argument, zodat de validatie uitgevoerd wordt. Dit doen we met CategoryUpdateRequest.

```
/** Update the specified resource in storage. ...*/
public function update(CategoryUpdateRequest $request, Category $category)
{
    $category->name = $request->name;
    $category->save();

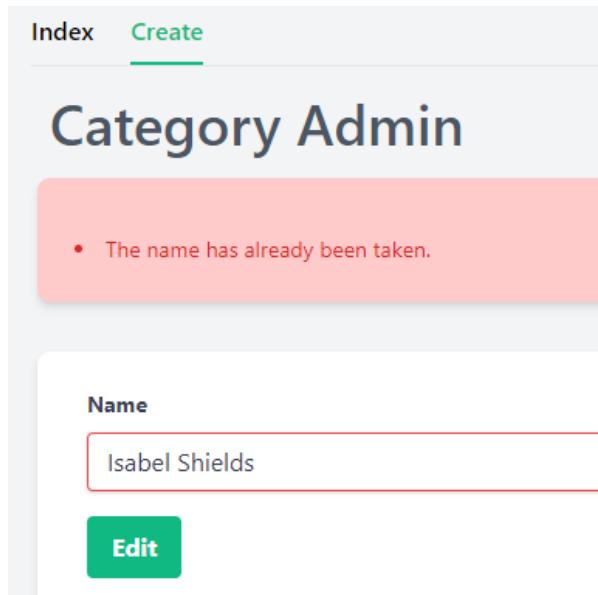
    return redirect()->route('categories.index')->with('status', 'Categorie geupdate');
}
```

Let wel even op dat je nog even bij use kijkt of de class er bij staat:

```
use App\Http\Controllers\Controller;
use App\Http\Requests\CategoryStoreRequest;
use App\Http\Requests\CategoryUpdateRequest;
use App\Models\Category;
use Illuminate\Http\Request;
```

Naast de goede request voor validatie, is het goed om na de wijziging terug te gaan naar de index met een bericht dat de categorie geupdate is.

Nu is alles klaar voor de update. Als eerst kunnen we testen of 2 keer zelfde naam bij categorie kan staan:



The screenshot shows a web application interface for managing categories. At the top, there are two buttons: 'Index' and 'Create'. Below this, the title 'Category Admin' is displayed. A prominent red callout box contains the error message 'The name has already been taken.' Below the message, there is a form field labeled 'Name' with the value 'Isabel Shields' entered. At the bottom left of the form area, there is a green button labeled 'Edit'.

We krijgen netjes een melding dat de naam al gebruikt word.

Als we dan een naam gebruiken die wel kan:

The screenshot shows a web application titled "Category Admin". At the top left are links for "Index" and "Create". A green banner at the top says "Categorie geupdate". Below is a table with columns "NAME", "DETAILS", and "EDIT". One row is shown, containing "Mrs. Nina McLaughlin2", a link "Details", and a link "Edit".

NAME	DETAILS	EDIT
Mrs. Nina McLaughlin2	<a href="#">Details</a>	<a href="#">Edit</a>

De categorie wordt netjes geupdate en je ziet de melding verschijnen.

### 3.15. Opdracht 8: edit & update

Maak binnen de resource controller van projects de edit en update methode en zorg dat het binnen je eigen masterpage werkt.

Op de update zal een formulier beschikbaar moeten zijn die post (+ PUT/PATCH) naar de update methode.

De view van de edit moet in de volgende map zitten en ook de correcte naam hebben:

- Resources/views/admin/projects/edit.blade.php

Als je het formulier hebt verstuurd kom je bij de update methode. Bij de validatie wordt gecontroleerd op de volgende voorwaarden:

- Naam van het project is ingevuld
- De naam is minimaal 5 karakters, maximaal 45 karakters
- De projectnaam moet uniek zijn, maar de bestaande naam van het project die je wilt wijzigen moet toegestaan zijn.
- De beschrijving moet ingevuld zijn. Hier is geen minimale of maximale waarde

Als er niet aan de validatie wordt voldaan, moet het zichtbaar in het formulier zijn welke onderdelen fout zijn. De foutmeldingen moeten binnen de lay-out op het scherm komen.

Als het aan de validatie voldoet moet het worden opgeslagen in de tabel projects. Er zal een redirect moeten komen naar de index met een melding dat het project is opgeslagen.

Je moet gebruik maken van de named routes die bij een resource controller aan gemaakt worden.

### Opdracht controle

Als je de opdracht af hebt en gebruik hebt gemaakt van de opdrachten repository, kan je het volgende commando gebruiken om te controleren of alles goed werkt:

```
.\vendor\bin\pest --group=Opdracht8
```

### 3.16. Delete

In Laravel zit eigenlijk geen delete methode, maar een destroy methode. Hoe deze standaard wordt geschreven is dat een item meteen wordt verwijderd, zonder bevestiging. Zelf vind ik dit minder prettig, want zou graag eerst willen zien wat ik verwijder, zodat als ik de verkeerde heb aangeklikt ik nog terug kan. Je kan ervoor kiezen om met bijv javascript toch een response te hebben, maar voor nu wil ik het juist bij de basics van Laravel houden en kijken hoe we zo iets kunnen oplossen met de kennis die je net hebt geleerd.

Hierdoor kies ik ervoor om nog een extra methode in de controller te maken, namelijk de delete methode. Nog even welke acties we standaard hebben:

#### Actions Handled By Resource Controller

Verb	URI	Action	Route Name
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	destroy	photos.destroy

Als eerst gaan we de index.blade wijzigen, zodat we per categorie naar een delete kunnen. (net zoals de edit)

```
<tbody class="bg-white divide-y divide-gray-200">
@foreach($categories as $category)
    <tr>
        <td class="px-6 py-4 whitespace nowrap text-sm font-medium text-gray-900">
            {{ $category->name }}
        </td>
        <td class="px-6 py-4 whitespace nowrap text-sm text-gray-500">
            <a href="{{ route('categories.show', ['category' => $category->id]) }}">Details</a>
        </td>
        <td class="px-6 py-4 whitespace nowrap text-sm text-gray-500">
            <a href="{{ route('categories.edit', ['category' => $category->id]) }}">Edit</a>
        </td>
        <td class="px-6 py-4 whitespace nowrap text-sm text-gray-500">
            <a href="{{ route('categories.delete', ['category' => $category->id]) }}">Delete</a>
        </td>
    </tr>
@endforeach
```

Om ervoor te zorgen dat deze link gaat werken, zullen we in de routes wat moeten aanpassen. Dit valt namelijk niet onder de resource route die we hebben staan voor categories. In de index gebruik ik nu namelijk een named route categories.delete, maar deze bestaat nog helemaal niet. Zou ik even naar de index in de browser nu gaan, krijgen we een foutmelding.

D:\Wamp\sites\laravelv1\

Symfony\Component\Routing\Exception\RouteNotFoundException  
Route [categories.delete] not defined. (View:  
D:\Wamp\sites\laravelv1\resources\views\admin\categories\index.blade.php)

<https://laravelv1/admin/categories>

Om ervoor te zorgen dat de named route die we willen gaan werken, zullen we de routes moeten aanpassen. De routes vinden we in de map routes/web.php

Als we de resource route willen uitbreiden moet dit altijd boven de resource route.

```
17     Route::get( uri: '/', function () {
18         return view( view: 'welcome');
19     });
20
21     Route::get( uri: 'admin/categories/{category}/delete', [CategoryController::class, 'delete'])
22         ->name( name: 'categories.delete');
23     Route::resource( name: '/admin/categories', controller: CategoryController::class);
24
25     Route::get( uri: '/dashboard', function () {
```

Op regel 21-22 is de nieuwe route aangemaakt. Bij de url wordt aangegeven dat als we naar 'admin/categories/1/delete' bijvoorbeeld gaan, de delete methode van de CategoryController ingeladen moet worden, waarbij de route de naam krijgt 'categories.delete'. Bij een resource route krijgen we automatisch de named routes, maar bij een get is dit niet het geval. Toch is het altijd handig om named routes te hebben.

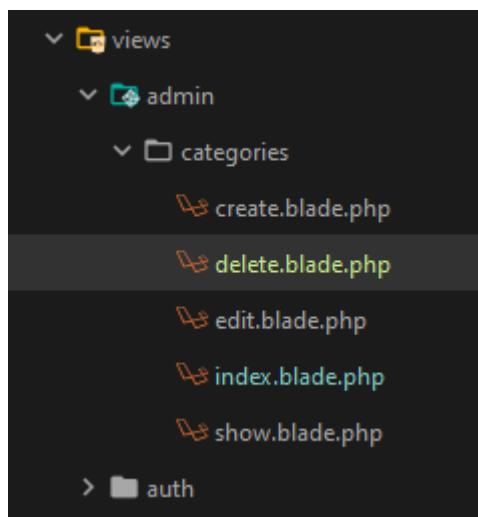
Nu gaan we de delete method toevoegen aan de controller. Ik zet het automatisch tussen de update en destroy method, zodat er een logische volgorde in zit. De code lijkt heel erg op die van de edit. Zelf zorg ik ook dat de phpdoc ook meteen goed staat, maar phpdoc zal nu niet uitgebreid behandeld worden.

```
 /**
 * Show the form for deleting the specified resource.
 *
 * @param \App\Models\Category $category
 * @return \Illuminate\Http\Response
 */
public function delete(Category $category)
{}
```

We gebruiken meteen de Model Binding om ervoor te zorgen dat we de id uit de url kunnen gebruiken en dat het product meteen is opgehaald. Hierdoor kan meteen de view aangeroepen worden.

```
    /** Show the form for deleting the specified resource. ....*/
    public function delete(Category $category)
    {
        return view( view: 'admin.categories.delete', compact( var_name: 'category'));
    }
```

Voor de view maken we een delete.blade.php aan in de categories map. We kunnen alles van de edit.blade kopieren omdat er heel veel overeenkomsten zijn.



Bij het formulier wijzigen we de method naar DELETE. In de action zorgen we dat we naar de juiste named route gaan, namelijk categories.destroy.

```
<form id="form" class="bg-white shadow-md rounded-lg px-8 pt-6 pb-8 mb-4"
    action="{{ route('categories.destroy', [ 'category' => $category->id]) }}" method="POST">
    @method('DELETE')
    @csrf
```

Om ervoor te zorgen dat je goed ziet dat we niet in de edit zitten, maar bij de delete, worden de inputs disabled. Er is dan ook geen old methode nodig meer, alleen maar de data die nu in de database staat. Daarnaast zorgen we dat de knop de tekst Delete heeft.

```
<div>
    <input
        class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
        focus:outline-none focus:shadow-outline" name="name" id="name"
        value="{{ $category->name }}" type="text" disabled>
</div>
<div class="flex items-center justify-between">
    <button id="submit"
        class="bg-green-500 hover:bg-green-700 text-white font-bold py-2 px-4 rounded
        focus:outline-none focus:shadow-outline" type="submit">Delete
    </button>
</div>
```

Als we dan gaan kijken of het werkt, zal je vanuit de index per regel een delete link hebben. Vanuit daar kom je bij een formulier.

Als we nu vanuit de index bij id = 1 naar de delete gaan zien we een net formulier met de waarde in het formulier, waarbij een knop Delete staat. De input is ook disabled. Tot zover werkt dus alles.

The screenshot shows a web browser window with the URL 'laravelv1/admin/categories/1/delete'. The page title is 'Category Admin'. On the left, there's a sidebar with 'Category Admin' and a notification bell icon. The main content area has tabs for 'Index' and 'Create', with 'Index' being active. Below the tabs, there's a form field labeled 'Name' containing 'Mrs. Nina McLaughlin2'. At the bottom of the form is a green button labeled 'Delete'.

### 3.17. Destroy

Nu het formulier klaar is kunnen we de method destroy gaan invullen.

Omdat we de categorie beschikbaar hebben vanwege de model binding, kunnen we meteen de categorie verwijderen. Daarna zorgen we ervoor dat we bij de index weer uitkomen met een bericht dat de categorie is verwijderd.

```
/** Remove the specified resource from storage. ...*/
public function destroy(Category $category)
{
    $category->delete();
    return redirect()->route('categories.index')->with('message', 'Categorie verwijderd');
}
```

Als je het test zal je zien dat we netjes een bericht krijgen. De categorie met id = 1 heb ik ook kunnen verwijderen en staat niet meer in het overzicht.

NAME	DETAILS	EDIT	DELETE
Isabel Shields	Details	Edit	Delete
Dr. Jessika Torp	Details	Edit	Delete
test	Details	Edit	Delete
123456	Details	Edit	Delete

Nu is de gehele CRUD voor categories klaar. We hebben een overzicht, kunnen toevoegen, wijzigen en verwijderen. Let wel op, dat dit nog een makkelijke crud was omdat we maar 1 veld hadden en niks met relaties te maken hadden. Dit gaat wel lastiger worden, maar de principes worden gebruikt met wat aanvullingen zodat ook alles met de relaties zal werken.

### 3.19. Opdracht 9: delete & destroy

Maak binnen de resource controller van projects de delete en destroy methode en zorg dat het binnen je eigen masterpage werkt.

Op de delete zal een formulier beschikbaar moeten zijn die post (+ DELETE) naar de destroy methode. Alle inputs moeten disabled zijn.

De view van de delete moet in de volgende map zitten en ook de correcte naam hebben:

- Resources/views/admin/projects/delete.blade.php

Als je het formulier hebt verstuurd kom je bij de destroy methode.

Bij de destroy methode zal het project worden verwijderd. Er zal een redirect moeten komen naar de index met een melding dat het project is verwijderd. Je moet gebruik maken van de named routes die bij een resource controller aan gemaakt worden.

#### Opdracht controle

Als je de opdracht af hebt en gebruik hebt gemaakt van de opdrachten repository, kan je het volgende commando gebruiken om te controleren of alles goed werkt:

```
.\vendor\bin\pest --group=Opdracht9
```

## 4. Rollen en Permissies

Nu we de eerste CRUD af hebben, zullen we wat aan de beveiliging moeten gaan doen. Op een website is het natuurlijk niet de bedoeling dat iedereen overal bij kan. Hiervoor gaan aan de slag met rollen en permissies.

Om ervoor te zorgen dat we makkelijk met rollen kunnen werken op onze website, gaan we gebruik maken van een package. Laravel-permission is gemaakt door het bedrijf Spatie, wat in Antwerpen zit. Laravel-permission wordt al een hele tijd erg veel gebruikt en het bedrijf Spatie geeft ook wel elke Laracon seminar een presentatie, waardoor ze erg bekend zijn in de Laravel communitie.

Nu is het leuke van dit bedrijf, dat ze veel packages hebben die PostcardWare hebben. Als de site die je maakt werkelijk online gaat en je gebruikt hun package, vragen ze alleen om een kaartje naar hun bedrijf.

GitHub: <https://github.com/spatie/laravel-permission>

Documentatie: <https://spatie.be/docs/laravel-permission/v4/introduction>

## Associate users with permissions and roles

### Sponsor



**Auth0**

If you want to quickly add authentication and authorization to Laravel projects, feel free to check Auth0's Laravel SDK and free plan at <https://auth0.com/developers>.

packagist v4.0.1

Run Tests

passing

downloads 7M

## Documentation, Installation, and Usage Instructions

See the [DOCUMENTATION](#) for detailed installation and usage instructions.

### What It Does

This package allows you to manage user permissions and roles in a database.

Once installed you can do stuff like this:

```
// Adding permissions to a user
$user->givePermissionTo('edit articles');

// Adding permissions via a role
$user->assignRole('writer');

$role->givePermissionTo('edit articles');
```

Om deze package binnen te halen gaan we de command line gebruiken. We kunnen dan met composer de package binnen halen met:

*Composer require spatie/laravel-permission*

```
D:\Wamp\sites\laravelv1>Composer require spatie/laravel-permission
Using version ^4.0 for spatie/laravel-permission
./composer.json has been updated
Running composer update spatie/laravel-permission
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
- Locking spatie/laravel-permission (4.0.1)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
- Downloading spatie/laravel-permission (4.0.1)
- Installing spatie/laravel-permission (4.0.1): Extracting archive
Generating optimized autoload files
```

Dan gaan we naar de configuratie om de package goed in Laravel werkend te krijgen. Hiervoor zullen we de package in de config moeten zetten. Dit doe je in *config/app.php*

We zetten hierin de regel: *Spatie\Permission\PermissionServiceProvider::class*,

```
162             Illuminate\Validation\ValidationServiceProvider::class,
163             Illuminate\View\ViewServiceProvider::class,
164
165         /*
166          * Package Service Providers...
167          */
168         Spatie\Permission\PermissionServiceProvider::class,
169
170         /*
171          * Application Service Providers...
172          */
173         App\Providers\AppServiceProvider::class,
174         App\Providers\AuthServiceProvider::class,
```

Nu kunnen we de package publishen in ons project. De migration en de permission bestanden worden daarmee op hun plek gezet. We doen dit met:

```
php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider"
```

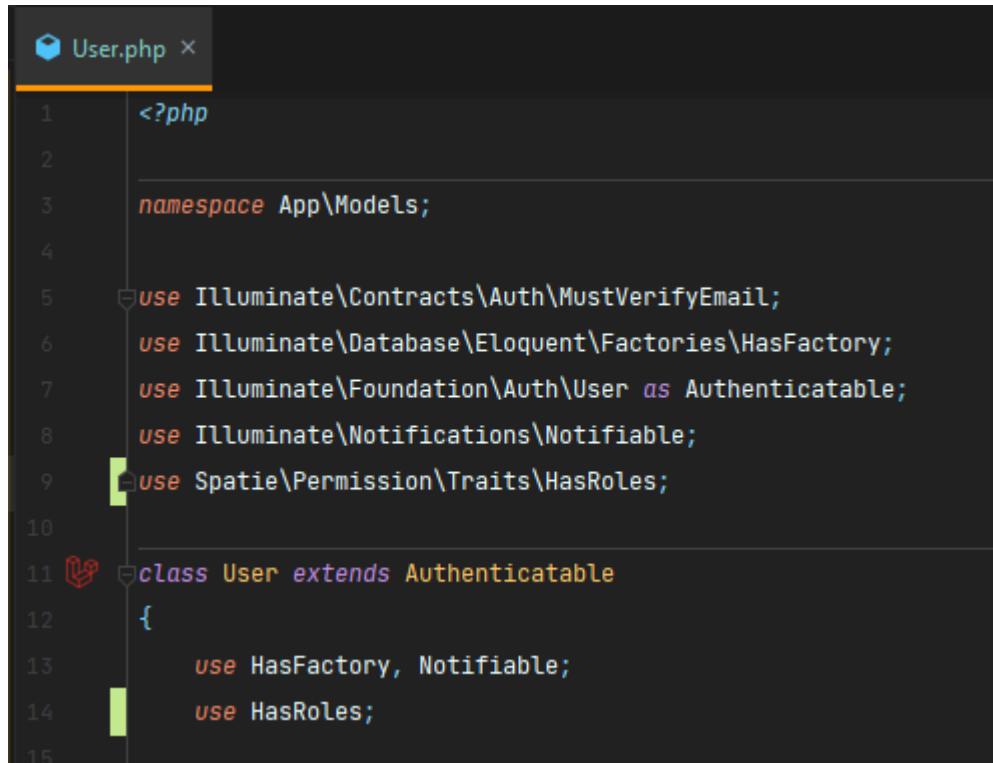
```
D:\Wamp\sites\laravelv1>php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider"

Copied File [\vendor\spatie\laravel-permission\config\permission.php] To [\config\permission.php]
Copied File [\vendor\spatie\laravel-permission\database\migrations\create_permission_tables.php.stub] To [\database\migrations\2021_01_200405_create_permission_tables.php]
Publishing complete.
```

Deze zorgt dat er een config bestand komt: config/permission.php

Verder krijgen we ook een migration erbij zodat we straks de rollen en permissies in de database kwijt kunnen.

De permissions zijn voor de user. Hierdoor moeten we dit netjes angeven bij de model. Dit doen we door in de User model het volgende toe te voegen: *use HasRoles;*



```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Contracts\Auth\MustVerifyEmail;
6 use Illuminate\Database\Eloquent\Factories\HasFactory;
7 use Illuminate\Foundation\Auth\User as Authenticatable;
8 use Illuminate\Notifications\Notifiable;
9 use Spatie\Permission\Traits\HasRoles;
10
11 class User extends Authenticatable
12 {
13     use HasFactory, Notifiable;
14     use HasRoles;
15 }
```

Als het goed is komt regel 9 er vanzelf bij, anders handmatig de regel toevoegen:

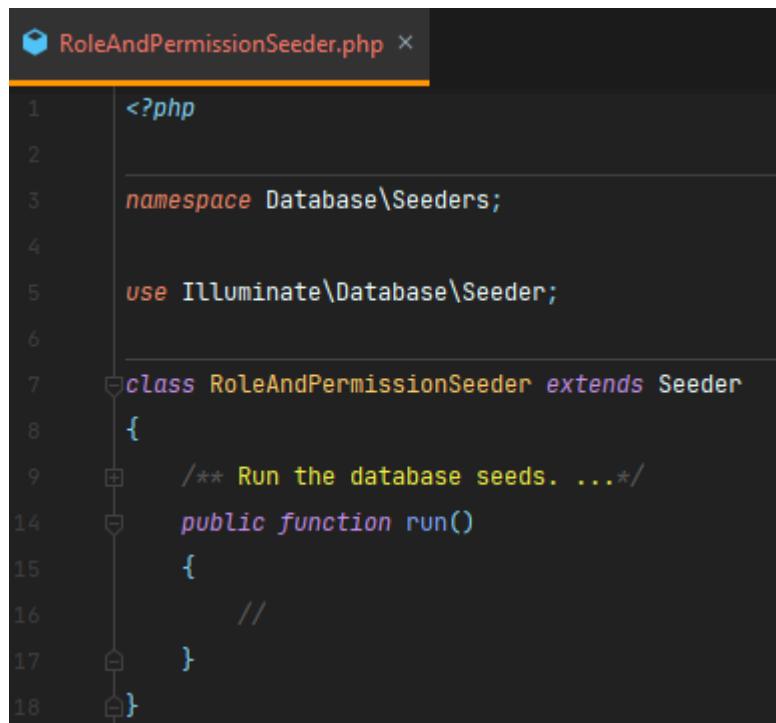
```
use Spatie\Permission\Traits\HasRoles;
```

## 4.1. Role and Permission Seeder

Om nu ervoor te zorgen dat dit allemaal gaat werken, gaan we een aantal rollen aanmaken. Dit doen we in een seeder: `RoleAndPermissionSeeder.php`

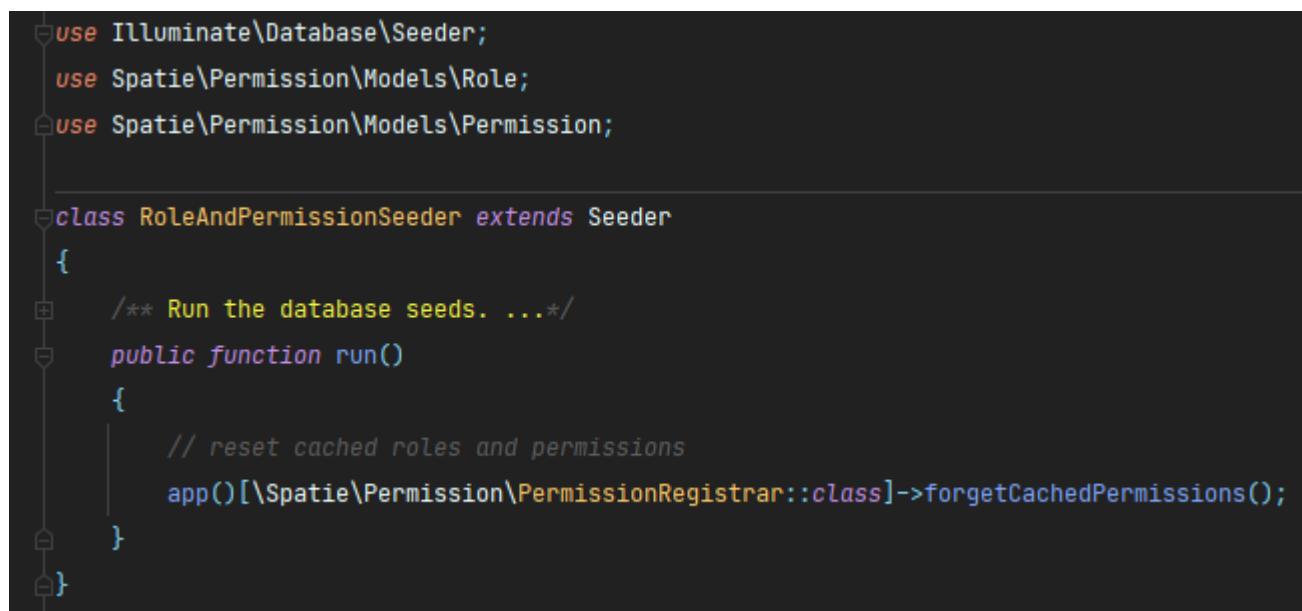
```
D:\Wamp\sites\laravelv1>php artisan make:seeder RoleAndPermissionSeeder  
Seeder created successfully.
```

Hierdoor staat de seeder weer klaar.



```
RoleAndPermissionSeeder.php ×  
1 <?php  
2  
3 namespace Database\Seeders;  
4  
5 use Illuminate\Database\Seeder;  
6  
7 class RoleAndPermissionSeeder extends Seeder  
8 {  
9     /** Run the database seeds. ... */  
10    public function run()  
11    {  
12        //  
13    }  
14}
```

Met de seeder gaan we de database dusdanig vullen dat we een aantal permisies hebben en een aantal rollen. Om ervoor te zorgen dat de seeder deze functionaliteit kan gebruiken voegen we netjes de model toe van Role en Permission met `use Spatie\Permission\Models\Role;` en `use Spatie\Permission\Models\Permission;`:



```
use Illuminate\Database\Seeder;  
use Spatie\Permission\Models\Role;  
use Spatie\Permission\Models\Permission;  
  
class RoleAndPermissionSeeder extends Seeder  
{  
    /** Run the database seeds. ... */  
    public function run()  
    {  
        // reset cached roles and permissions  
        app()[\Spatie\Permission\PermissionRegistrar::class]->forgetCachedPermissions();  
    }  
}
```

Verder zorgen we dat zodra de seed uitgevoerd wordt, we netjes de cache legen. Laravel gebruikt eigenlijk altijd een cache. Het is dan slim, zeker wat betreft beveiliging, om dan de oude settings weg te halen voordat je de nieuwe erin zet.

Nu gaan we hieronder de permissies toevoegen. Dit zijn allemaal mogelijke permissies voor onze functionaliteit. We hebben op dit moment de crud voor categorie. Zodra we meer functionaliteit maken gaan we hiervoor ook de permissies toevoegen.

```
/** Run the database seeds. ...*/
public function run()
{
    // reset cached roles and permissions
    app()[\Spatie\Permission\PermissionRegistrar::class]->forgetCachedPermissions();

    Permission::create(['name' => 'index category']);
    Permission::create(['name' => 'show category']);
    Permission::create(['name' => 'create category']);
    Permission::create(['name' => 'edit category']);
    Permission::create(['name' => 'delete category']);
}
```

Op deze manier kunnen we verschillende permissies aan een persoon geven, bijvoorbeeld dat je wel een categorie mag aanmaken, maar niet mag wijzigen of verwijderen. Om straks dit niet voor elke persoon te hoeven doen, gaan we Rollen aanmaken. De rollen krijgen dan rechten over bepaalde permissies.

```
/** Run the database seeds. ...*/
public function run()
{
    // reset cached roles and permissions
    app()[\Spatie\Permission\PermissionRegistrar::class]->forgetCachedPermissions();

    // permissions for category CRUD
    Permission::create(['name' => 'index category']);
    Permission::create(['name' => 'show category']);
    Permission::create(['name' => 'create category']);
    Permission::create(['name' => 'edit category']);
    Permission::create(['name' => 'delete category']);

    // customer role
    $customer = Role::create(['name' => 'customer']);

    // sales role
    $sales = Role::create(['name' => 'sales'])
        ->givePermissionTo(['index category', 'show category', 'create category', 'edit category']);

    // admin role
    $admin = Role::create(['name' => 'admin'])
        ->givePermissionTo(Permission::all());
}
```

Nu zijn de rollen customer, sales en admin aangemaakt, waarbij er permissies aan de rollen zijn gekoppeld.

De customer heeft verder nog geen permissies. We hebben op dit moment namelijk nog niks gemaakt waar hij bij mag komen. Lijkt me ook wel logisch dat een customer geen rechten krijgt in een admin van de site.

Sales mag wel een categorie aanmaken en wijzigen, maar niet verwijderen. Dit omdat een categorie verwijderen wel gevaarlijk kan zijn. De admin mag alles, dus ook een categorie verwijderen.

De rollen en permissies zijn nu klaar.

## 4.2. User Seeder

Om ervoor te zorgen dat een user een rol gaat krijgen, zullen we de *UserSeeder* aan gaan maken

```
D:\Wamp\sites\laravelv1>php artisan make:seeder UserSeeder
Seeder created successfully.
```

We geven op dit moment alleen even de 3 eigen gemaakte users een role. Ik maak expres voor elke rol in ieder geval 1 user aan, zodat je later hiermee kan testen. Ook de naamgeving van deze gebruikers zijn duidelijk.

Let even op dat we de *use App\Models\User;* bovenin gebruiken. Voor wachtwoorden hebben we ook een hash nodig, waardoor we ook *use Illuminate\Support\Facades\Hash;* nodig hebben

```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6 use App\Models\User;
7 use Illuminate\Support\Facades\Hash;
8
9 class UserSeeder extends Seeder
10 {
11     /**
12      * Run the database seeds.
13      *
14      * @param \Database\Factories\UserFactory $factory
15      */
16     public function run(\Database\Factories\UserFactory $factory)
17     {
18         $user = $factory->newModel()
19             ->make([
20                 'name' => $this->faker->name(),
21                 'email' => $this->faker->unique()->safeEmail(),
22                 'email_verified_at' => now(),
23                 'password' => Hash::make('$2y$10$92IXUNpkjO0rQ5byMi.Ye4oKoEa3Rc9llC/.og/at2.uheWG/igi'), // password
24                 'remember_token' => Str::random( length: 10),
25             ]);
26
27         $user->save();
28     }
29 }
```

Door middel van de model User kunnen we namelijk de factory van de user gebruiken. In de factory, die we al van Laravel krijgen, staat dit:

```
public function definition()
{
    return [
        'name' => $this->faker->name(),
        'email' => $this->faker->unique()->safeEmail(),
        'email_verified_at' => now(),
        'password' => '$2y$10$92IXUNpkjO0rQ5byMi.Ye4oKoEa3Rc9llC/.og/at2.uheWG/igi', // password
        'remember_token' => Str::random( length: 10),
    ];
}
```

Wat we hier steeds doen is voor de aangemaakte user de assignRole methode gebruiken om de user een rol te geven. Op deze manier krijgen we niet random users, maar users waarbij we weten welk rol ze hebben. Het wachtwoord is ook even simpel zodat we straks ook kunnen inloggen met die users.

```
class UserSeeder extends Seeder
{
    /**
     * Run the database seeds. ...
     */
    public function run()
    {
        // create customer
        $customer = User::factory()->create([
            'name' => 'Customer',
            'email' => 'customer@tcrmbo.nl',
            'password' => Hash::make('test1234')
        ]);
        $customer->assignRole('customer');

        // create sales account
        $sales = User::factory()->create([
            'name' => 'Sales',
            'email' => 'sales@tcrmbo.nl',
            'password' => Hash::make('test1234')
        ]);
        $sales->assignRole('sales');

        // create admin
        $admin = User::factory()->create([
            'name' => 'Admin',
            'email' => 'admin@tcrmbo.nl',
            'password' => Hash::make('test1234')
        ]);
        $admin->assignRole('admin');
    }
}
```

Om ervoor te zorgen dat de RoleAndPermissionSeeder en UserSeeder uitgevoerd gaan worden, gaan we nog even de DatabaseSeeder aanpassen.

```
class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     */
    public function run()
    {
        $this->call([
            RoleAndPermissionSeeder::class,
            UserSeeder::class,
            CategorySeeder::class
        ]);
    }
}
```

Let er even op dat je de RoleAndPermissionSeeder boven de UsersSeeder zet. Dit omdat eerst de permissies en rollen bekend moeten zijn voordat je users er aan koppelt. Daarna komen de andere seeders.

Om ervoor te zorgen dat we weer een schone database krijgen doe ik eerst een migrate:fresh --seed. Het verschil tussen migrate:refresh en migrate:fresh is, dat bij een fresh gewoon alles wordt verwijderd. Bij een refresh wordt de down() methode van de migrations gebruikt. Soms, als relaties niet goed zijn gegaan waardoor je errors kan krijgen met de down methode is dit handig. Ook als je niet naar een vorige situatie hoeft, maar gewoon een schone database is het handig. We willen nu een schone database, waardoor we dit commando gaan gebruiken: `php artisan migrate:fresh --seed`

```
D:\Wamp\sites\laravelv1>php artisan migrate:fresh --seed
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (16.86ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (16.86ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (16.58ms)
Migrating: 2021_03_12_195928_create_categories_table
Migrated: 2021_03_12_195928_create_categories_table (7.51ms)
Migrating: 2021_03_23_184829_update_name_length_categories_table
Migrated: 2021_03_23_184829_update_name_length_categories_table (42.90ms)
Migrating: 2021_04_01_200405_create_permission_tables
Migrated: 2021_04_01_200405_create_permission_tables (213.75ms)
Seeding: Database\Seeders\RoleAndPermissionSeeder
Seeded: Database\Seeders\RoleAndPermissionSeeder (76.20ms)
Seeding: Database\Seeders\UserSeeder
Seeded: Database\Seeders\UserSeeder (185.47ms)
Seeding: Database\Seeders\CategorySeeder
Seeded: Database\Seeders\CategorySeeder (7.20ms)
Database seeding completed successfully.
```

In de database hebben we nu wel veel meer tabellen gekregen

Tabel	Actie	Rijen
categories	★ Verkennen Structuur Zoeken Invoegen Legen Verwijderen	3
failed_jobs	★ Verkennen Structuur Zoeken Invoegen Legen Verwijderen	0
migrations	★ Verkennen Structuur Zoeken Invoegen Legen Verwijderen	6
model_has_permissions	★ Verkennen Structuur Zoeken Invoegen Legen Verwijderen	0
model_has_roles	★ Verkennen Structuur Zoeken Invoegen Legen Verwijderen	3
password_resets	★ Verkennen Structuur Zoeken Invoegen Legen Verwijderen	0
permissions	★ Verkennen Structuur Zoeken Invoegen Legen Verwijderen	5
roles	★ Verkennen Structuur Zoeken Invoegen Legen Verwijderen	3
role_has_permissions	★ Verkennen Structuur Zoeken Invoegen Legen Verwijderen	9
users	★ Verkennen Structuur Zoeken Invoegen Legen Verwijderen	3
10 tabellen	Som	32

De RoleAndPermission maakt 5 tabellen: model\_has\_permissions, model\_has\_roles, permissions, roles, role\_has\_permissions. Als we naar de rollen kijken staan deze netjes in de tabel.

id	name	guard_name
1	customer	web
2	sales	web
3	admin	web

Ook de 3 gebruikers staan netjes in de database.

id	name	email
1	Customer	customer@tcrmbo.nl
2	Sales	sales@tcrmbo.nl
3	Admin	admin@tcrmbo.nl

De seeders zijn nu klaar, want alles staat in de database.

### 4.3. Opdracht 10: Roles & Users

Maak nu bij projects alle rolles en permissies aan. De volgende rollen moeten aangemaakt worden:

- Id = 1: Student
- Id = 2: Teacher
- Id = 3: Admin

De seeder die je hiervoor aan maakt moet heten: `RoleAndPermissionSeeder`.

Verder maak je de volgende gebruikers aan, zodat je straks met deze gebruikers kan testen:

<b>id</b>	<b>name</b>	<b>email</b>	<b>Password</b>
1	student	<a href="mailto:student@tcrmbo.nl">student@tcrmbo.nl</a>	student
2	teacher	<a href="mailto:teacher@tcrmbo.nl">teacher@tcrmbo.nl</a>	teacher
3	admin	<a href="mailto:admin@tcrmbo.nl">admin@tcrmbo.nl</a>	admin

De seeder die je hiervoor aan maakt moet heten: `UserSeeder`.

#### Opdracht controle

Als je de opdracht af hebt en gebruik hebt gemaakt van de opdrachten repository, kan je het volgende commando gebruiken om te controleren of alles goed werkt:

```
.\vendor\bin\pest --group=Opdracht10
```

## 4.4. Middleware gebruiken in Routes

De eerste controle of je ergens bij mag komen kunnen we al in de routes doen. Dit doen we dan met middleware. Nu is er heel veel in middleware wat mogelijk is, maar op dit moment gaan we alleen doen wat we nu nodig hebben.

Wat we willen is dat gebruikers die niet ingelogd zijn, of niet de juiste rol hebben, niet bij de categorie crud kunnen komen. Als eerst zullen we de middleware aan onze kernel moeten toevoegen. De kernel vinden we in `app/Http/Kernel.php`

We gaan hier de volgende onderdelen toevoegen:

```
'role' => \Spatie\Permission\Middlewares\RoleMiddleware::class,  
'permission' => \Spatie\Permission\Middlewares\PermissionMiddleware::class,  
'role_or_permission' => \Spatie\Permission\Middlewares\RoleOrPermissionMiddleware::class,
```

We voegen deze toe aan protected \$routeMiddleware

```
/** The application's route middleware. ... */  
  
protected $routeMiddleware = [  
  
    'auth' => \App\Http\Middleware\Authenticate::class,  
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,  
    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,  
    'can' => \Illuminate\Auth\Middleware\Authorize::class,  
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,  
    'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,  
    'signed' => \Illuminate\Routing\Middleware\ValidateSignature::class,  
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,  
    'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,  
    'role' => \Spatie\Permission\Middlewares\RoleMiddleware::class,  
    'permission' => \Spatie\Permission\Middlewares\PermissionMiddleware::class,  
    'role_or_permission' => \Spatie\Permission\Middlewares\RoleOrPermissionMiddleware::class,  
];
```

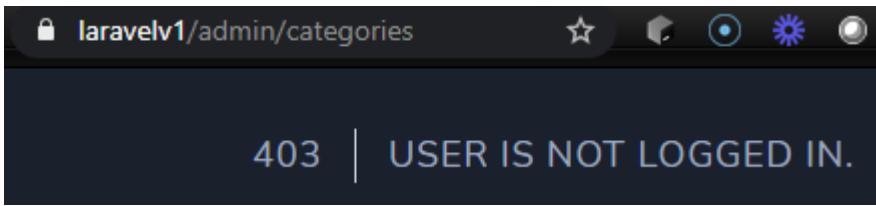
In de routes/web.php gaan we voor de categories routes een middleware gebruiken. Dit doen we zo:

```
21     Route::group(['middleware' => ['role:customer|sales|admin']], function () {  
22         Route::get('admin/categories/{category}/delete', [CategoryController::class, 'delete'])  
23             ->name('categories.delete');  
24         Route::resource('admin/categories', CategoryController::class);  
25     });
```

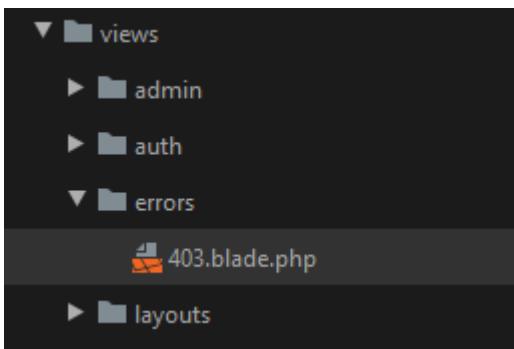
Dit zorgt ervoor dat als we in de browser naar categories gaan, eerst gekeken wordt of je rechten hebt.

Als je middleware gebruikt geef je aan dat je met authenticatie bezig bent. Ik geef nu aan dat je de rol customer, sales of admin moet hebben om deze routes te zien.

Als we dit uitproberen zullen we als we ingelogd zijn als owner gewoon de pagina zien. We zien een 403 pagina zien als we niet zijn ingelogd.



Nu wil ik niet dat de error pagina er zo uitziet. De error pagina wil ik binnen mijn layout hebben. Dit kan je voor elkaar krijgen als je deze zelf maakt. Hiervoor gaan we in onze views map een map voor errors maken met een `403.blade.php` erin.



In de `403.blade.php` zorgen we ervoor dat we een foutmelding zien.

```
@extends('layouts.layout')

@section('topmenu_items')
    
@endsection

@section('content')
    <div class="container mx-1">
        <div class="ml-2 flex flex-col">
            <h2 class="my-4 text-4xl font-semibold text-gray-600 dark:text-gray-400">
                Error
            </h2>
            <div class="bg-red-200 text-red-900 rounded-lg shadow-md p-6 pr-10 mb-8"
                 style="...>
                {{ $exception->getMessage() }}
            </div>
        </div>
    </div>
@endsection
```

Als we nu opnieuw gaan kijken naar de categories, zien we de foutmelding binnen de eigen layout.

The screenshot shows a web browser window with the URL 'laravelv1/admin/categories'. On the left, there's a sidebar with the title 'My Admin' and a user profile picture. Below it, the text 'Hello Koningstein' is displayed. The main content area has a large 'Error' heading and a pink message box containing the text 'User is not logged in.'

Voorlopig houden we even deze standaard foutmeldingen. Later kunnen we nog kijken dat je eigen foutmeldingen kan schrijven.

#### 4.5. Authenticatie

Nu we niet meer bij onze categories kunnen, zullen we aan de slag moeten met de login. Alles van de login is eigenlijk al in het project. We kunnen bijvoorbeeld als we naar localhost/login gaan, het formulier zien om in te loggen. (en ja, deze werkt al!)

The screenshot shows a web browser window with the URL 'laravelv1/login'. The page features a large, stylized letter 'U' logo. Below it is a login form with fields for 'Email' and 'Password', each accompanied by a file upload icon. There is also a 'Remember me' checkbox and a 'LOG IN' button. At the bottom of the form, there is a link for 'Forgot your password?'

Op dit moment heeft alles van auth zijn eigen layout. Nu zou het mooi zijn om dit in onze eigen layout te zetten, alleen wordt er met een techniek gewerkt die even wat lastiger is. Dit zal later in deze lessenserie wel behandeld worden. Voor nu houden we heel even de standaard layout van het login gedeelte. Wel is het handig om in onze eigen layout te weten of je bent ingelogd, en de linkjes te hebben naar login / logout.

Als we naar de index van categorie gaan krijgen we een foutmelding. Dit weten we en is nu ook even goed

The screenshot shows a browser window with the URL 'laravelv1/admin/categories'. The page has a dark header with navigation icons. On the left, there's a sidebar with a user profile picture, the name 'Hello Koningstein' in bold, and the role 'Admin'. Below that is a 'Category Admin' section. At the bottom of the sidebar is a red 'Log Out' button. The main content area has a large 'Error' heading and a pink box containing the text 'User is not logged in.'

Wat wel mooi is, is dat links 'Hello Koningstein' staat. In de layout die ik heb gebruikt stond namelijk een plek voor de gebruikersnaam. We gaan zorgen dat daar de naam komt als je bent ingelogd. Verder zie je ook onderaan links een logout knop staan. Deze gaat nog nergens naar toe, maar kunnen we wel mooi gebruiken.

Als eerst de naam. Als je niet bent ingelogd, ben je een guest. Dan kan je @guest gebruiken binnen blade.

```
<h2
    class="mt-4 text-xl dark:text-gray-300 font-extrabold capitalize">
    Hello
    @guest
        Guest
    @else
        {{ Auth::user()->name }}
    @endguest
</h2>
```

Als je geen guest bent, zal je ingelogd zijn. Dan kan je de ingebouwde Auth van Laravel gebruiken om de naam van de user op te halen. Het ziet er dan al zo uit.

The screenshot shows a user profile with a picture, the name 'Hello Guest', and the role 'Role....' below it.

In het groen stond eerst admin. Heb al even voorbereid dat we ook de rol zouden kunnen ophalen straks.

Voor de linkjes naar login en register kunnen we dit natuurlijk op dezelfde manier doen.

```
<ul class="mt-2 text-gray-600">
    <!-- Links -->
    @guest
        <li class="mt-8">
            <a href="{{ route('login') }}>
                <span class="ml-2 capitalize font-medium text-black dark:text-gray-300">{{ __('Login') }}</span>
            </a>
        </li>
        @if (Route::has('register'))
            <li class="mt-8">
                <a href="{{ route('register') }}>
                    <span class="ml-2 capitalize font-medium text-black dark:text-gray-300">{{ __('Register') }}</span>
                </a>
            </li>
        @endif
    @else
        <li class="mt-8">
            <a href="{{ route('categories.index') }}" class="flex">
                <span class="ml-2 capitalize font-medium text-black dark:text-gray-300">Category Admin</span>
            </a>
        </li>
    @endguest
</ul>
```

Je kan het registreren ook uitzetten als optie, hierdoor nog even de extra check of je mag registreren voordat de link getoond wordt. Daarnaast wordt nu de Categorie Admin link alleen getoond als je bent ingelogd. Natuurlijk moet je dan straks ook nog permissie hebben door middel van je role, maar dat is nog niet geregeld waardoor we het nu nog zo doen.

Nu denk je, de logout kunnen we hetzelfde doen. Helaas niet helemaal het geval. Als we een link maken met de route logout krijgen we dit.

```
<a href="{{ route('logout') }}>
```

D:\Wamp\sites\laravelv1\

Symfony\Component\HttpKernel\Exception\MethodNotAllowedHttpException  
The GET method is not supported for this route. Supported methods: POST.

<https://laravelv1/logout>

Je mag dus niet door middel van een get naar de logout, maar alleen met een POST methode. Als we perse een link willen kunnen we dit wel faken door middel van javascript.

```

<div class="mt-auto flex items-center text-red-700 dark:text-red-400">
    <!-- important action -->
    <a href="{{ route('logout') }}" class="flex items-center" onclick="event.preventDefault();
        document.getElementById('logout-form').submit();">
        <svg class="fill-current h-5 w-5" viewBox="0 0 24 24">
            <path
                d="M16 17v-3H9v-4h7V7L5 5-5 5M14 2a2 2 0 0 1 0 12
                2v2h-2V4H5v16h9v-2h2v2a2 2 0 0 1-2 2H5a2 2 0 0 1-2-2V4a2 2
                0 0 1-2-2h9z"></path>
        </svg>
        <span class="ml-2 capitalize font-medium"> {{ __('Logout') }} </span>
    </a>
    <form id="logout-form" action="{{ route('logout') }}" method="POST" style="display: none;">
        @csrf
    </form>
</div>

```

Aan de link hebben we een onclick toegevoegd die het formulier eronder gebruikt om de actie uit te voeren. Het formulier zelf heeft style display none zodat het niet zichtbaar is. Als je nu op de logout klikt kom je netjes bij het login formulier terecht. Nu kunnen we meteen testen of de inlog werkt met bijv het admin account wat we in de userseeder hebben gemaakt.

Als je op login klikt zodra je de juiste gegevens hebt ingevoerd, zien we netjes de categorie admin.

The screenshot shows a web browser window with the URL 'laravelv1/admin/categories'. On the left, there's a sidebar with a user profile picture, a 'Hello Admin' greeting, and a 'Role....' link. Below that is a 'Category Admin' section. At the bottom of the sidebar is a 'Logout' button. The main content area has a title 'Category Admin' and a table with four columns: NAME, DETAILS, EDIT, and DELETE. The table contains three rows of data:

NAME	DETAILS	EDIT	DELETE
Prof. Isaac Smith	Details	Edit	Delete
Samara Rau IV	Details	Edit	Delete
Julian Dibbert	Details	Edit	Delete

Verder zie je ook netjes de 'Hello Admin' aan de linkerkant. De login / register links zie je niet meer, maar er is wel weer een link naar de category admin. Er is nog een ding wat minder prettig is. Als we naar de site gaan zien we nog steeds het standaard beginscherm.

The screenshot shows the standard Laravel welcome page. At the top, it says 'laravelv1/' and has a 'Log in' link. The main content features the Laravel logo and the word 'Laravel' in red. Below that are four sections: 'Documentation', 'Laracasts', 'Laravel News', and 'Vibrant Ecosystem'. At the bottom, there are links for 'Shop' and 'Sponsor', and the text 'Laravel v8.34.0 (PHP v8.0.3)'.

We gaan dit veranderen naar de categorie index, zodat we onze eigen lay-out zien.

Dit is te doen doen de route van '/' te wijzigen. Daarin staat nu dat de CategoryController ingeladen gaat worden.

```
17     Route::get('/', [CategoryController::class, 'index']);
18
19     Route::group(['middleware' => ['role:customer|sales|admin']], function () {
20         Route::get('admin/categories/{category}/delete', [CategoryController::class, 'delete'])
21             ->name('categories.delete');
22         Route::resource('admin/categories', CategoryController::class);
23     });

```

Als je dacht dat de admin al geheel beveiligd was, helaas. We kunnen het nu nog steeds zien.

Doordat we dit nu gedaan hebben weten we ook dat de permissies gewoon nog niet goed zijn. Dit klopt ook wel, want we hebben niet aangegeven in de controller waar je wel of niet bij mag komen, we hebben alleen de url afgeschermd van /admin/categories

## 4.6. Permissies in de Controller

In de seed (RoleAndPermissionSeeder) hebben we een aantal permissies aangemaakt voor bepaalde methodes.

```
public function run()
{
    // reset cached roles and permissions
    app()[\Spatie\Permission\PermissionRegistrar::class]->forgetCachedPermissions();

    // permissions for category CRUD
    Permission::create(['name' => 'index category']);
    Permission::create(['name' => 'show category']);
    Permission::create(['name' => 'create category']);
    Permission::create(['name' => 'edit category']);
    Permission::create(['name' => 'delete category']);
}
```

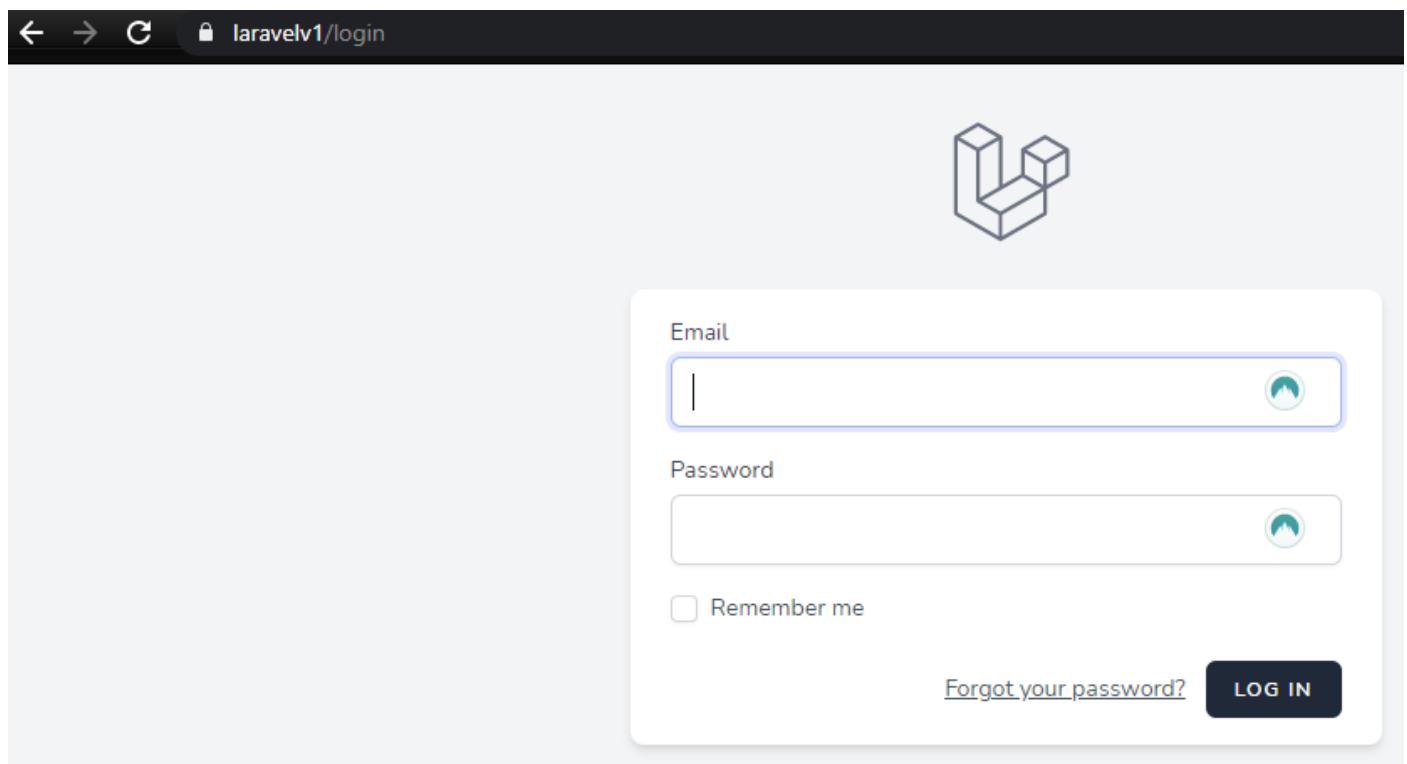
Deze permissions gaan we in de controller koppelen aan methodes. Vanuit OOP ken je de `__construct` methode, die wordt uitgevoerd als er een instantie van de class wordt aangemaakt. De construct gaan we dan voor de koppeling gebruiken.

```
11  class CategoryController extends Controller
12  {
13      /** Set permissions on methods ...*/
14
15      public function __construct()
16      {
17          $this->middleware( middleware: 'auth');
18
19          $this->middleware( middleware: 'permission:index category', ['only' => ['index']]);
20
21          $this->middleware( middleware: 'permission:show category', ['only' => ['show']]);
22
23          $this->middleware( middleware: 'permission:create category', ['only' => ['create', 'store']]);
24
25          $this->middleware( middleware: 'permission:edit category', ['only' => ['edit', 'update']]);
26
27          $this->middleware( middleware: 'permission:delete category', ['only' => ['delete', 'destroy']]);
28      }
29  }
```

Je ziet als eerst dat de construct de bovenste methode is in de controller. Zoals je weet is dit een standaard afspraak. In de methode zeggen we eerst dat we authenticatie gebruiken binnen de controller. Daarna geven we aan de 'permission:index category' de koppeling met de index methode binnen de controller.

Omdat de permissie al ergens anders met een Role is gekoppeld, is het koppelen van een permissie handiger om te doen. Mocht een Role toch niet erbij mogen haal je de permissie van de Role af. Je kan ook een rol aan een methode koppelen, maar hoe we het nu hebben ingericht is dit veel handiger.

Als we naar de category admin gaan, zonder ingelogd te zijn, zien we meteen het login scherm.



Je kan dus nu niet meer bij de admin zonder de juiste permissies.

We loggen in met admin@tcrmbo.nl en test1234 en het resultaat is, is dat we nu wel weer de admin kunnen zien.

NAME	DETAILS	EDIT	DELETE
Prof. Isaac Smith	Details	Edit	Delete
Samara Rau IV	Details	Edit	Delete
Julian Dibbert	Details	Edit	Delete

Als we uitloggen en inloggen met [sales@tcrmbo.nl](mailto:sales@tcrmbo.nl) en test1234 zien we dit:

NAME	DETAILS	EDIT	DELETE
Prof. Isaac Smith	Details	Edit	Delete
Samara Rau IV	Details	Edit	Delete
Julian Dibbert	Details	Edit	Delete

Een Sales mag wel bij de index, create en edit, maar niet bij de delete. Als je op delete klikt bij een categorie zie je ook een error scherm dat je niet de juiste permissies hebt.

User does not have the right permissions.

Eigenlijk zou je niet een delete mogelijkheid moeten hebben als je als sales bent ingelogd. Binnen Blade zit de mogelijkheid om dit af te schermen. Met @can kan je aangeven welke permissie je moet hebben om het te kunnen zien. (weer goed om dit niet op basis van een rol te doen maar met permissie)

```
<tbody class="bg-white divide-y divide-gray-200">
@foreach($categories as $category)
<tr>
    <td class="px-6 py-4 whitespace nowrap text-sm font-medium text-gray-900">
        {{ $category->name }}
    </td>
    <td class="px-6 py-4 whitespace nowrap text-sm text-gray-500">
        <a href="{{ route('categories.show', ['category' => $category->id]) }}">Details</a>
    </td>
    <td class="px-6 py-4 whitespace nowrap text-sm text-gray-500">
        <a href="{{ route('categories.edit', ['category' => $category->id]) }}">Edit</a>
    </td>
    @can('delete category')
    <td class="px-6 py-4 whitespace nowrap text-sm text-gray-500">
        <a href="{{ route('categories.delete', ['category' => $category->id]) }}">Delete</a>
    </td>
    @endcan
    </tr>
@endforeach
</tbody>
```

Als we dan naar de index gaan van de categorie admin met als sales ingelogd, zien we de delete mogelijkheid gewoon niet meer. Voorkomen van het zien van een error scherm is altijd beter dan er een krijgen!

NAME	DETAILS	EDIT
Prof. Isaac Smith	Details	Edit
Samara Rau IV	Details	Edit
Julian Dibbert	Details	Edit

## 4.7. Opdracht 11: Permissions

Maak de volgende permissies aan.

Permissie naam	Methodes	Rollen met rechten
Project index	Index	Student, teacher, admin
Project create	Create + store	Student, teacher, admin
Project show	Show	Student, teacher, admin
Project edit	Edit + update	Student, teacher, admin
Project delete	Delete + destroy	Teacher, admin

Je zorgt dat in routes en in de controller de beveiliging is gemaakt voor deze permissies.

### Opdracht controle

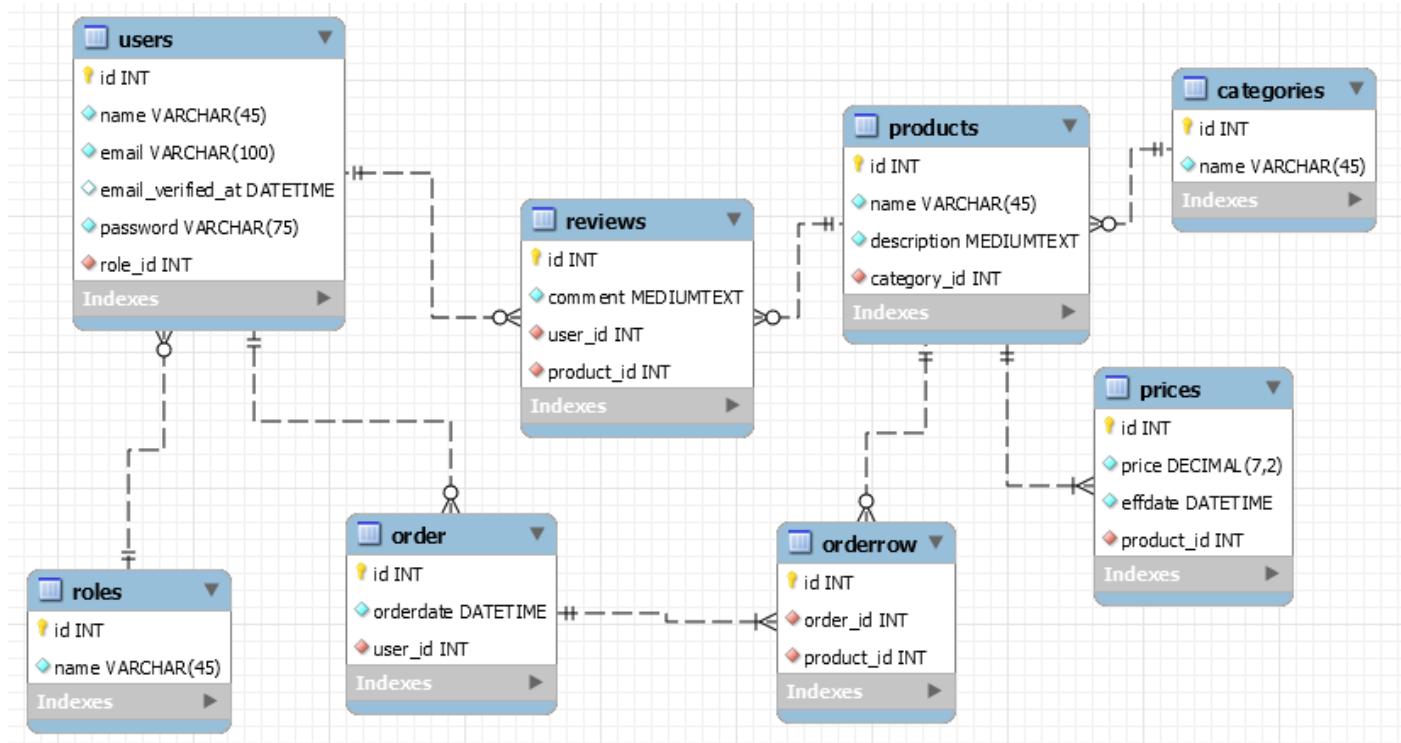
Als je de opdracht af hebt en gebruik hebt gemaakt van de opdrachten repository, kan je het volgende commando gebruiken om te controleren of alles goed werkt:

```
.\vendor\bin\pest --group=Opdracht11
```

Let op, dat bij deze controle alle testen worden gedaan of de gehele crud van project correct is, met de permissies erbij. Omdat behoorlijk wat testen worden gedaan kan het wat langer duren.

## 5. Product & Price

Het wordt nu tijd om aan de slag te gaan met Product en Price. We doen ze meteen beide, zodat we allerlei onderdelen tegen gaan komen met relaties. Als we nog even kijken naar de database opzet, zie je een category\_id in de tabel products staan. Verder staat er een product\_id in prices. Ook is het goed om de huidige prijs van een product in het overzicht te hebben van producten.



Nu je de basis al wel onder de knie hebt van Laravel, gaan we kijken naar hoe we het ons zelf wat makkelijk kunnen maken. Als eerst gaan we door middel van het aanmaken van de model alle andere onderdelen aanmaken.

We gaan natuurlijk dezelfde volgorde gebruiken als bij categorie. De volgorde wordt dus:

- Migrations
- Factories
- Seeds
- Resource controller Product
- Resource controller Price

Zoals je ziet doen we de database onderdelen wel meteen voor beide tabellen, maar de controller kan na elkaar. Dit omdat voor de database je meteen goed de koppeling tussen de tabellen kan regelen.

## 5.1. Models

Nu als eerst de models. Voor Product doen we dit met: `php artisan make:model --all Product`

```
D:\Wamp\sites\laravelv1>php artisan make:model --all Product
Model created successfully.
Factory created successfully.
Created Migration: 2021_04_05_122631_create_products_table
Seeder created successfully.
Controller created successfully.
```

Zoals je ziet zijn er allerlei bestanden aangemaakt:

- Model: Product
- Factory: ProductFactory
- Migration: create\_products\_table
- Seeder: ProductSeeder
- Controller: ProductController

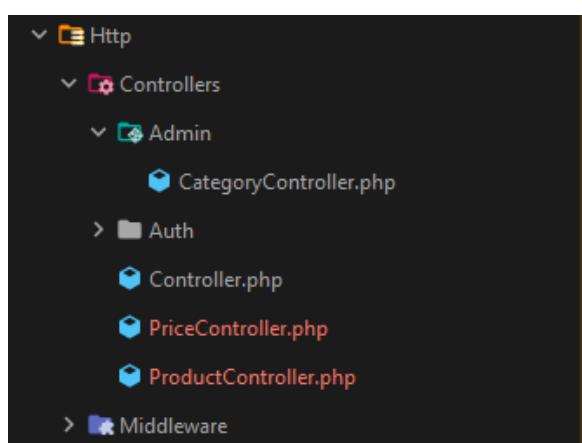
Dit gaan we ook meteen voor Price doen met: `php artisan make:model --all Price`

```
D:\Wamp\sites\laravelv1>php artisan make:model --all Price
Model created successfully.
Factory created successfully.
Created Migration: 2021_04_05_123102_create_prices_table
Seeder created successfully.
Controller created successfully.
```

Ook hiervoor zijn nu de bestanden aangemaakt:

- Model: Price
- Factory: PriceFactory
- Migration: create\_prices\_table
- Seeder: PriceSeeder
- Controller: PriceController

Zoals je ziet is deze optie een stuk makkelijker dan voor elk bestand een php artisan commando te moeten typen. Er zit wel 1 nadeel aan deze manier. We krijgen onze controller nu niet in een admin map.



Om even goed naar de verschillen te kijken van CategoryController en ProductController, zetten we deze naast elkaar:

The screenshot shows two code editors side-by-side. The left editor contains the code for CategoryController.php, which has a namespace of App\Http\Controllers\Admin; and uses Controller, CategoryStoreRequest, CategoryUpdateRequest, Category, and Illuminate\Http\Request. The right editor contains the code for ProductController.php, which has a namespace of App\Http\Controllers; and uses Model, Request, and extends Controller. Both files have a public index() function.

```
<?php  
namespace App\Http\Controllers\Admin;  
use App\Http\Controllers\Controller;  
use App\Http\Requests\CategoryStoreRequest;  
use App\Http\Requests\CategoryUpdateRequest;  
use App\Models\Category;  
use Illuminate\Http\Request;  
class CategoryController extends Controller  
{  
    /** Display a listing of the resource. ... */  
    public function index()  
}
```

```
<?php  
namespace App\Http\Controllers;  
use App\Models\Product;  
use Illuminate\Http\Request;  
class ProductController extends Controller  
{  
    /** Display a listing of the resource. ... */  
    public function index()  
}
```

Hier zien we dat de namespace anders is. Bij de CategoryController staan er nog \Admin achter. Tevens omdat de namespace dan anders is, moet ook nog use App\Http\Controllers\Controller; gebruikt worden, zodat de basis controller gebruikt kan worden.

We gaan dit ook doorvoeren voor Product en Price. We verplaatsen de bestanden naar de admin map waar de CategoryController ook staat. De namespace veranderen we naar de Admin map en de basis controller wordt ingeladen.

The screenshot shows a code editor with two files: ProductController.php and PriceController.php. They are located in the laravelv1\app\Http\Controllers\Admin folder. Both files have namespaces of App\Http\Controllers\Admin; and extend the Controller class. They also use Model and Request, and have a public index() function. The project structure on the left shows the files under laravelv1\app\Http\Controllers\Admin.

```
<?php  
namespace App\Http\Controllers\Admin;  
use App\Http\Controllers\Controller;  
use App\Models\Product;  
use Illuminate\Http\Request;  
class ProductController extends Controller  
{  
    /** Display a listing of the resource. ... */  
    public function index()  
}
```

```
<?php  
namespace App\Http\Controllers\Admin;  
use App\Http\Controllers\Controller;  
use App\Models\Price;  
use Illuminate\Http\Request;  
class PriceController extends Controller  
{  
    /** Display a listing of the resource. ... */  
    public function index()  
}
```

Als je bestanden verplaatst is het slim om altijd even een composer dump-autoload te doen.

```
D:\Wamp\sites\laravelv1>composer dump-autoload  
Generating optimized autoload files  
composer/package-versions-deprecated: Generating version class...  
composer/package-versions-deprecated: ...done generating version class
```

Nu de models zijn aangemaakt, zullen we ook de relaties in de models moeten zetten. Dit zodat we deze straks kunnen gebruiken als we bijvoorbeeld een combinatie nodig hebben van categorie + product, of product + prijs. De model regelt alles met de data. Je zal merken dat we door middel van de model straks makkelijk de gekoppelde data kunnen krijgen zonder een moeilijke join te schrijven. We beginnen met de model Category, omdat we hier maar 1 relatie hebben, namelijk met Product.

In Category geven we aan dat deze een relatie heeft met Product door middel van een methode.

```
class Category extends Model
{
    use HasFactory;

    protected $fillable = ['name'];

    public function product()
    {
        return $this->hasMany(related: Product::class);
    }
}
```

Het is van belang om de methode zo duidelijk mogelijk te maken. Met bijv. product() is duidelijk dat dit met producten te maken heeft. Als je even terug denkt aan de database lessen met ERDish:

*Each entity1 (may/must) relationname (1 and only 1/1 or more) entity2.*

Dit moet je in beide richtingen uitschrijven om de relatie te kunnen tekenen. Vanaf de product zou het zijn:

- *Each category may have 1 or more products*

De andere richting:

- *Each product must have 1 and only 1 category*

Je ziet dus dat in de model we hetzelfde doen. De naam van de methode is naar welk model we toe verwijzen. Bij hasMany geven we dan ook de model aan. In de model Product gaan we ook de relaties maken. Deze heeft een relatie met Category, maar ook met Price.

```
class Product extends Model
{
    use HasFactory;

    // relation to Category
    public function category()
    {
        return $this->belongsTo(related: Category::class);
    }

    // relation to Price
    public function price()
    {
        return $this->hasMany(related: Price::class);
    }
}
```

Bij category gebruiken we een belongsTo. Dit is altijd bij een 1 op veel relatie zo, aan de ene kant een hasMany en aan de andere kant een belongsTo. Dit klopt ook, want een product heeft maar 1 category.

Bij de methode price() geven we dan de relatie van Product met Price. Een Product kan meerdere prijzen hebben, waardoor we de hasMany gebruiken.

En dan de Price model.

```
class Price extends Model
{
    use HasFactory;

    // relation to Product
    public function product()
    {
        return $this->belongsTo( related: Product::class);
    }
}
```

Een Price behoort bij een product. Hierdoor ook in Price een methode product(). Nu hebben we binnen Laravel aangegeven hoe de relaties zijn. Bij de migrations kunnen we zo aangeven hoe dit moet in de database.

## 5.2. Migrations

Bij de migrations gaan we nu te maken krijgen met relaties. Deze relaties willen we ook netjes in de database vastleggen, zodat hierin een extra controle wordt gedaan of de data die we gaan gebruiken wel klopt.

Als eerst de products migration.

```
class CreateProductsTable extends Migration
{
    /** Run the migrations. ...*/
    public function up()
    {
        Schema::create( table: 'products', function (Blueprint $table) {
            $table->id();
            $table->string( column: 'name', length: 45);
            $table->text( column: 'description');
            $table->foreignId( column: 'category_id')->constrained()
                ->onUpdate( action: 'restrict')->onDelete( action: 'restrict');
            $table->timestamps();
        });
    }

    /** Reverse the migrations. ...*/
    public function down()
    {
        Schema::dropIfExists( table: 'products');
    }
}
```

Als eerste dus up methode. De string en tekst zijn denk ik wel al duidelijk, maar het lastige stuk is de foreignId. (<https://laravel.com/docs/8.x/migrations#foreign-key-constraints>)

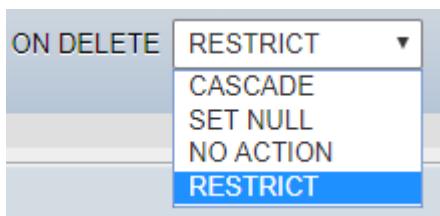
Een foreign key is in Laravel een unsignedBigInteger als datatype. Dit komt omdat een foreign key dezelfde grootte moet hebben als een primary key. Bij een primary key gebruiken we id(), wat dus een bigIncrements is. Maar om het makkelijker te maken, hebben ze de foreignId gemaakt:

The `foreignId` method is an alias for `unsignedBigInteger` while the `constrained` method will use conventions to determine the table and column name being referenced. If your table name does not match Laravel's conventions, you may specify the table name by passing it as an argument to the `constrained` method:

Je geeft dus aan met reference naar welke tabel het moet. Alleen zie je bij de code verder niks staan. Dit komt omdat er conventies (afspraken) zijn binnen Laravel. Omdat de foreign key category\_id is, weet Laravel dat deze refereert naar de tabel categories. Die tabel heeft als primary key dan id.

Een onDelete en onUpdate komt vanuit een database, maar toch krijg ik vaak vragen hierover als we met migrations aan de slag gaan. Bij een onDelete wordt er gekeken wat er gaat gebeuren als we wat gaan verwijderen. Stel je voor, we verwijderen een categorie waar producten aan gekoppeld staan. Als we dit zouden toelaten betekent dit dat we in products een verwijzing hebben naar een categorie, maar deze niet meer bestaat. Dit kan natuurlijk niet.

Hiervoor zijn binnen een database een aantal mogelijkheden. Als we binnen MySql gaan kijken zien we deze opties:



No Action:

Bij het verwijderen van een categorie laat je de foreign key bij product staan en krijgt dus een verwijzing naar een primary key die er niet meer is.

Set NULL:

Bij het verwijderen van een categorie worden alle verwijzingen naar die id in de product tabel op NULL gezet. Let op dat NULL een onbekende waarde is

Cascade:

Bij het verwijderen van een categorie met een bepaald id, worden alle producten die hieraan gekoppeld staan ook verwijderd.

Restrict:

Bij het verwijderen van een categorie en er staan nog producten gekoppeld aan die categorie, krijg je een foutmelding. Dit omdat de integriteit van de database dan niet meer goed is.

Wat we nu dus doen is ervoor zorgen dat een categorie niet zomaar verwijderd mag worden als er producten in staan. De migration voor product is nu wel klaar.

## Dan de Price migration

```
7  class CreatePricesTable extends Migration
8  {
9      /** Run the migrations. ...*/
10     public function up()
11     {
12         Schema::create('prices', function (Blueprint $table) {
13             $table->id();
14             $table->decimal('price', 8, 2);
15             $table->dateTime('effdate');
16             $table->foreignId('product_id')->constrained()
17                 ->onUpdate('cascade')->onDelete('cascade');
18             $table->timestamps();
19         });
20     }
21 }
22
23 }
```

Je ziet hier een effdate. Dit is een effective date, wat veel in databases gebruikt wordt. Het is de ingangsdatum wanneer die prijs in gaat.

Als voorbeeld:

van 1 mei t/m 8 mei is een brood 2,00 euro

van 9 mei t/m 20 mei is een brood 2,10 euro

de rest van de maand is een brood 2,20.

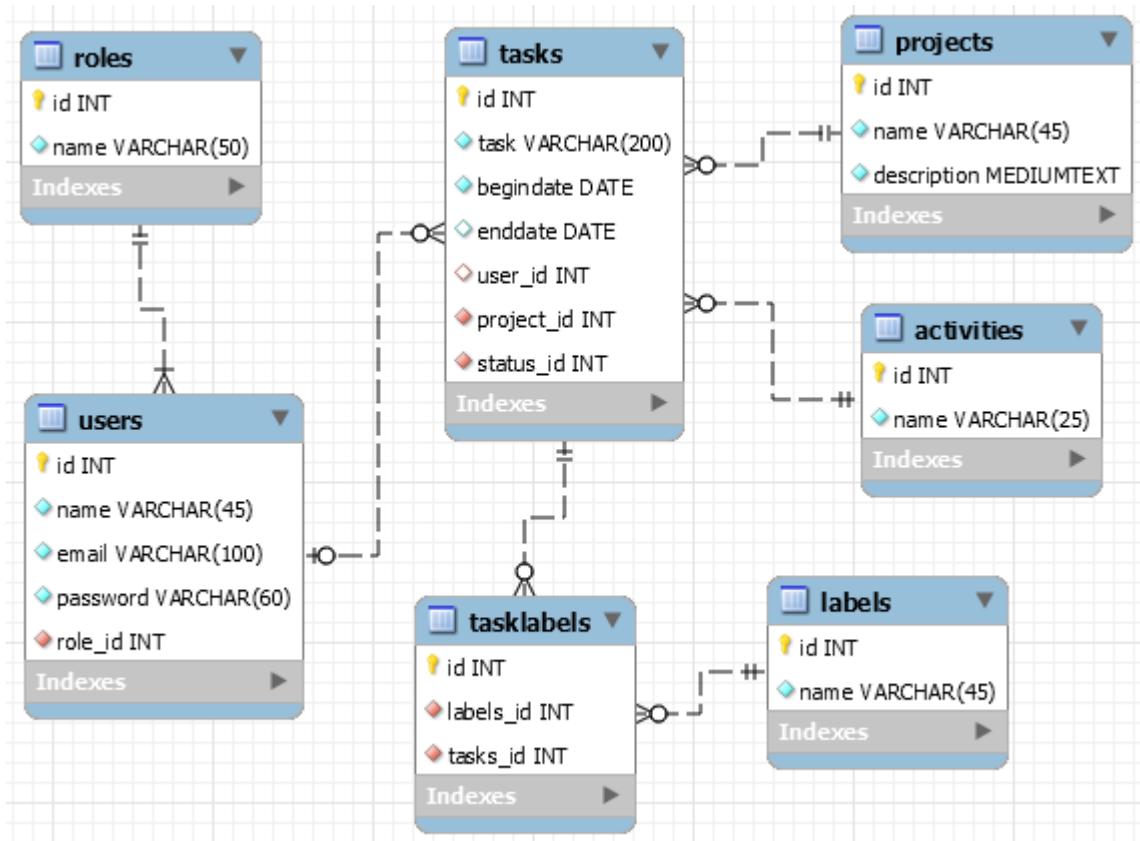
Als we per dag nu 100 broden verkopen, want hebben we dan aan inkomsten ?

Door middel van de effective date kan je dus zeggen wanneer een prijs ingaat. Je overschrijft een prijs dus niet, waardoor de geschiedenis hoeveel een product heeft gekost wordt bewaard. Ook een voordeel is, dat je die datums zelfs in de toekomst kan maken, al gaan we dit niet voor ons huidige project gebruiken. Voor ons project zal gelden, zodra we een nieuwe prijs hebben geldt deze meteen.

Bij de foreign key zie je tevens dat ik nu niet restrict gebruik, maar cascade. Als je een product verwijderd mogen ook de prijzen van dat product weg. De migrations zijn nu zover klaar voor product en price.

### 5.3. Opdracht 12: Models en Migrations

Bij deze opdracht gaan we aan de slag met de tasks. Hiervoor zullen we naast de tasks ook het database gedeelte voor activities moeten gaan regelen, want dat is een verplichte foreign key in de tabel tasks.



Maak dus de models aan voor activities en tasks.

De volgende relaties beschrijf je met een methode in de model Task: user, project, activity

Beschrijf ook de relatie met een methode in de model User: tasks

Beschrijf ook de relatie met een methode in de model Project: tasks

Beschrijf ook de relatie met een methode in de model Activity: tasks

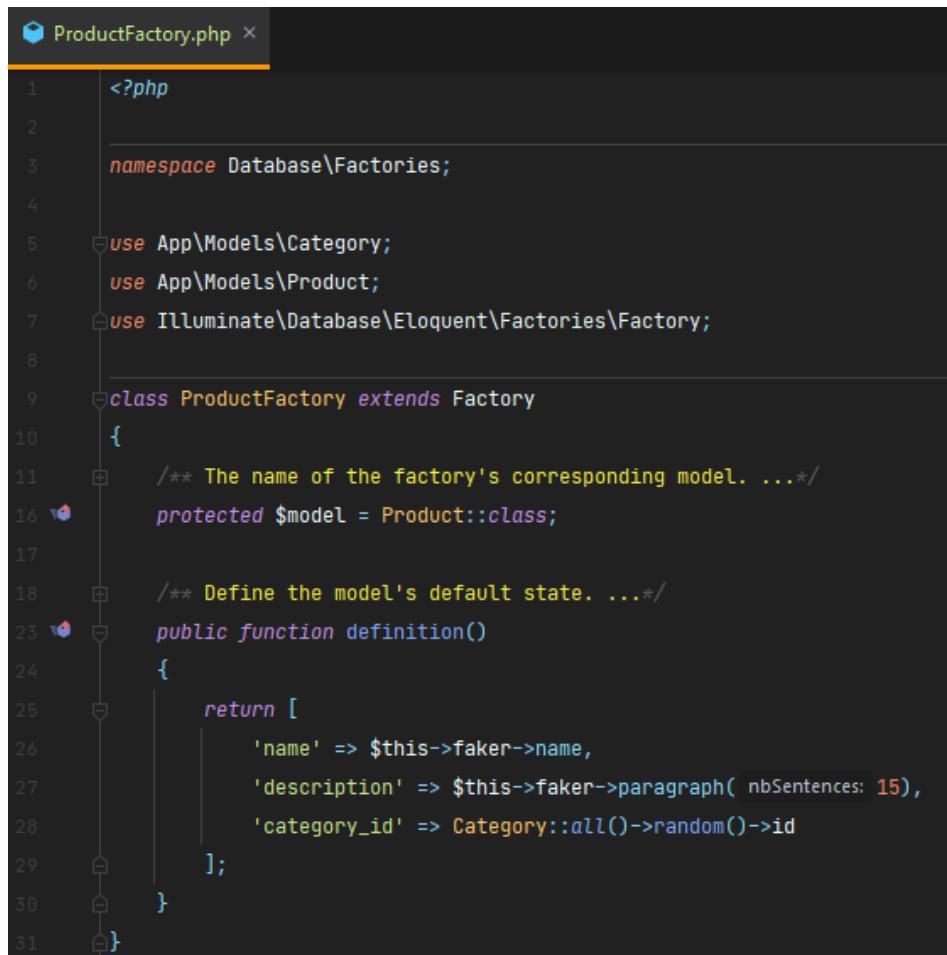
Maak ook de migrations aan voor deze 2 tabellen. Zorg dat de relaties hier goed staan voor projects, activities en users. Neem het volgende mee voor de relaties:

- Bij users update & delete: no action (een tasks hoeft niet perse een user te hebben)
  - Dit betekent ook dat dit veld nullable moet zijn.
- Bij projects update & delete: cascade (als we een project verwijderen moeten alle tasks ook weg)
- Bij activities update & delete: restrict (een taak moet een activity hebben, dus we mogen niet een activity verwijderen als die nog in gebruik is)

Zorg dat bij de migration de standaard id en timestamps gebruikt worden. Je ziet in de ERD de lengtes van de attributen die aangehouden moeten worden.

## 5.4. Factory

Als eerst gaan we aan de slag met de ProductFactory. Een Product heeft een name, description en behoort bij een category. De Factory ziet er dan zo uit.



```
<?php

namespace Database\Factories;

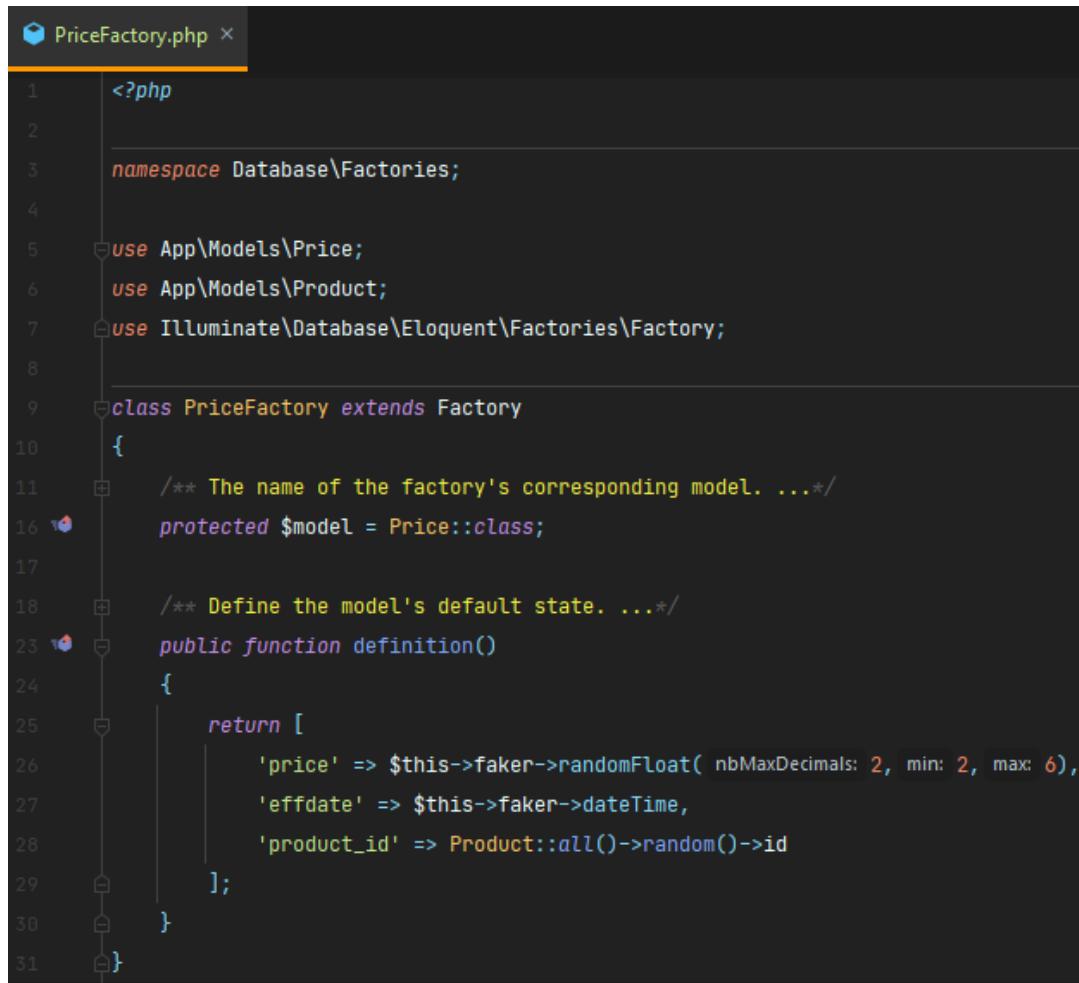
use App\Models\Category;
use App\Models\Product;
use Illuminate\Database\Eloquent\Factories\Factory;

class ProductFactory extends Factory
{
    /**
     * The name of the factory's corresponding model. ...
     */
    protected $model = Product::class;

    /**
     * Define the model's default state. ...
     */
    public function definition()
    {
        return [
            'name' => $this->faker->name,
            'description' => $this->faker->paragraph( nbSentences: 15 ),
            'category_id' => Category::all()->random()->id
        ];
    }
}
```

Omdat we op regel 28 de model van Category gebruiken om een random id te krijgen, zullen we op regel 5 de class wel moeten toevoegen. En ja, op regel 28 gaan we dus alle categorien ophalen, daar een random uit kiezen en hiervan de id gebruiken om in category\_id te zetten.

Bij Price doen we eigenlijk hetzelfde. Let er wel even op dat ik bij de migration totaal 8 cijfers heb, waarvan 2 achter de komma. Nu zeg ik hier bij de factory, 2 cijfers achter de komma, en tussen 2 en 6 cijfers.



```
PriceFactory.php
1 <?php
2
3 namespace Database\Factories;
4
5 use App\Models\Price;
6 use App\Models\Product;
7 use Illuminate\Database\Eloquent\Factories\Factory;
8
9 class PriceFactory extends Factory
10 {
11     /**
12      * The name of the factory's corresponding model. ...
13      */
14     protected $model = Price::class;
15
16     /**
17      * Define the model's default state. ...
18      */
19     public function definition()
20     {
21         return [
22             'price' => $this->faker->randomFloat( nbMaxDecimals: 2, min: 2, max: 6),
23             'effdate' => $this->faker->dateTime,
24             'product_id' => Product::all()->random()->id
25         ];
26     }
27 }
28 }
```

Door middel van de factory van Product en Price hebben we aangegeven hoe 1 instantie ervan eruit moet zien.

## 5.5. Seed

Bij de seed doen we eerst even Price. Dit lijkt gewoon net zoals de seed van de Category. We gebruiken de Price model, dus regel 5 komt erbij. We geven aan de factory te gebruiken en dat we 10 prijzen willen.

```
3  namespace Database\Seeders;
4
5  use App\Models\Price;
6  use Illuminate\Database\Seeder;
7
8  class PriceSeeder extends Seeder
9  {
10     /** Run the database seeds. ...*/
11     public function run()
12     {
13         Price::factory()
14             ->times( count: 10)
15             ->create();
16     }
17 }
18
19
20
21 }
```

De product seed zal er natuurlijk heel erg op lijken.

```
3  namespace Database\Seeders;
4
5  use App\Models\Product;
6  use Illuminate\Database\Seeder;
7
8  class ProductSeeder extends Seeder
9  {
10     /** Run the database seeds. ...*/
11     public function run()
12     {
13         Product::factory()
14             ->times( count: 10)
15             ->create();
16     }
17 }
18
19
20
21 }
```

Ook hier de model gebruikt, dus op regel 5 toegevoegd. Aangegeven de factory te willen gebruiken en ook hiervan willen we er 10 hebben.

We moeten nog even de PriceSeeder en ProductSeeder toevoegen aan de DatabaseSeeder

```
3     namespace Database\Seeders;
4
5     use Illuminate\Database\Seeder;
6
7     class DatabaseSeeder extends Seeder
8     {
9         /**
10          * Seed the application's database. ...
11         */
12         public function run()
13         {
14             $this->call([
15                 RoleAndPermissionSeeder::class,
16                 UserSeeder::class,
17                 CategorySeeder::class,
18                 ProductSeeder::class,
19                 PriceSeeder::class
20             ]);
21         }
22     }
23 }
24 }
```

Nu kunnen we gaan testen. We zorgen dat we nieuwe tabellen hebben met verse data door middel van `php artisan migrate:fresh --seed`

```
D:\Wamp\sites\laravelv1>php artisan migrate:fresh --seed
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (16.77ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (13.91ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (14.19ms)
Migrating: 2021_03_12_195928_create_categories_table
Migrated: 2021_03_12_195928_create_categories_table (6.88ms)
Migrating: 2021_03_23_184829_update_name_length_categories_table
Migrated: 2021_03_23_184829_update_name_length_categories_table (60.79ms)
Migrating: 2021_04_01_200405_create_permission_tables
Migrated: 2021_04_01_200405_create_permission_tables (221.26ms)
Migrating: 2021_04_05_123055_create_products_table
Migrated: 2021_04_05_123055_create_products_table (31.63ms)
Migrating: 2021_04_05_123102_create_prices_table
Migrated: 2021_04_05_123102_create_prices_table (30.76ms)
```

Seeding: Database\Seeders\RoleAndPermissionSeeder
Seeded: Database\Seeders\RoleAndPermissionSeeder (76.23ms)
Seeding: Database\Seeders\UserSeeder
Seeded: Database\Seeders\UserSeeder (192.84ms)
Seeding: Database\Seeders\CategorySeeder
Seeded: Database\Seeders\CategorySeeder (7.74ms)
Seeding: Database\Seeders\ProductSeeder
Seeded: Database\Seeders\ProductSeeder (31.42ms)
Seeding: Database\Seeders\PriceSeeder
Seeded: Database\Seeders\PriceSeeder (24.53ms)
Database seeding completed successfully.

Nu kan de database worden gecontroleerd.

<b>id</b>	<b>name</b>	<b>description</b>	<b>category_id</b>
1	Else Spinka	Incidunt minus asperiores aliquid eveniet nulla do...	2
2	Jacquelyn Ritchie I	Placeat quod explicabo sed assumenda laudantium qu...	3
3	Roberta McGlynn	Explicabo odit recusandae dolor eius minus. Debiti...	3
4	Imani Yundt DVM	Dolore et ad est. Maiores voluptas enim id sint fu...	2
5	Johnnie Hessel	Sapiente accusantium et nam id cum. Ut doloribus s...	2
6	Maymie Koepp	Ut laboriosam qui rerum deleniti. Reiciendis vero ...	3
7	Abraham Price	Officiis eligendi et impedit accusamus ut. Dolores...	1
8	Dr. Jo Auer III	Rem sapiente temporibus consectetur. Id molestiae ...	3
9	Khalil Reinger	Nam velit id consequatur dolores neque. Aut dolor ...	1
10	Gertrude Hudson	Vel ipsum id ipsum nemo culpa ducimus nulla enim. ...	1

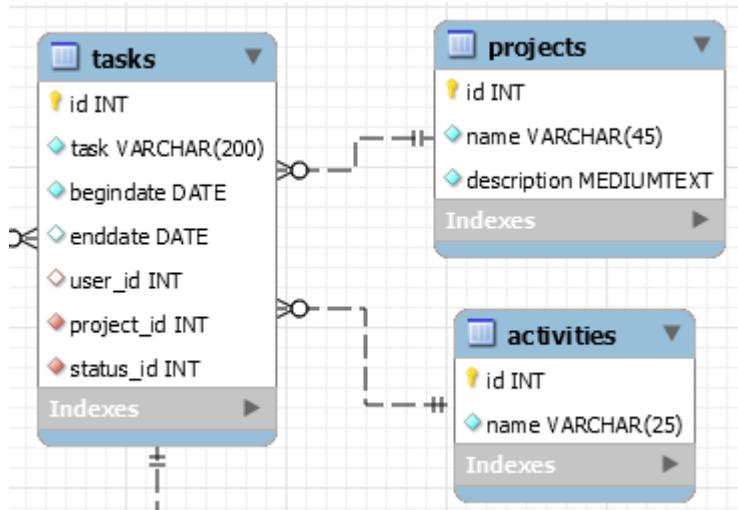
<b>id</b>	<b>price</b>	<b>effdate</b>	<b>product_id</b>
1	5.81	2007-11-18 16:00:52	1
2	5.29	1983-08-28 20:25:35	6
3	2.30	2019-03-28 23:50:06	3
4	3.59	1993-02-09 13:04:00	1
5	5.10	1977-04-15 18:22:03	4
6	3.81	2007-01-18 12:25:48	2
7	5.83	1973-03-08 02:41:35	6
8	3.26	2003-11-25 19:15:24	10
9	3.25	2008-06-03 05:51:24	1
10	3.95	2006-02-01 13:37:51	5

Zoals je ziet zijn de tabellen voor products en prices gevuld met ieder 10 rijen. Je ziet dat dit gewoon goed werkt. Er zit alleen een maar aan, want je ziet bijvoorbeeld met de testdata dat product met id=7, id=8 en id=9 geen prijzen hebben. Als we straks een overzicht willen maken met alle producten en hun huidige prijs gaan we in de problemen raken.

Het geeft niet als je nog niet begrijpt waarom. Op het moment wanneer we zover komen en er errors zijn, gaan we ook aan de oplossing werken.

## 5.6. Opdracht 13: Factory

Zorg dat de tabellen voor activities en tasks gevuld kunnen worden met data, door middel van een factory . Bij opdracht 12 heb je de migration gemaakt voor deze tabel.



Daarnaast heb je timestamps erbij gedaan. De timestamps worden automatisch gevuld als je deze niet invult in je factory. Gebruik in de factory faker om random data erin te zetten. Zorg dat de enddate 10 dagen na de begindate is.

Zorg dat je in het begin alle tasks met een user al koppelt, dit om het in het begin nog even makkelijk te houden.

### Opdracht controle

Als je de opdracht af hebt en gebruik hebt gemaakt van de opdrachten repository, kan je het volgende commando gebruiken om te controleren of alles goed werkt:

```
.\vendor\bin\pest --group=Opdracht12_13
```

## 5.7. Product Controller

### 5.7.1. Index

Bij de index gaan we meteen al aan de slag met relaties. Voor elk product wil ik namelijk laten zien in welke Category deze zit. Tevens zou ik de laatste Price willen laten zien. (met de afspraak dat we geen prijzen in de toekomst er nu inzetten)

We gaan dit wel stap voor stap doen. Als eerst gaan we alleen de producten ophalen en naar de view sturen.

```
9  class ProductController extends Controller
10 {
11     /**
12      * Display a listing of the resource. ...
13     */
14     public function index()
15     {
16         $products = Product::all();
17         return view('admin.products.index', compact('products'));
18     }
19 }
20 }
```

Voor de view, maken we eerst een map aan in de admin map voor products. We kunnen de index view van categories kopieren, want we hebben natuurlijk een soort gelijke pagina nodig.

Bij de index van products kunnen we wel meteen alle onderdelen toepassen die we bij categories ook hadden. Zoals bijvoorbeeld de topmenu. (we zullen wel zo de route moeten aanmaken, anders krijgen we errors)

```
1  @extends('layouts.layout')
2
3  @section('topmenu_items')
4      <!-- Top NavBar -->
5
6      <a href="{{ route('products.index') }}"
7          class="py-2 block text-green-500 border-green-500
8              dark:text-green-200 dark:border-green-200
9                  focus:outline-none border-b-2 font-medium capitalize
10                 transition duration-500 ease-in-out">
11
12         Index
13     </a>
14     <a href="{{ route('products.create') }}>
15         <button
16             class="ml-6 py-2 block border-b-2 border-transparent
17                 focus:outline-none font-medium capitalize text-center
18                     focus:text-green-500 focus:border-green-500
19                         dark-focus:text-green-200 dark-focus:border-green-200
20                             transition duration-500 ease-in-out">
21
22         Create
23     </button>
24     </a>
25
26     @endsection
```

De titel is aangepast naar Products.

```
@section('content')

    <div class="container mx-1">
        <div class="ml-2 flex flex-col">
            <h2 class="my-4 text-4xl font-semibold text-gray-600 dark:text-gray-400">
                Product Admin
            </h2>
        </div>

        @if(session('status'))
            <div class="bg-green-200 text-green-900 rounded-lg shadow-md p-6 pr-10 mb-8"
                style="...">
                {{ session('status') }}
            </div>
        @endif
    </div>
```

De routes van de linkjes zijn ook naar de product routes. In de foreach verwachten we een \$products array, die we doorlopen met \$product

```
<tbody class="bg-white divide-y divide-gray-200">
@foreach($products as $product)
    <tr>
        <td class="px-6 py-4 whitespace nowrap text-sm font-medium text-gray-900">
            {{ $product->name }}
        </td>
        <td class="px-6 py-4 whitespace nowrap text-sm text-gray-500">
            <a href="{{ route('products.show', ['product' => $product->id]) }}">Details</a>
        </td>
        <td class="px-6 py-4 whitespace nowrap text-sm text-gray-500">
            <a href="{{ route('products.edit', ['product' => $product->id]) }}">Edit</a>
        </td>
        @can('delete product')
            <td class="px-6 py-4 whitespace nowrap text-sm text-gray-500">
                <a href="{{ route('products.delete', ['product' => $product->id]) }}">Delete</a>
            </td>
        @endcan
    </tr>
@endforeach
</tbody>
```

Let even op, dat er ook al in staat, dat alleen met permissie 'delete product' we de delete kunnen zien.

De tabel zal straks uitgebreid worden met een aantal kolommen, zoals categorie en laatste prijs, maar eerst even een simpel overzicht voordat we het moeilijker maken.

Om nu de index te laten werken zullen we de routes nog moeten updaten

```
3  use Illuminate\Support\Facades\Route;
4
5  use App\Http\Controllers\Admin\CategoryController;
6
7  use App\Http\Controllers\Admin\ProductController;
8
9  /*
10 */
11
12
13
14
15
16
17
18  Route::get( uri: '/', [CategoryController::class, 'index']);
19
20  Route::group(['middleware' => ['role:customer|sales|admin']], function () {
21      Route::get( uri: 'admin/categories/{category}/delete', [CategoryController::class, 'delete'])
22          ->name( name: 'categories.delete');
23      Route::resource( name: '/admin/categories', controller: CategoryController::class);
24
25      Route::get( uri: 'admin/products/{product}/delete', [ProductController::class, 'delete'])
26          ->name( name: 'products.delete');
27      Route::resource( name: '/admin/products', controller: ProductController::class);
28  });
29
30  Route::get( uri: '/dashboard', function () {
31      return view( view: 'dashboard');
32  })->middleware(['auth'])->name( name: 'dashboard');
33
34  require __DIR__ . '/auth.php';
```

We zetten hem meteen binnen de middleware, want we zijn al ingelogd. De permissies in de controller zelf laten we nog even weg.

Als laatst nog even zorgen dat we wel vanuit onze site bij de pagina kunnen. Dan doen we in de layout (masterpage)

```
@else
    <li class="mt-8">
        <a href="{{ route('categories.index') }}" class="flex">
            <span class="ml-2 capitalize font-medium text-black dark:text-gray-300">Category Admin</span>
        </a>
    </li>
    <li class="mt-8">
        <a href="{{ route('products.index') }}" class="flex">
            <span class="ml-2 capitalize font-medium text-black dark:text-gray-300">Product Admin</span>
        </a>
    </li>
@endguest
```

Als we nu gaan uitproberen of we op de site kunnen en ook bij de product admin wat krijgen te zien, zien we dit:

NAME	DETAILS	EDIT
Else Spinka	Details	Edit
Jacquelyn Ritchie I	Details	Edit
Roberta McGlynn	Details	Edit
Imani Yundt DVM	Details	Edit
Johnnie Hessel	Details	Edit

Alles werkt netjes. Je ziet dat we geen delete zien , omdat daar wel permissie voor nodig is en die hebben we nog niet geregeld.

## 5.7.2. Installeren debugbar

Bij de uitbreiding van de index is het handig om te weten wat er allemaal op de achtergrond gebeurt. Hiervoor gaan we de debugbar installeren.

Binnen Laravel heb je een hele handige debug, namelijk de laravel debugbar. Deze wordt erg veel gebruikt als je aan het programmeren bent met Laravel. (op dit moment bij 28 miljoen downloads!!) (<https://github.com/barryvdh/laravel-debugbar>)

## Laravel Debugbar



This is a package to integrate [PHP Debug Bar](#) with Laravel. It includes a ServiceProvider to register the debugbar and attach it to the output. You can publish assets and configure it through Laravel. It bootstraps some Collectors to work with Laravel and implements a couple custom DataCollectors, specific for Laravel. It is configured to display Redirects and (jQuery) Ajax Requests. (Shown in a dropdown) Read the [documentation](#) for more configuration options.



Dit gaan we dan ook installeren binnen ons project. Dit doen we volgens de installatie die op Github staat. Met `composer require barryvdh/laravel-debugbar --dev`

```
D:\Wamp\sites\laravelv1>composer require barryvdh/laravel-debugbar --dev
Using version ^3.5 for barryvdh/laravel-debugbar
./composer.json has been updated
Running composer update barryvdh/laravel-debugbar
Loading composer repositories with package information
Updating dependencies
Lock file operations: 3 installs, 0 updates, 0 removals
- Locking barryvdh/laravel-debugbar (v3.5.4)
- Locking maximebf/debugbar (v1.16.5)
- Locking symfony/debug (v4.4.20)

Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 3 installs, 0 updates, 0 removals
- Downloading symfony/debug (v4.4.20)
- Downloading maximebf/debugbar (v1.16.5)
- Downloading barryvdh/laravel-debugbar (v3.5.4)
- Installing symfony/debug (v4.4.20): Extracting archive
- Installing maximebf/debugbar (v1.16.5): Extracting archive
- Installing barryvdh/laravel-debugbar (v3.5.4): Extracting archive
2 package suggestions were added by new dependencies, use `composer suggest` to see details.
```

Daarna even een check op dit:

The Debugbar will be enabled when APP\_DEBUG is true .

Binnen de .env file kunnen we dit vinden:

```
4 APP_DEBUG=true
```

Deze optie staat dus voor ons al aan. De debugbar zou nu al moeten werken. Zoniet, dan moet je de volgende stap ook nog doen.

Binnen de *config/app.php* gaan we dan zorgen dat de ServiceProvider gebruikt kan worden door middel van: *Barryvdh\Debugbar\ServiceProvider::class*,

```
165      /*
166       * Package Service Providers...
167     */
168     Spatie\Permission\PermissionServiceProvider::class,
169     Barryvdh\Debugbar\ServiceProvider::class,
170   /*
171    * Application Service Providers...
172   */
```

Willen we de log messages ook nog gebruiken, moeten we de debugbar ook nog toevoegen bij de facades. Deze staan ook nog in de config/app.php helemaal onderaan (class aliases). Hier voegen we dan aan toe:

'Debugbar' => *Barryvdh\Debugbar\Facade::class*,

```
228     'Validator' => Illuminate\Support\Facades\Validator::class,
229     'View' => Illuminate\Support\Facades\View::class,
230     'Debugbar' => Barryvdh\Debugbar\Facade::class,
231   ],
232
233 ];
```

Wil je later nog settings veranderen van de debugbar, kan je ook nog de laatste stap doen. Hiermee publish je de config voor de debugbar.

```
D:\Wamp\sites\laravelv1>php artisan vendor:publish --provider="Barryvdh\Debugbar\ServiceProvider"
Copied File [\vendor\barryvdh\laravel-debugbar\config\debugbar.php] To [\config\debugbar.php]
Publishing complete.
```

Als ik nu naar de site ga, zie ik onderin de browser allerlei dingen staan. Rechts onderin kan je dit uitvouwen, wat ik al heb gedaan. Ook ben ik al naar Queries gegaan, om te zien welke sql queries worden uitgevoerd als ik naar deze pagina ga. Je ziet nu bijvoorbeeld dat user 3 wordt opgevraagd (dat is de admin) en de rollen worden bekeken hiervan. Dit doen we nog allemaal in de masterpage. Daarna wordt pas de informatie van de products opgehaald.

The screenshot shows a web browser displaying a Laravel application at `laravelv1/admin/products`. On the left sidebar, there are links for "My Admin", "Hello Admin" (with a role dropdown), "Category Admin", and "Product Admin". The main content area is titled "Product Admin" and lists three users in a table:

NAME	DETAILS	EDIT
Else Spinka	Details	Edit
Jacquelyn Ritchie I	Details	Edit
Roberta McGlynn	Details	Edit

Below the table, a query profiler displays the following SQL statements:

```

5 statements were executed 0 29.96ms

select * from `users` where `id` = 3 0 25.76ms \vendor\laravel\framework\src\Illuminate\Auth\EloquentAuthProvider.php:52 laravelv1
limit 1

select column_name as `column_name` from
information_schema.columns where table_schema =
'laravelv1' and table_name = 'roles' 0 1.74ms \vendor\spatie\laravel-permission\src\Models\Role.php:26 laravelv1

select `roles`.*, `model_has_roles`.`model_id` as
`pivot_model_id`, `model_has_roles`.`role_id` as
`pivot_role_id`, `model_has_roles`.`model_type` as
`pivot_model_type` from `roles` inner join
`model_has_roles` on `roles`.`id` =
`model_has_roles`.`role_id` where
`model_has_roles`.`model_id` = 3 and
`model_has_roles`.`model_type` = 'App\Models\User'

select * from `products` 0 470μs \app\Http\Controllers\Admin\ProductController.php:18 laravelv1

select column_name as `column_name` from
information_schema.columns where table_schema =
'laravelv1' and table_name = 'permissions' 0 1.62ms \vendor\spatie\laravel-permission\src\Models\Permission.php:27 laravelv1

```

Bij de uitbreiding van de index gaan we hier gebruik van maken, zodat we zien wat er gebeurd.

### 5.7.3. Uitbreiden van de index

De eerste uitbreiding van de index zal de categorie worden. Dit is eigenlijk heel erg makkelijk binnen Laravel. De product heeft namelijk een relatie met category, waardoor we deze relatie kunnen gebruiken om bij de naam van de category te komen.

In de model hebben we de relatie al beschreven:

```
class Product extends Model
{
    use HasFactory;

    // relation to Category
    public function category()
    {
        return $this->belongsTo( related: Category::class);
    }
}
```

Hierdoor kunnen we nu categorie gebruiken in de index bij product.

```
<tbody class="bg-white divide-y divide-gray-200">
@foreach($products as $product)
<tr>
    <td class="px-6 py-4 whitespace nowrap text-sm font-medium text-gray-900">
        {{ $product->name }}
    </td>
    <td class="px-6 py-4 whitespace nowrap text-sm font-medium text-gray-900">
        {{ $product->category->name }}
    </td>
```

Je ziet hier dus \$product->category->name

Dit betekent, van dit product, met de relatie category, willen we de naam verkrijgen.

Als we dan nu naar onze index van products gaan kijken in de browser zien we netjes de categorie erbij staan.

The screenshot shows a Laravel application's admin panel. On the left, there's a sidebar with 'Hello Admin' and links for 'Role....', 'Category Admin', and 'Product Admin'. The main area is titled 'Product Admin' and lists four products with their names and categories. Below the table, a debug bar displays 15 database queries. The first query is a complex join from 'roles' to 'model\_has\_roles' and 'model\_type'. Subsequent queries show eager loading for categories, with multiple queries for each category ID (2, 3, 1) to fetch the associated products. The queries are timestamped and show they originate from 'laravelv1' and 'app\Http\Controllers\Admin\ProductController.php:18'.

Name	Category	Details	Edit
Else Spinka	Kaylah Reichel Sr.	Details	Edit
Jacquelyn Ritchie I	Dr. Sigurd Auer	Details	Edit
Roberta McGlynn	Dr. Sigurd Auer	Details	Edit
Imani Yundt DVM	Kaylah Reichel Sr.	Details	Edit

```
select `roles`.* , `model_has_roles`.`model_id` as `pivot_model_id` , `model_has_roles`.`role_id` as `pivot_role_id` , `model_has_roles`.`model_type` as `pivot_model_type` from `roles` inner join `model_has_roles` on `roles`.`id` = `model_has_roles`.`role_id` where `model_has_roles`.`model_id` = 3 and `model_has_roles`.`model_type` = 'App\Models\User'

select * from `products`

select column_name as `column_name` from information_schema.columns where table_schema = 'laravelv1' and table_name = 'permissions'

select * from `categories` where `categories`.`id` = 2 limit 1
select * from `categories` where `categories`.`id` = 3 limit 1
select * from `categories` where `categories`.`id` = 3 limit 1
select * from `categories` where `categories`.`id` = 2 limit 1
select * from `categories` where `categories`.`id` = 2 limit 1
select * from `categories` where `categories`.`id` = 3 limit 1
select * from `categories` where `categories`.`id` = 1 limit 1
```

Ik heb expres nu de debugbar open staan, want kijk is naar de hoeveelheid queries die uitgevoerd worden. Voordat we de categories hadden toegevoegd werden er 5 queries uitgevoerd en nu 15. Dit komt omdat per productregel een query wordt uitgevoerd om te kijken welke categorie erbij hoort. Bij Laravel is dit bekend als het N+1 probleem. Gelukkig hebben ze hier al wat op bedacht en dat eager loading.  
(<https://laravel.com/docs/8.x/eloquent-relationships#eager-loading>)

We veranderen de code naar dit:

```
9 class ProductController extends Controller
10 {
11     /**
12      * Display a listing of the resource. ...
13     */
14     public function index()
15     {
16         $products = Product::with('category')->get();
17
18         return view('admin.products.index', compact('products'));
19     }
20 }
```

Wat we hier aangeven is, haal alle producten op en maak gebruik van de methode category om ook meteen die gegevens op te halen.

Het resultaat op de pagina is precies hetzelfde

NAME	CATEGORY	DETAILS	EDIT
Else Spinka	Kaylah Reichel Sr.	Details	Edit
Jacquelyn Ritchie I	Dr. Sigurd Auer	Details	Edit
Roberta McGlynn	Dr. Sigurd Auer	Details	Edit
Imani Yundt DVM	Kaylah Reichel Sr.	Details	Edit

6 statements were executed 24.59ms

```
select * from `users` where `id` = 3 limit 1
select column_name as `column_name` from information_schema.columns where table_schema = 'laravelv1' and table_name = 'roles'
select `roles`.* , `model_has_roles`.`model_id` as `pivot_model_id` , `model_has_roles`.`role_id` as `pivot_role_id` , `model_has_roles`.`model_type` as `pivot_model_type` from `roles` inner join `model_has_roles` on `roles`.`id` = `model_has_roles`.`role_id` where `model_has_roles`.`model_id` = 3 and `model_has_roles`.`model_type` = 'App\Models\User'
select * from `products`
select * from `categories` where `categories`.`id` in (1, 2, 3)
select column_name as `column_name` from information_schema.columns where table_schema = 'laravelv1' and table_name = 'permissions'
```

Maar als je kijkt hoeveel queries worden uitgevoerd, is het nog maar 6. Het maakt nu niet uit hoeveel rijen we ophalen, want het getal zal niet groter worden, wat eerst wel het geval was omdat het voor elke rij een query uitvoerde. In de lay-out blijft de code gewoon hetzelfde, hierdoor werkt het ook gewoon meteen.

Voor de huidige prijs van een product wordt het lastiger. Dit omdat we met de relatie die we hebben beschreven dit niet kunnen doen.

```
class Product extends Model
{
    use HasFactory;

    // relation to Category
    public function category()
    {
        return $this->belongsTo( related: Category::class );
    }

    // relation to Price
    public function price()
    {
        return $this->hasMany( related: Price::class );
    }
}
```

Bij de methode price() staat namelijk dat we per product meerdere prijzen hebben. We willen maar 1 prijs laten zien en dat is namelijk de huidige prijs.

We kunnen het wel op een soortgelijke manier doen. Hiervoor gaan we eerst een nieuwe methode in de model van Product maken. De noemen we dan latest\_price

```
class Product extends Model
{
    use HasFactory;

    // relation to Category
    public function category()
    {
        return $this->belongsTo( related: Category::class);
    }

    // relation to Price
    public function price()
    {
        return $this->hasMany( related: Price::class);
    }

    public function latest_price()
    {
        return $this->hasOne( related: Price::class)->orderBy( column: 'effdate', direction: 'desc');
    }
}
```

Nu zie je bij latest\_price dat we geenhasMany hebben, maar hasOne. Dit komt omdat we maar 1 prijs terug willen krijgen. Om deze laatste prijs te krijgen sorteren we de prijzen van het product op effdate, en met de laatste datum willen we eerst. Deze wordt namelijk op dat moment alleen teruggegeven. Binnen de tabel in de index voegen we dit ook toe en gebruiken `$product->latest_price->price` om de info te krijgen

```
<tbody class="bg-white divide-y divide-gray-200">
@foreach($products as $product)
<tr>
    <td class="px-6 py-4 whitespace nowrap text-sm font-medium text-gray-900">
        {{ $product->name }}
    </td>
    <td class="px-6 py-4 whitespace nowrap text-sm font-medium text-gray-900">
        {{ $product->category->name }}
    </td>
    <td class="px-6 py-4 whitespace nowrap text-sm font-medium text-gray-900">
        {{ $product->latest_price->price }}
    </td>
```

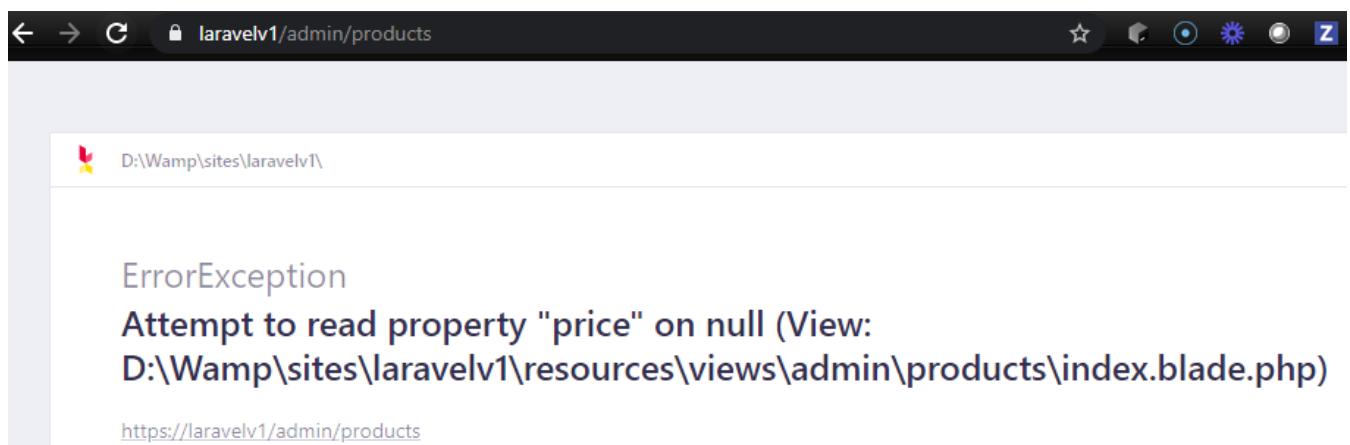
Binnen de controller gaan we gebruiken maken van Eager Loading voor categorie, maar ook de prijs.

```
class ProductController extends Controller
{
    /**
     * Display a listing of the resource. ...
     */
    public function index()
    {
        $products = Product::with('category', 'latest_price')->get();
        return view('admin.products.index', compact('products'));
    }
}
```

In ons overzicht op de pagina zie je dan nu ook de laatste prijs staan. De index is tot zo ver klaar.

NAME	CATEGORY	LATEST PRICE	DETAILS	EDIT
Else Spinka	Kaylah Reichel Sr.	5.81	Details	Edit
Jacquelyn Ritchie I	Dr. Sigurd Auer	3.81	Details	Edit
Roberta McGlynn	Dr. Sigurd Auer	2.30	Details	Edit
Imani Yundt DVM	Kaylah Reichel Sr.	5.10	Details	Edit
Johnnie Hessel	Kaylah Reichel Sr.	3.95	Details	Edit
At the bottom of the browser window, there are several icons and a status bar showing 'GET admin/products', '21MB', '201ms', and '8.0.3'." data-bbox="56 298 942 578"/>

Maar ik vermoed dat je een error krijgt en niet het resultaat dat hierboven staat. (Stiekem heb ik de data veranderd in de database, zodat elk product een prijs heeft, waardoor het bij mij werkt). Waarschijnlijk zie je deze fout:



Dit betekent, dat we een price willen uitlezen, maar die bestaat niet. We kunnen dit aan 2 kanten oplossen, of in onze index zodat je ook geen prijs bij een product kan hebben, of in de seeder zodat je altijd minimaal 1 prijs per product hebt. Mijn voorkeur gaat als eerst uit bij de seeder.

#### 5.7.4. Oplossen Seed probleem

Het seed probleem oplossen is eigenlijk heel simpel. We kunnen bij een seed aangeven dat een bepaalde factory uitgevoerd moet worden. Dit heeft ook wel, seeding met gebruik van model factories.

(<https://laravel.com/docs/8.x/seeding#using-model-factories>)

Bij onze product seed zorgen we ervoor dat er bijvoorbeeld 2 prijzen worden aangemaakt per product. Dit doen we met: `->hasPrice(2)`

```
5  use App\Models\Product;
6  use Illuminate\Database\Seeder;
7
8  class ProductSeeder extends Seeder
9  {
10     /**
11      * Run the database seeds. ...
12     */
13     public function run()
14     {
15         Product::factory()
16             ->times( count: 10 )
17             ->hasPrice(2)
18             ->create();
19     }
20 }
21
22 }
```

Als we dan opnieuw de database seeden.

```
D:\Wamp\sites\laravelv1>php artisan migrate:fresh --seed
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (15.11ms)
```

Als we nu kijken in de database zien we dat elk product minimaal 2 prijzen heeft.

<b>id</b>	<b>price</b>	<b>effdate</b>	<b>product_id</b>
1	5.47	1999-10-07 15:59:04	1
2	2.01	1985-02-15 07:17:30	1
3	4.67	2009-01-08 06:41:01	2
4	3.53	2011-01-02 23:04:35	2
5	3.08	2020-06-11 01:43:21	3
6	3.82	1993-02-21 07:52:32	3
7	2.79	1976-09-13 23:13:32	4
8	5.55	1986-06-10 08:37:35	4

Let op dat we ook nog de gewone price seeder uitvoeren, dus na de 2 prices per product komen ook nog de random rijen.

21	3.96	1982-03-28 19:09:28	10
22	5.55	2018-12-29 03:06:31	3
23	5.64	2009-07-12 12:30:45	10
24	4.08	1991-11-03 05:07:14	3
25	3.00	2009-01-23 14:56:58	2

Maar dit maakt op zich niet uit. Het resultaat ziet er wel goed uit en door eager loading worden er niet teveel queries uitgevoerd.

← → C 🔒 laravelv1/admin/products

My Admin  Index Create  Search

Hello Admin  
Role....

Category Admin

Product Admin

Logout

## Product Admin

NAME	CATEGORY	LATEST PRICE	DETAILS	EDIT
Prof. Marlen Lind	Elta Torphy	5.47	Details	Edit
Glennie Harvey	Ms. Lavada Mante	3.53	Details	Edit
Alyson Tremblay	Bret Nienow	3.08	Details	Edit
Miss Juliana Jacobs	Ms. Lavada Mante	5.55	Details	Edit
Filomena Beahan	Bret Nienow	4.39	Details	Edit
Alessia Howe DVM	Bret Nienow	3.34	Details	Edit
Dr. Antonina Steuber	Ms. Lavada Mante	4.65	Details	Edit
Mrs. Oceane Ortiz II	Ms. Lavada Mante	5.34	Details	Edit
Dr. Raymundo Lockman II	Elta Torphy	5.11	Details	Edit

GET admin/products 21MB 198ms 8.0.3

Het enige wat nu nog opgelost moet worden zijn de permisies, want we zien als admin nog niet de delete.

### 5.7.5. Opdracht 14: Seeder

Bij opdracht 12 en 13 zijn de migration en factories gemaakt voor activities en tasks. Nu wordt het tijd om de seeder te gaan maken, waarbij je netjes de koppeling maakt met activities, projects, users en tasks.

De users zijn al aangemaakt, net als de projects. Verder stond bij de hoofdopdracht het volgende:

- Een taak heeft een activity, bijvoorbeeld: Todo, Doing, Testing, Verify, Done

Zorg dat deze allemaal in de tabel activities komen door middel met de [ActivitySeeder](#), als het volgende:

id	name
1	Todo
2	Doing
3	Testing
4	Verify
5	Done

De volgende seeder zal je moeten aanpassen / aanmaken:

- ProjectSeeder: Project moet minimaal 2 tasks hebben. Er moeten minimaal 5 projecten aangemaakt worden.
- TaskSeeder: Er moeten minimaal 10 random tasks aangemaakt worden in de TaskSeeder

### Opdracht controle

Als je de opdracht af hebt en gebruik hebt gemaakt van de opdrachten repository, kan je het volgende commando gebruiken om te controleren of alles goed werkt:

```
.\vendor\bin\pest --group=Opdracht14
```

### 5.7.6. Opdracht 15: Index van Tasks

Maak voor de tabel tasks een admin door middel van een resource controller. Zorg dat je de index ziet binnen je eigen masterpage. Op de index moeten de volgende onderdelen te zien zijn:

- Task id
- De taak
- Startdatum
- Einddatum
- Naam van gebruiker
- Projectnaam
- Naam van de activity

De view van de index moet in de volgende map zitten en ook de correcte naam hebben:

- Resources/views/admin/tasks/index.blade.php

Je moet gebruik maken van de named routes die bij een resource controller aan gemaakt worden.

### Opdracht controle

Als je de opdracht af hebt en gebruik hebt gemaakt van de opdrachten repository, kan je het volgende commando gebruiken om te controleren of alles goed werkt:

```
.\vendor\bin\pest --group=Opdracht15
```

### 5.7.7. Permissies voor product controller

Om ervoor te zorgen dat de permissies voor products op orde zijn, zullen we als eerst de permissies moeten aanmaken. Dit doen we in de RoleAndPermissionSeeder.

```
26     Permission::create(['name' => 'delete category']);  
27  
28     // permissions for product CRUD  
29     Permission::create(['name' => 'index product']);  
30     Permission::create(['name' => 'show product']);  
31     Permission::create(['name' => 'create product']);  
32     Permission::create(['name' => 'edit product']);  
33     Permission::create(['name' => 'delete product']);  
34  
35     // customer role  
36     $customer = Role::create(['name' => 'customer']);
```

Het koppelen van deze permissies zal eronder gaan staan.

```
35     // customer role  
36     $customer = Role::create(['name' => 'customer']);  
37  
38     // sales role  
39     $sales = Role::create(['name' => 'sales'])  
40         ->givePermissionTo(['index category', 'show category', 'create category', 'edit category',  
41             'index product', 'show product', 'create product', 'edit product']);  
42  
43     // admin role  
44     $admin = Role::create(['name' => 'admin'])  
45         ->givePermissionTo(Permission::all());  
46 }  
47 }
```

De customer heeft nog steeds nergens recht op in de admin, de sales mag alleen niet verwijderen en de admin mag sowieso al alles. In de ProductController zullen we dan de permissies moeten koppelen aan de methodes.

```
9      class ProductController extends Controller  
10     {  
11         /** Set permissions on methods ... */  
12         public function __construct()  
13         {  
14             $this->middleware('auth');  
15             $this->middleware('permission:index product', ['only' => ['index']]);  
16             $this->middleware('permission:show product', ['only' => ['show']]);  
17             $this->middleware('permission:create product', ['only' => ['create', 'store']]);  
18             $this->middleware('permission:edit product', ['only' => ['edit', 'update']]);  
19             $this->middleware('permission:delete product', ['only' => ['delete', 'destroy']]);  
20         }  
21  
22         /** Display a listing of the resource. ... */  
23     }  
24 }
```

Als we nu de seed opnieuw uitvoeren zijn de permissies geregeld.

```
D:\Wamp\sites\laravelv1>php artisan migrate:fresh --seed
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (18.08ms)
```

En als admin zien we dan ook de delete vanzelf verschijnen bij de index van producten.

De index van producten is nu eindelijk af. Op naar de volgende onderdelen van de product CRUD!

### 5.7.8. Opdracht 16: Permissies voor tasks

Maak de volgende permissies aan.

Permissie naam	Methodes	Rollen met rechten
Task index	Index	Student, teacher, admin
Task create	Create + store	Student, teacher, admin
Task show	Show	Student, teacher, admin
Task edit	Edit + update	Student, teacher, admin
Task delete	Delete + destroy	Student, teacher, admin

Je zorgt dat in routes en in de controller de beveiliging is gemaakt voor deze permissies.

### Opdracht controle

Als je de opdracht af hebt en gebruik hebt gemaakt van de opdrachten repository, kan je het volgende commando gebruiken om te controleren of alles goed werkt:

```
.\vendor\bin\pest --group=Opdracht16
```

Er wordt nu alleen gecontroleerd of de rechten bij de index goed staan, de rest van de functionaliteiten worden later getest, waarbij de permissie ook meegenomen zal worden.

## 5.7.9. Create

Voor de create hebben we een formulier nodig, waarbij we de naam en beschrijving van het product aan kunnen geven en in welke categorie het product zit. Daarnaast hebben we natuurlijk een prijs nodig van het product.

Voor de categories gaan we een dropdown maken, waar je dan je categorie kan selecteren. Hiervoor hebben we wel alle categorien nodig. Deze moeten we meesturen naar de view.

```
36     /** Show the form for creating a new resource. ...*/
37
38     public function create()
39     {
40
41         $categories = Category::all();
42
43         return view('admin.products.create', compact('var_name: 'categories'));
44
45     }
46
```

Let even op, dat de Category model dan wel wordt gebruikt, waardoor dit wel aangeroepen moet worden.

```
5      use App\Http\Controllers\Controller;
6
7      use App\Models\Category;
8
9      use App\Models\Product;
10
11     use Illuminate\Http\Request;
```

Net zoals bij de index, kunnen we nu de create van category kopieren en deze aanpassen.

De titel moet wel weer aangepast worden, en de routes van de linkjes in het menu. Dit zou je ondertussen zelf wel moeten kunnen. Het formulier is natuurlijk wel een stukje groter dan bij categories.

```
<form id="form" class="bg-white shadow-md rounded-lg px-8 pt-6 pb-8 mb-4"
      action="{{ route('products.store') }}" method="POST">
    @csrf
    <div class="mb-4">
        <label class="block text-gray-700 text-sm font-bold mb-2" for="name">
            ProductName
        </label>
        <input
            class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
            focus:outline-none focus:shadow-outline @error('name') border-red-500 @enderror" name="name" id="name"
            value="{{ old('name') }}" type="text" required>
    </div>
    <div class="mb-4">
        <label class="block text-gray-700 text-sm font-bold mb-2" for="description">
            Description
        </label>
        <textarea
            class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
            focus:outline-none focus:shadow-outline @error('description') border-red-500 @enderror" name="description"
            id="description" required>{{ old('description') }}</textarea>
    </div>
```

```

<div class="mb-4">
    <label class="block text-gray-700 text-sm font-bold mb-2" for="price">
        Price
    </label>
    <input
        class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
        focus:outline-none focus:shadow-outline @error('price') border-red-500 @enderror" name="price" id="price"
        value="{{ old('price') }}" type="text" required>
</div>
<div class="mb-4">
    <label class="block text-gray-700 text-sm font-bold mb-2" for="category_id">
        Category
    </label>
    <select
        class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
        focus:outline-none focus:shadow-outline" name="category_id" id="category_id">
        @foreach($categories as $category)
            <option value="{{ $category->id }}>{{ $category->name }}</option>
        @endforeach
    </select>
</div>
<div class="flex items-center justify-between">
    <button id="submit"
        class="bg-green-500 hover:bg-green-700 text-white font-bold py-2 px-4 rounded
        focus:outline-none focus:shadow-outline" type="submit">Submit
    </button>
</div>
</form>

```

De action aangepast naar products.store, want daar gaan we het formulier opvangen.

De productname, description en price kunnen gewone velden zijn. Bij de dropdown willen we de id hebben van de categorie, omdat we deze straks als foreign key gaan opslaan.

Natuurlijk zijn de onderdelen die we al bij de categorie admin hebben gemaakt nu ook gebruikt. Zoals de @error en @enderror, en bijvoorbeeld de old('name'), old('description') en old('price').

Als we dan naar het resultaat gaan kijken, zien we een net formulier bij de create staan.

The screenshot shows a web browser window with the URL 'laravelv1/admin/products/create'. The page title is 'Product Admin'. On the left, there's a sidebar with a user profile picture, the text 'Hello Admin', and 'Role....'. Below that are links for 'Category Admin' and 'Product Admin'. The main content area has four input fields: 'ProductName' (empty), 'Description' (empty), 'Price' (empty), and 'Category' (a dropdown menu showing 'Prof. Lupe Medhurst'). A green 'Submit' button is at the bottom.

Bij de category hebben we netjes een dropdown gekregen. De create is dan ook hiermee klaar.

### 5.7.10. Store

Voor de store zullen we goed moeten kijken hoe we dit moeten gaan doen. Dit omdat de gegevens namelijk vanuit 1 formulier in 2 tabellen moeten komen.

Als we even kijken naar deze 2 tabellen hebben we dus de volgende gegevens nodig:

- Products: name, description, category\_id
- Price: price, effdate, product\_id

Wanneer we eerst het product opslaan in de database kunnen we deze id gebruiken om bij Price de product\_id in te vullen. De andere attributen krijgen we via het formulier binnen, behalve de effdate. Deze gaan we zo op een andere manier doen. We beginnen eerst met het opslaan van het product.

```
/** Store a newly created resource in storage. ...*/
public function store(Request $request)
{
    $product = new Product();
    $product->name = $request->name;
    $product->description = $request->description;
    $product->category_id = $request->category_id;
    $product->save();
}
```

Dit is nog niet zo bijzonder. Maar nu moeten we de prijs gaan opslaan. Als we een product aanmaken, gaan we ook meteen een prijs aanmaken, hierdoor wordt hiervoor ook een object aangemaakt.

```
    /** Store a newly created resource in storage. ...*/
    public function store(Request $request)
    {
        $product = new Product();
        $product->name = $request->name;
        $product->description = $request->description;
        $product->category_id = $request->category_id;
        $product->save();

        $price = new Price();
        $price->price = $request->price;
        $price->effdate = Carbon::now();
        $price->product_id = $product->id;
        $price->save();
    }
}
```

Je ziet hier dat we met new Price() een object aanmaken. Verder gebruik ik Carbon voor een datum (<https://carbon.nesbot.com/>). Carbon zit al standaard in Laravel, dus ook erg makkelijk om te gebruiken. Zorg er wel voor dat het ook bij de use staat:

```
3     namespace App\Http\Controllers\Admin;
4
5     use App\Http\Controllers\Controller;
6     use App\Models\Category;
7     use App\Models\Price;
8     use App\Models\Product;
9     use Carbon\Carbon;
10    use Illuminate\Http\Request;
11
12    class ProductController extends Controller
```

Voor de product\_id gebruik ik natuurlijk de variabel \$product. Hier is een id nu beschikbaar omdat we al save() hebben gedaan.

Natuurlijk hebben we validatie nodig, dus hiervoor maken we een request aan.

```
D:\Wamp\sites\laravelv1>php artisan make:request ProductStoreRequest
Request created successfully.
```

Bij de request moeten we altijd goed kijken naar wat er in de database kan. De naam van het product mag maximaal 45 characters zijn. Verder kan een prijs maar 6 cijfers voor en 2 achter de komma hebben. Helaas zit er niet een standaard validatie voor komma getallen in Laravel en wil ik nu nog niet met custom validatie gaan beginnen. Wat ik wel kan aangeven is het maximale getal en dit is 999999.99 euro

```
class ProductStoreRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request. ...
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request. ...
     */
    public function rules()
    {
        return [
            'name' => 'required|max:45|unique:products',
            'description' => 'required',
            'price' => 'required|numeric|max:999999.99'
        ];
    }
}
```

Als we nu naar het geheel kijken van de store methode zien we dit.

```
/** Store a newly created resource in storage. ...*/
public function store(ProductStoreRequest $request)
{
    $product = new Product();
    $product->name = $request->name;
    $product->description = $request->description;
    $product->category_id = $request->category_id;
    $product->save();

    $price = new Price();
    $price->price = $request->price;
    $price->effdate = Carbon::now();
    $price->product_id = $product->id;
    $price->save();

    return redirect()->route('products.index')->with('status', 'Product toegevoegd');
}
```

De request gebruiken we nu als validatie. Na het opslaan gaan we terug naar de index met een bericht dat het product is toegevoegd. We kunnen nu testen of dit werkt. Bij de create vul ik dus wat testdata in het formulier.

Index [Create](#) Search

## Product Admin

**ProductName**  
Product 1

**Description**  
Dit is een kleine beschrijving

**Price**  
80.58

**Category**  
Lorenzo Schultz

**Submit**

We krijgen ook netjes een bericht dat het product is toegevoegd.

Index [Create](#) Search

## Product Admin

Product toegevoegd

En het product zie je ook meteen in de index van producten.

Freeda Moore	Prof. Ernesto Ruecker III	2.16	Details	Edit	Delete
Product 1	Lorenzo Schultz	80.58	Details	Edit	Delete

Probeer ik een grotere prijs dan die we hebben ingevuld bij de request, krijg ik netjes een foutmelding

[Index](#) [Create](#)

## Product Admin

- The price may not be greater than 999999.99.

**ProductName**

Product 2

**Description**

Dit is een 2e test

**Price**

19999999.99

**Category**

Prof. Lupe Medhurst

**Submit**

Waar we even nog naar moeten kijken, is dat je bij de categorie ook de oude waarde krijgt. Dit is namelijk nog niet zo. Dit kunnen we bij een select op de volgende manier doen:

```
<select  
    class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight  
    focus:outline-none focus:shadow-outline" name="category_id" id="category_id">  
    @foreach($categories as $category)  
        <option value="{{ $category->id }}"  
            @if( old('category_id') == $category->id)  
                selected  
            @endif  
        >{{ $category->name }}</option>  
    @endforeach  
</select>
```

Nu we dit hebben zijn we klaar met de store, want validatie wordt nu geheel goed verwerkt.

#### 5.7.11. Opdracht 17: Create & store voor tasks met validatie

Maak binnen de resource controller van tasks de create en store methode en zorg dat het binnen je eigen masterpage werkt.

Op de create zal een formulier beschikbaar moeten zijn die post naar de store methode. In het formulier zijn de volgende inputs:

- Task: text
- Begindate: Datum
- Enddate: Datum
- User\_id: select
- Project\_id: select
- Activity\_id: select

De selects zullen alle drie een dropdown moeten worden.

De view van de create moet in de volgende map zitten en ook de correcte naam hebben:

- Resources/views/admin/tasks/create.blade.php

Als je het formulier hebt verstuurd kom je bij de store methode. Hier zal de data worden opgeslagen in de tabel tasks. Je moet gebruik maken van de named routes die bij een resource controller aan gemaakt worden. Bij de validatie wordt gecontroleerd op de volgende voorwaarden:

- De task is ingevuld, minimaal 10 karakters, maximaal 200 karakters
- Begindate moet ingevuld zijn, enddate is niet verplicht.
- De user is niet verplicht. Er moet een project en activity aan gekoppeld staan en die moeten een integer zijn.

Als er niet aan de validatie wordt voldaan, moet het zichtbaar in het formulier zijn welke onderdelen fout zijn. De foutmeldingen moeten binnen de lay-out op het scherm komen.

Als het aan de validatie voldoet moet het worden opgeslagen in de tabel tasks. Er zal een redirect moeten komen naar de index met een melding dat de task is opgeslagen.

## Opdracht controle

Als je de opdracht af hebt en gebruik hebt gemaakt van de opdrachten repository, kan je het volgende commando gebruiken om te controleren of alles goed werkt:

```
.\vendor\bin\pest --group=Opdracht17
```

### 5.7.12. Show

De show wordt niet zo moeilijk in de controller. We willen het Product hebben, die al in de variabel \$product staat vanwege de model binding. Hierdoor kan meteen het product naar de view toe.

```
    /**
     * Display the specified resource. ...
     */
    public function show(Produkt $product)
    {
        return view('admin.products.show', compact('var_name: 'product'));
    }
```

In de view hebben we de show van categorie gekopieerd en aangepast. Natuurlijk het menu weer naar products veranderd. Dan de card, waarin alles van het product komt. Je kan ervoor kiezen om alleen de huidige prijs te laten zien. Dit hebben we al in de index gebruikt, dus is nu weer te gebruiken.

```
<!-- component -->
<div class="max-w-sm bg-white shadow-lg rounded-lg overflow-hidden my-4">
    
        <h1 class="mx-3 text-white font-semibold text-lg">{{ $product->name }}</h1>
    </div>
    <div class="py-4 px-6">
        <h1 class="text-2xl font-semibold text-gray-800"></h1>
        <p class="py-2 text-lg text-gray-700">Category: {{ $product->category->name }}</p>
        <p class="py-2 text-lg text-gray-700">Latest Price: {{ $product->latest_price->price }}</p>
        <p class="py-2 text-lg text-gray-700">{{ $product->description }}</p>
    </div>
</div>
```

Maar waarom bewaren we nu de geschiedenis van prijzen dan. Is het dan niet handig om de prijzen van dit product dan hier te zien, zodat je het verloop kan bekijken. Dit gaan we dan ook doen.

Zouden we dan hiervoor alle prijzen die bij het product horen moeten ophalen?? Het antwoord is nee. Dit omdat we de relatie gewoon in onze model hebben, krijgen we een collection terug van prijzen.

```
// relation to Price
public function price()
{
    return $this->hasMany('Price::class');
}
```

Op dit moment ga ik niet te diep in op een collection. (eigenlijk gebruiken we dit al heel vaak nu)

Het stukje van Prices wordt dan veranderd. Hier hebben we een tabel nodig.

```
<tbody class="bg-white divide-y divide-gray-200">
@foreach($product->price->sortByDesc('effdate') as $item)
<tr>
    <td class="px-6 py-4 whitespace nowrap text-sm font-medium text-gray-900">
        {{ $item->id }}
    </td>
    <td class="px-6 py-4 whitespace nowrap text-sm font-medium text-gray-900">
        {{ $item->price }}
    </td>
    <td class="px-6 py-4 whitespace nowrap text-sm font-medium text-gray-900">
        {{ $item->effdate }}
    </td>
</tr>
@endforeach
</tbody>
<p class="py-2 text-lg text-gray-700">{{ $product->description }}</p>
```

Je ziet in de @foreach dat ik van het product de relatie price gebruik. De prijzen wil ik nog wel gesorteerd bij de datum wanneer deze ingaat, waarbij dan meeste recente datum bovenaan komt. Hiervoor gebruiken we dan de sortByDesc functie. Als we naar het resultaat gaan kijken ziet het er nu zo uit:

## Product Admin



Mr. Colten Doyle V

Category: Prof. Ernesto Ruecker III

#	PRICE	DATE
2	5.41	2001-08-06 02:29:44
24	3.05	1991-02-17 20:36:58
1	3.50	1977-06-04 00:40:33

Perspiciatis modi aut quaerat adipisci dolores nihil ut. Rerum omnis voluptatem nihil doloremque laudantium natus consequuntur. In sit sapiente velit qui. Tempore id soluta voluptas repellendus et

Misschien in opmaak kan het beter, maar het resultaat is er. Lay-out kan altijd nog aangepast worden.

### 5.7.13. Opdracht 18: Show voor task

Maak de show voor tasks. Zorg dat je de show ziet binnen je eigen masterpage. Op de show moeten de volgende onderdelen te zien zijn:

- Task id
- De task
- Begindatum
- Einddatum, indien beschikbaar
- De user met naam, indien beschikbaar
- De projectnaam
- De activity status
- Datum van het aanmaken van de task

De view van de show moet in de volgende map zitten en ook de correcte naam hebben:

- Resources/views/admin/tasks/show.blade.php

Je moet gebruik maken van de named routes die bij een resource controller aan gemaakt worden.

#### Opdracht controle

Als je de opdracht af hebt en gebruik hebt gemaakt van de opdrachten repository, kan je het volgende commando gebruiken om te controleren of alles goed werkt:

```
.\vendor\bin\pest --group=Opdracht18
```

### 5.7.14. Edit

Voor de edit kunnen we gaan kijken wat we bij de create hebben gedaan en tevens naar de edit van categorie. Als eerst is het slim om weer een goed beeld te hebben wat we willen zien. Een ingevuld formulier van een product, waarbij we een dropdown hebben voor categorie. Vanwege de dropdown zullen we wel alle categories mee moeten sturen.

De controller gaat er dan zo uitzien.

```
/** Show the form for editing the specified resource. ...*/
public function edit(Product $product)
{
    $categories = Category::all();
    return view('admin.products.edit', compact('product', ...$varNames, 'categories'));
}
```

In de edit.blade kopieren we als eerst wat we in de create hebben staan. Het menu is hetzelfde als bij de create, dus de wijzigingen zullen in het formulier zijn.

Als eerst zorgen we ervoor dat de action wordt veranderd naar products.update, waarbij we de id meegeven. De methode PUT zal er ook bij moeten.

```
<form id="form" class="bg-white shadow-md rounded-lg px-8 pt-6 pb-8 mb-4"
      action="{{ route('products.update', ['product' => $product->id]) }}" method="POST">
    @method('PUT')
    @csrf
    <div class="mb-4">
        <label class="block text-gray-700 text-sm font-bold mb-2" for="name">
            ProductName
        </label>
        <input
            class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
            focus:outline-none focus:shadow-outline @error('name') border-red-500 @enderror" name="name" id="name"
            value="{{ old('name', $product->name) }}" type="text" required>
    </div>
```

Je ziet ook bij de input dat ik een kleine wijziging heb gedaan ten opzichte van de create. In de create hebben we met old() gewerkt. We gebruiken dit natuurlijk weer, alleen hebben we nu ook te maken met data uit de database. Bij old() kan je eerst de oude data krijgen en als die er niet is, krijg je de database data. Erg handig voor nu!

In de rest van het formulier eigenlijk hetzelfde gedaan bij alle inputs.

```
<div class="mb-4">
    <label class="block text-gray-700 text-sm font-bold mb-2" for="description">
        Description
    </label>
    <textarea
        class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
        focus:outline-none focus:shadow-outline @error('description') border-red-500 @enderror" name="description"
        id="description" required>{{ old('description', $product->description) }}</textarea>
</div>
<div class="mb-4">
    <label class="block text-gray-700 text-sm font-bold mb-2" for="price">
        Price
    </label>
    <input
        class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
        focus:outline-none focus:shadow-outline @error('price') border-red-500 @enderror" name="price" id="price"
        value="{{ old('price', $product->latest_price->price) }}" type="text" required>
</div>
<div class="mb-4">
    <label class="block text-gray-700 text-sm font-bold mb-2" for="category_id">
        Category
    </label>
    <select
        class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
        focus:outline-none focus:shadow-outline" name="category_id" id="category_id">
        @foreach($categories as $category)
            <option value="{{ $category->id }}"
                @if( old('category_id', $product->category_id) == $category->id)
                    selected
                @endif
            >{{ $category->name }}</option>
        @endforeach
    </select>
</div>
<div class="flex items-center justify-between">
    <button id="submit"
        class="bg-green-500 hover:bg-green-700 text-white font-bold py-2 px-4 rounded
        focus:outline-none focus:shadow-outline" type="submit">Update
    </button>
</div>
```

Als we dit uitproberen zal je bij de edit een net ingevuld formulier zien.

# Product Admin

**ProductName**

Mr. Colten Doyle V

**Description**

Perspiciatis modi aut quaerat adipisci dolores nihil ut. Rerum omnis voluptatem nihil doloremque laudantium natus consequuntur. In sit

**Price**

5.41

**Category**

Prof. Ernesto Ruecker III

**Update**

Precies zoals we het willen hebben.

### 5.7.15. Update

Bij de update zal even opgelet moeten worden. Als er een nieuwe prijs wordt ingevuld, moet deze prijs niet gewijzigd worden, maar moet er een nieuwe prijs aangemaakt worden. Dit omdat we de geschiedenis willen bewaren.

```
/** Update the specified resource in storage. ...*/
public function update(Request $request, Product $product)
{
    // er is al een bestaand $product
    $product->name = $request->name;
    $product->description = $request->description;
    $product->category_id = $request->category_id;
    $product->save();

    // bij de update gaan we een nieuwe prijs erin zetten (dus niet een oude waarde aanpassen)
    // Alleen uitvoeren als er een nieuwe prijs is ingevuld, dus niet met de huidige prijs.
    if($product->latest_price->price != $request->price)
    {
        $price = new Price();
        $price->price = $request->price;
        $price->effdate = Carbon::now();
        $price->product_id = $product->id;
        $price->save();
    }

    return redirect()->route('products.index')->with('status', 'Product geupdate');
}
```

De gegevens van \$product worden opgeslagen bij product. Er wordt gekeken of de prijs die we invullen anders is dan de huidige prijs. Zoja, dan wordt er een nieuwe prijs aan het product gekoppeld.

Verder heeft een product een naam die uniek is. Hierdoor zullen we een update request nodig hebben, want anders krijgen we een error als we bijvoorbeeld alleen de prijs updaten. De error zou zeggen dat we het product niet met dezelfde naam op mogen slaan. Hiervoor maken we dus een nieuwe Request aan: ProductUpdateRequest.

```
D:\Wamp\sites\laravelv1>php artisan make:request ProductUpdateRequest
Request created successfully.
```

In de update request zorgen we er dan voor dat hij niet naar de huidige naam van het product kijkt.

```
class ProductUpdateRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request. ...
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request. ...
     */
    public function rules()
    {
        $product = $this->route('product');
        return [
            'name' => 'required|max:45|unique:products,name,' . $product->id,
            'description' => 'required',
            'price' => 'required|numeric|max:999999.99'
        ];
    }
}
```

Nu de request klaar is kunnen we deze in de update methode neer zetten.

```
/** Update the specified resource in storage. ...
public function update(ProductUpdateRequest $request, Product $product)
{
    // er is al een bestaand $product
    $product->name = $request->name;
```

We hebben de ProductUpdateRequest gebruikt, dus vergeet niet dat deze bij use moet komen boven de class.

```
5   use App\Http\Controllers\Controller;
6   use App\Http\Requests\ProductStoreRequest;
7   use App\Http\Requests\ProductUpdateRequest;
8   use App\Models\Category;
9   use App\Models\Price;
10  use App\Models\Product;
11  use Carbon\Carbon;
12  use Illuminate\Http\Request;
```

Nu kunnen we de edit en update uitproberen. We veranderen alles wat bij id=1 stond.

The screenshot shows a web browser window with the URL `laravelv1/admin/products/1/edit`. The page title is "Product Admin". On the left, there is a sidebar with the word "Admin" repeated three times. The main content area has tabs for "Index" and "Create", with "Create" being active. There is a search bar at the top right. The form contains fields for "ProductName" (containing "Dit is edit test 1"), "Description" (containing "Blablabla"), "Price" (containing "6.78"), and "Category" (a dropdown menu set to "Lorenzo Schultz"). A green "Update" button is at the bottom of the form.

Het resultaat ziet er goed uit.

The screenshot shows a web browser window with the URL `laravelv1/admin/products`. The page title is "Product Admin". The sidebar on the left shows "Admin" repeated. The main content area has tabs for "Index" and "Create", with "Index" being active. A green banner at the top says "Product geupdate". Below it is a table with columns: NAME, CATEGORY, LATEST PRICE, DETAILS, EDIT, and DELETE. The table has two rows. The first row contains "Dit is edit test 1", "Lorenzo Schultz", "6.78", "Details", "Edit", and "Delete". The second row contains "Dr. Mariane Wisoky Sr.", "Lorenzo Schultz", "3.26", "Details", "Edit", and "Delete".

NAME	CATEGORY	LATEST PRICE	DETAILS	EDIT	DELETE
Dit is edit test 1	Lorenzo Schultz	6.78	Details	Edit	Delete
Dr. Mariane Wisoky Sr.	Lorenzo Schultz	3.26	Details	Edit	Delete

Als we dan proberen een error te krijgen, door een te grote prijs te gebruiken. Expres de product naam van id=2 gekozen om bij product met id = 1 neer te zetten. Verder de prijs groter dan het maximum te zetten.

laravelv1/admin/products/1/edit

Index Create

## Product Admin

Product Name: Dr. Mariane Wisoky Sr.

Description: Blablabla

Price: 1999999.00

Category: Lorenzo Schultz

**Update**

Het resultaat wat we willen. Een foutmelding met wat er anders moet en de velden die moeten veranderen met een rode rand.

## Product Admin

- The name has already been taken.
- The price may not be greater than 999999.99.

Product Name: Dr. Mariane Wisoky Sr.

Description: Blablabla

Price: 1999999.00

Category: Lorenzo Schultz

**Update**

De edit en update zijn nu geheel klaar.

### 5.7.16. Opdracht 19: Edit & update voor tasks

Maak binnen de resource controller van tasks de edit en update methode en zorg dat het binnen je eigen masterpage werkt.

Op de update zal een formulier beschikbaar moeten zijn die post (+ PUT/PATCH) naar de update methode. In het formulier zijn de volgende inputs die al ingevuld moeten zijn:

- Task: text
- Begindate: Datum
- Enddate: Datum
- User\_id: select
- Project\_id: select
- Activity\_id: select

De selects zullen alle drie een dropdown moeten worden, waarbij de huidige waarde geselecteerd is.

De view van de edit moet in de volgende map zitten en ook de correcte naam hebben:

- Resources/views/admin/tasks/edit.blade.php

Als je het formulier hebt verstuurd kom je bij de update methode. Bij de validatie wordt gecontroleerd op de volgende voorwaarden:

- De task is ingevuld, minimaal 10 karakters, maximaal 200 karakters
- Begindate moet ingevuld zijn, enddate is niet verplicht.
- De user is niet verplicht. Er moet een project en activity status aan gekoppeld staan en die moeten een integer zijn.

Als er niet aan de validatie wordt voldaan, moet het zichtbaar in het formulier zijn welke onderdelen fout zijn. De foutmeldingen moeten binnen de lay-out op het scherm komen.

Als het aan de validatie voldoet moet het worden opgeslagen in de tabel tasks. Er zal een redirect moeten komen naar de index met een melding dat de task is opgeslagen.

Je moet gebruik maken van de named routes die bij een resource controller aan gemaakt worden.

#### Opdracht controle

Als je de opdracht af hebt en gebruik hebt gemaakt van de opdrachten repository, kan je het volgende commando gebruiken om te controleren of alles goed werkt:

```
.\vendor\bin\pest --group=Opdracht19
```

### 5.7.18. Delete

Voor het verwijderen van producten gaan we aan de slag met de delete, net zoals bij categorie. Dit omdat we toch weer een bevestiging willen maken dat we het juiste product verwijderen.

De methode moet dan natuurlijk weer in de controller worden toegevoegd.

```
/** Show the form for deleting the specified resource. ...*/
public function delete(Product $product)
{
    return view( view: 'admin.products.delete', compact( var_name: 'product'));
}
```

In de controller kunnen we meteen het product naar het formulier sturen.

Voor de view gaan we de edit.blade kopieren naar de delete.blade, want we hebben een ingevuld formulier nodig.

Voor het formulier zorgen we dat de velden disabled zijn. Voor de categorie hebben we geen dropdown nodig, dus dit zou gewoon in een input kunnen. De old() methode hebben we ook niet nodig, omdat we hier sowieso niks kunnen wijzigen.

Let even op dat je de route en @method nog even veranderd!

```
<form id="form" class="bg-white shadow-md rounded-lg px-8 pt-6 pb-8 mb-4"
      action="{{ route('products.destroy', ['product' => $product->id]) }}" method="POST">
    @method('DELETE')
    @csrf
    <div class="mb-4">
        <label class="block text-gray-700 text-sm font-bold mb-2" for="name">
            ProductName
        </label>
        <input class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" name="name" id="name"
               value="{{ $product->name }}" type="text" disabled>
    </div>
    <div class="mb-4">
        <label class="block text-gray-700 text-sm font-bold mb-2" for="description">
            Description
        </label>
        <textarea class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" name="description"
                  id="description" disabled>{{ $product->description }}</textarea>
    </div>
```

```

<div class="mb-4">
    <label class="block text-gray-700 text-sm font-bold mb-2" for="price">
        Price
    </label>
    <input class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-normal focus:outline-none focus:shadow-outline" name="price" id="price" value="{{ $product->latest_price->price }}" type="text" disabled>
</div>
<div class="mb-4">
    <label class="block text-gray-700 text-sm font-bold mb-2" for="category_id">
        Category
    </label>
    <input class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-normal focus:outline-none focus:shadow-outline" name="category_id" id="category_id" value="{{ $product->category->name }}" type="text" disabled>
</div>
<div class="flex items-center justify-between">
    <button id="submit" class="bg-green-500 hover:bg-green-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline" type="submit">Delete
    </button>
</div>
</form>

```

Als we gaan kijken hoe dit er dan uit ziet, zien we een net formulier met gegevens erin, die we kunnen deleten.

## Product Admin

ProductName

Meredith Goodwin I

Description

Occaecati repellendus omnis sunt. Omnis libero magnam cupiditate a sed nihil. Ut dolor illum est modi qui consectetur at. Veniam nostrum quia nesciunt fugit. Optio commodi et ea nihil est dolores beatae.

Price

5.71

Category

Alexandria Barton

Delete

### 5.7.19. Destroy

Het verwijderen van een product is op zich simpel. We zeggen gewoon, delete().

```
/** Remove the specified resource from storage. ...*/
public function destroy(Product $product)
{
    $product->delete();
    return redirect()->route('products.index')->with('status', 'Product deleted');
}
```

Verder zorgen we natuurlijk dat we een bericht krijgen dat het product is verwijderd.

Als we dit dan uittesten, zien we dat het product verwijderd is en we weer op de index zijn. Het product met id = 1 is nu verwijderd, wat ook de bedoeling was.

## Product Admin

Product Admin					
Name	Category	Latest Price	Details	Edit	Delete
Reggie Carroll	Prof. Keyon Howe	3.36	Details	Edit	Delete
Verona Kassulke	Alexandria Barton	3.94	Details	Edit	Delete

En weet je wat het mooie is. De prices die er aan gekoppeld zijn, staan ook niet meer in de database.

ID	Price	Effdate	Product ID	Created At	Updated At
3	3.36	1992-12-01 13:49:16	2	2021-04-10 11:50:38	2021-04-10 11:50:38
4	3.80	1972-11-24 18:28:01	2	2021-04-10 11:50:38	2021-04-10 11:50:38
5	3.80	1992-03-27 18:01:13	3	2021-04-10 11:50:38	2021-04-10 11:50:38
6	5.37	1999-11-08 10:03:05	3	2021-04-10 11:50:38	2021-04-10 11:50:38
7	4.44	1977-06-12 08:47:16	4	2021-04-10 11:50:38	2021-04-10 11:50:38

Dit is het mooie aan database relaties goed configureren. In de migration hebben we dit aangegeven bij prices.

```
public function up()
{
    Schema::create('prices', function (Blueprint $table) {
        $table->id();
        $table->decimal('price', 8, 2);
        $table->dateTime('effdate');
        $table->foreignId('product_id')->constrained()
            ->onUpdate('cascade')->onDelete('cascade');
        $table->timestamps();
    });
}
```

Dus, zodra een product wordt verwijderd, is de relatie bekent, staat er onDelete('cascade') erbij, waardoor ook meteen de prices die aan het product stonden gekoppeld verwijderd worden. De database blijft dan netjes op orde, want anders zouden we prices hebben die niet aan een product gekoppeld stonden. Dit heet ook wel, data integriteit.

#### 5.7.20. Opdracht 20: Delete & destroy voor tasks

Maak binnen de resource controller van tasks de delete en destroy methode en zorg dat het binnen je eigen masterpage werkt.

Op de delete zal een formulier beschikbaar moeten zijn die post (+ DELETE) naar de destroy methode. In het formulier zijn de volgende inputs die al ingevuld moeten zijn:

- Task: text
- Begindate: Datum
- Enddate: Datum
- User\_id: select
- Project\_id: select
- Activity\_id: select

De selects zullen alle drie een dropdown moeten worden, waarbij alleen de huidige waarde een optie is. Alle inputs moeten disabled zijn.

De view van de delete moet in de volgende map zitten en ook de correcte naam hebben:

- Resources/views/admin/tasks/delete.blade.php

Als je het formulier hebt verstuurd kom je bij de destroy methode.

Bij de destroy methode zal de task worden verwijderd. Er zal een redirect moeten komen naar de index met een melding dat de task is verwijderd. Je moet gebruik maken van de named routes die bij een resource controller aan gemaakt worden.

#### Opdracht controle

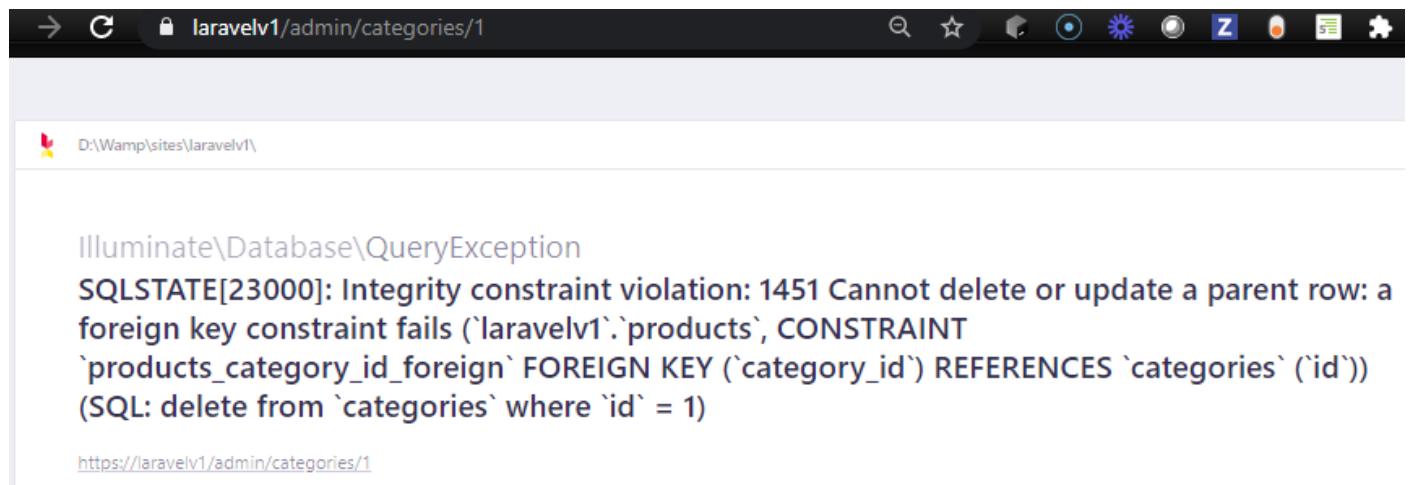
Als je de opdracht af hebt en gebruik hebt gemaakt van de opdrachten repository, kan je het volgende commando gebruiken om te controleren of alles goed werkt:

```
.\vendor\bin\pest --group=Opdracht20
```

### 5.7.21. Category Test

Nu de hele crud van Product af is, is het vaak slim of even terug te kijken of alles verder nog werkt. Een product heeft namelijk een categorie, waardoor we misschien iets hebben gedaan waardoor er een onderdeel in de Category CRUD het niet meer doet.

Bij het testen kom je er achter dat de delete van een categorie het niet meer doet, zodra er een product aan gekoppeld is. Bij het verwijderen krijgen we deze error.



```
Illuminate\Database\QueryException
SQLSTATE[23000]: Integrity constraint violation: 1451 Cannot delete or update a parent row: a
foreign key constraint fails ('laravelv1`.`products`, CONSTRAINT
`products_category_id_foreign` FOREIGN KEY (`category_id`) REFERENCES `categories` (`id`))
(SQL: delete from `categories` where `id` = 1)

https://laravelv1/admin/categories/1
```

Deze error komt namelijk doordat we in de database een restrictie hebben opgelegd.  
In de product migration staat dit

```
public function up()
{
    Schema::create('products', function (Blueprint $table) {
        $table->id();
        $table->string('name', 45);
        $table->text('description');
        $table->foreignId('category_id')->constrained()
            -> onUpdate('restrict')->onDelete('restrict');
        $table->timestamps();
    });
}
```

Dit betekent, dat als er nog een product aan een category vast zit, de relatie zegt: restrict.

Je mag op dat moment dus niet de categorie verwijderen, omdat we anders producten krijgen die in een niet bestaande categorie zitten. Dit kan natuurlijk niet.

Wel is het netjes, om in ieder geval een nette foutmelding te krijgen, waarin staat dat er bijvoorbeeld nog producten aan de categorie staan gekoppeld.

In de categorie controller gaan we dan ook de destroy aanpassen.

```
/** Remove the specified resource from storage. ...*/
public function destroy(Category $category)
{
    try {
        $category->delete();
    } catch (Throwable $error) {
        report($error);
        return redirect()->route('categories.index')->with('status', 'Categorie is niet leeg.
        Er mogen geen producten meer in de categorie staan.');
    }
    return redirect()->route('categories.index')->with('status', 'Categorie verwijderd');
}
```

Wat we doen is de delete met een try - catch uitvoeren. Als het dan niet lukt, laten we de index weer zien met een melding dat de categorie niet leeg is.

Hiervoor is de class Throwable nodig. Let even op dat deze ook bovenin bij de use moet komen. We gebruiken ook een report functie zodat het in de log goed komt te staan.

```
namespace App\Http\Controllers\Admin;

use App\Http\Controllers\Controller;
use App\Http\Requests\CategoryStoreRequest;
use App\Http\Requests\CategoryUpdateRequest;
use App\Models\Category;
use Illuminate\Http\Request;
use Throwable;

class CategoryController extends Controller
{
```

Als we het nu weer proberen zien we een melding dat het niet gelukt is.

## Category Admin

Categorie is niet leeg. Er mogen geen producten meer in de categorie staan.

NAME	DETAILS	EDIT	DELETE
Prof. Keyon Howe	Details	Edit	Delete
Stephania Hauck	Details	Edit	Delete

De is nog wel in het groen, terwijl groen lijkt alsof het is gelukt. Hiervoor zou je dan nog een stukje kunnen maken in de lay-out, zodat dit niet groen is

In de index.blade van categories gaan we dus nog een wijziging moeten maken. We zorgen ervoor dat we een rode kleur hebben voor session('status-wrong')

```
@if(session('status'))  
    <div class="bg-green-200 text-green-900 rounded-lg shadow-md p-6 pr-10 mb-8"  
        style="...">  
        {{ session('status') }}  
    </div>  
@endif  
@if(session('status-wrong'))  
    <div class="bg-red-200 text-red-900 rounded-lg shadow-md p-6 pr-10 mb-8"  
        style="...">  
        {{ session('status-wrong') }}  
    </div>  
@endif
```

We updaten de delete methode. Let op dat alleen als het fout gaat je dan wrong gebruikt.

```
/** Remove the specified resource from storage. ...*/  
public function destroy(Category $category)  
{  
    try {  
        $category->delete();  
    } catch (Throwable $error)  
    {  
        report($error);  
        return redirect()->route('categories.index')->with('status-wrong', 'Categorie is niet leeg.  
        Er mogen geen producten meer in de categorie staan.');//  
    }  
    return redirect()->route('categories.index')->with('status', 'Categorie verwijderd');//  
}
```

Bij het testen zien we dan ook netjes de melding in de goede kleur staan.

## Category Admin

Categorie is niet leeg. Er mogen geen producten meer in de categorie staan.

NAME	DETAILS	EDIT	DELETE
Prof. Keyon Howe	Details	Edit	Delete
Stephania Hauck	Details	Edit	Delete

Alles werkt nu weer zoals het de bedoeling is. Tot zover de basis met Laravel8 en blade.

## 6. Unit & Feature testing

### 6.1. Omgeving

Nu we klaar zijn met de product crud gaan we aan de slag met testen. Je hebt gezien dat vanwege de product relatie de categorie delete niet meer werkend was. Om ervoor te zorgen dat je niet steeds handmatig alles moet gaan testen kunnen we gaan kijken hoe het gaat met geautomatiseerd testen.

Hiervoor gaan we goed kijken hoe we om kunnen gaan met unit en feature testen binnen Laravel. In Laravel zit standaard het testen er al in, met phpUnit. Op zich zou dit al voldoende zijn, maar we gaan het ons nog iets makkelijker maken met behulp van het test framework Pest.

The screenshot shows the official website for Pest. At the top, there's a dark header bar with a lock icon and the URL 'pestphp.com'. To the right of the URL are several small, colorful icons. Below the header is the Pest logo, which consists of the word 'PEST' in a bold, sans-serif font where each letter has a different color gradient: P is pink, E is teal, S is yellow, and T is blue. The main title 'An elegant PHP Testing Framework' is displayed in large, bold, black and green letters respectively. Below the title is a subtitle in a smaller, gray font: 'Pest is a Testing Framework with a focus on simplicity. It was carefully crafted to bring the joy of testing to PHP.' At the bottom of the screenshot are two buttons: a pink 'Get started' button and a purple 'Source Code' button.

Voor Pest gaan we de installatie doen zoals op de site staat. (<https://pestphp.com/docs/installation/>) De minimale php versie voor Pest is 7.3. Zelf gebruik ik wamp.net en zit op versie 8.03, dus voldoe ik aan de eis. Sowieso is 7.3 al een eis voor Laravel, dus als dat draait voldoe je al.

1. First, install Pest via the [Composer](#) package manager:

```
composer require pestphp/pest --dev --with-all-dependencies
```

We gaan dus door middel van composer Pest installeren in ons project.

Zo te zien wordt er wel wat aangepast, 5 installs en 5 updates

```
D:\Wamp\sites\laravelv1>composer require pestphp/pest --dev --with-all-dependencies
Using version ^1.0 for pestphp/pest
./composer.json has been updated
Running composer update pestphp/pest --with-all-dependencies
Loading composer repositories with package information
Updating dependencies
Lock file operations: 5 installs, 5 updates, 0 removals
- Upgrading filp/whoops (2.11.0 => 2.12.0)
- Upgrading nunomaduro/collision (v5.3.0 => v5.4.0)
- Locking pestphp/pest (v1.0.5)
- Locking pestphp/pest-plugin (v1.0.0)
- Locking pestphp/pest-plugin-coverage (v1.0.0)
- Locking pestphp/pest-plugin-expectations (v1.0.1)
- Locking pestphp/pest-plugin-init (v1.0.0)
- Upgrading phpunit/php-code-coverage (9.2.5 => 9.2.6)
- Upgrading symfony/console (v5.2.5 => v5.2.6)
- Upgrading symfony/string (v5.2.4 => v5.2.6)
```

2. On Laravel, require the `pest-plugin-laravel` and run the `pest:install` Artisan command:

```
composer require pestphp/pest-plugin-laravel --dev
php artisan pest:install
```

Er is een pest plugin voor Laravel. Het lijkt of deze al geïnstalleerd is als je goed hierboven kijkt. Toch zal er nog wat geïnstalleerd worden, namelijk het stuk specifiek voor Laravel.

```
D:\Wamp\sites\laravelv1>composer require pestphp/pest-plugin-laravel --dev
Using version ^1.0 for pestphp/pest-plugin-laravel
./composer.json has been updated
Running composer update pestphp/pest-plugin-laravel
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
- Locking pestphp/pest-plugin-laravel (v1.0.0)

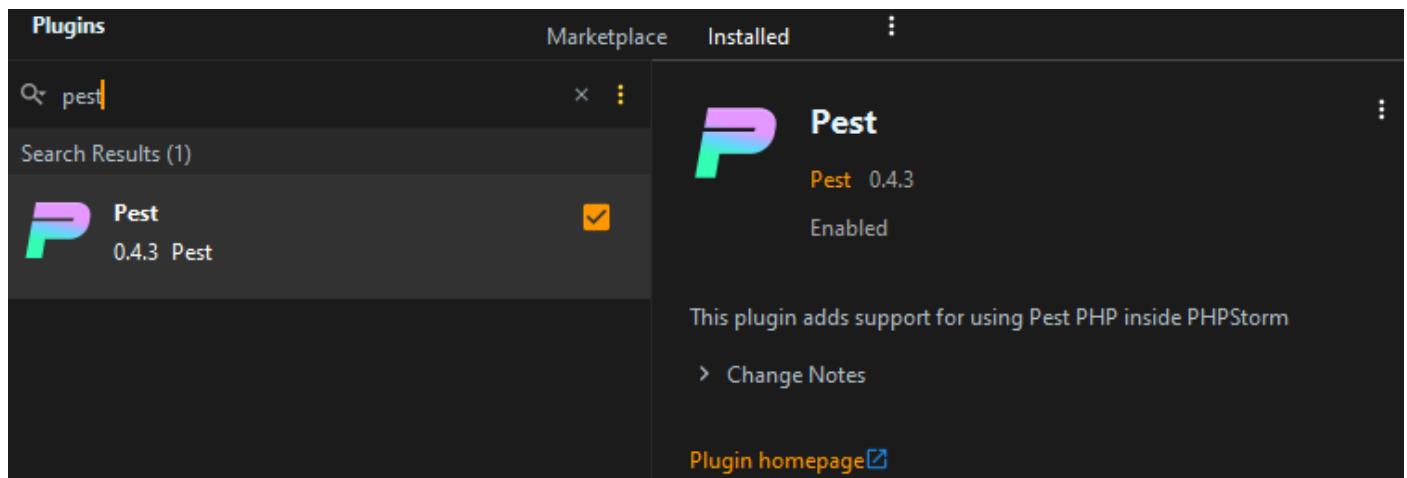
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
- Downloading pestphp/pest-plugin-laravel (v1.0.0)
- Installing pestphp/pest-plugin-laravel (v1.0.0): Extracting archive
```

De volgende stap is, met php artisan pest installeren.

```
D:\Wamp\sites\laravelv1>php artisan pest:install
```

```
[OK] `tests/Pest.php` created successfully.
```

Het kan zijn dat het bij mij al er anders uitziet. Er is voor PhpStorm een plugin voor Pest. Deze heb ik dan ook geïnstalleerd.



## 6.2. Gebruik van Pest / Phpununit

We kunnen nu testen met Pest. Let er even op dat in onze terminal we een backslash hebben en niet een slash. Als je het commando verkeerd doet krijg je dit:

```
D:\Wamp\sites\laravelv1>./vendor/bin/pest
'.' is not recognized as an internal or external command,
operable program or batch file.
```

Voer je het de test uit met de \ , dan werkt het.

En ja, we krijgen meteen een error want we hebben al allerlei wijzigingen in een standaard project gedaan waardoor de standaard aangeleverde test meteen een fout geeft.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest

  PASS Tests\Unit\ExampleTest
    ✓ basic test

  PASS Tests\Feature\AuthenticationTest
    ✓ login screen can be rendered
    ✓ users can authenticate using the login screen
    ✓ users can not authenticate with invalid password

  PASS Tests\Feature\EmailVerificationTest
    ✓ email verification screen can be rendered
    ✓ email can be verified
    ✓ email is not verified with invalid hash

  FAIL Tests\Feature\ExampleTest
    ✗ basic test

  PASS Tests\Feature\PasswordConfirmationTest
    ✓ confirm password screen can be rendered
    ✓ password can be confirmed
    ✓ password is not confirmed with invalid password

  PASS Tests\Feature>PasswordResetTest
    ✓ reset password link screen can be rendered
    ✓ reset password link can be requested
    ✓ reset password screen can be rendered
    ✓ password can be reset with valid token

  PASS Tests\Feature\RegistrationTest
    ✓ registration screen can be rendered
    ✓ new users can register
```

We krijgen ook meteen mee wat er dan precies fout is:

```
• Tests\Feature\ExampleTest > basic test
Expected status code 200 but received 302.
Failed asserting that 200 is identical to 302.

at D:\Wamp\sites\laravelv1\tests\Feature\ExampleTest.php:19
15 |     public function testBasicTest()
16 |     {
17 |         $response = $this->get('/');
18 |
→ 19 |         $response->assertStatus(200);
20 |     }
21 |
22 |

Tests: 1 failed, 16 passed
Time: 11.68s
```

De status 200 betekent dat alles goed gaat. Maar op onze pagina wordt je meteen naar de login pagina gestuurd. Dit zorgt ervoor dat we de status 200 niet meer krijgen, maar de 302. Nu gebruikt Pest ook gewoon phpunit, maar zorgt dat het zo meteen wat makkelijker wordt en tevens ook duidelijker wat er fout gaat. Voor onze fout nu, kunnen we ook is phpunit zelf gebruiken, zonder Pest.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\phpunit
PHPUnit 9.5.4 by Sebastian Bergmann and contributors.

.....F.....
17 / 17 (100%)

Time: 00:01.602, Memory: 42.00 MB

There was 1 failure:

1) Tests\Feature\ExampleTest::testBasicTest
Expected status code 200 but received 302.
Failed asserting that 200 is identical to 302.

D:\Wamp\sites\laravelv1\vendor\laravel\framework\src\Illuminate\Testing\TestResponse.php:187
D:\Wamp\sites\laravelv1\tests\Feature\ExampleTest.php:19

FAILURES!
Tests: 17, Assertions: 27, Failures: 1.
```

Als je dit vergelijkt met de uitkomst eerder, ziet het er wat anders uit. Je zal merken dat het steeds makkelijker zal worden.

### 6.3. Response Status

Bij de ExampleTest hebben we nu voor het eerst iets gezien van een status bij testen. De status is erg van belang. Om ervoor te zorgen dat je makkelijker straks met de testen om te gaan, is het van belang dat je weet wat voor status je kan verwachten, zodat je daarop kan testen. Als je naar http status gaat zoeken, kom je bijvoorbeeld op de documentatie van mozilla. (<https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>)

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

1. Informational responses (100–199),
2. Successful responses (200–299),
3. Redirects (300–399),
4. Client errors (400–499),
5. and Server errors (500–599).

The below status codes are defined by section 10 of RFC 2616. You can find an updated specification in RFC 7231.

Er zijn er heel wat, maar met een aantal gaan we veel te maken krijgen:

#### **200 OK**

de data die opgevraagd is van de webserver is naar de client gestuurd.

#### **301 Moved Permanently**

de data kan niet opgevraagd worden onder het huidige adres.

#### **302 Found**

de data kan niet opgevraagd worden onder het huidige adres op dit moment. Je kan later wel weer terecht op dit adres voor de data

#### **400 Bad Request**

de http request heeft een verkeerde syntax, bijvoorbeeld dat de url verkeerde parameters heeft.

#### **401 Not authorized**

Niet ingelogd

#### **403 Forbidden**

Wel ingelogd, maar geen permissie om hier te komen

#### **404 Not Found**

pagina is niet gevonden

#### **422 Unprocessable Entity**

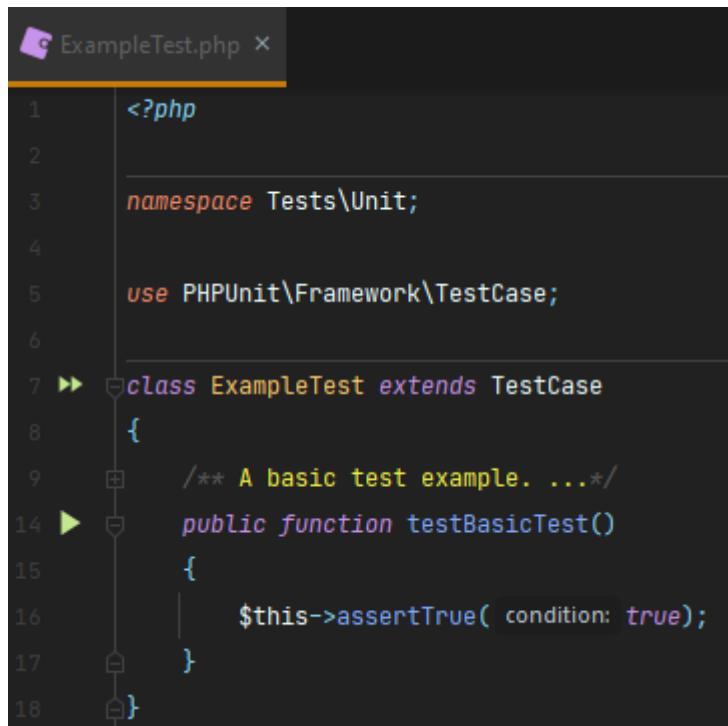
validatie is niet gelukt

#### **500 Internal server error**

Algemene server error

## 6.4. Schrijven van een test

Het schrijven van een test wordt simpel door middel van Pest, alleen de voorbeeld testen die standaard met Laravel worden meegeleverd zijn niet op basis van Pest. Deze gaan we als eerst even herschrijven met het gebruik van Pest. De eerste test waar we naar gaan kijken staat in `tests/Unit/ExampleTest.php`



```
<?php

namespace Tests\Unit;

use PHPUnit\Framework\TestCase;

class ExampleTest extends TestCase
{
    /**
     * A basic test example. ...
     */
    public function testBasicTest()
    {
        $this->assertTrue( condition: true);
    }
}
```

Op zich een simpele test. Je ziet de namespace dat we in Unittesten zitten. We erven de TestCase class over, waar de basis instaat en we hebben een testBasicTest() methode als 1<sup>e</sup> test. Met Pest hoeven we dit allemaal niet meer in de test zelf te schrijven, waardoor een test veel simpeler wordt. Met `php artisan pest:test Example2Test --unit` maken we dan een nieuwe test aan.

```
D:\Wamp\sites\laravelv1>php artisan pest:test Example2Test --unit
```

```
[OK] `tests/Unit/Example2Test.php` created successfully.
```

We hebben nu exact dezelfde test als bij phpunit. Zie het verschil



```
<?php

test( description: 'example2', function () {
    $this->assertTrue( condition: true);
});
```

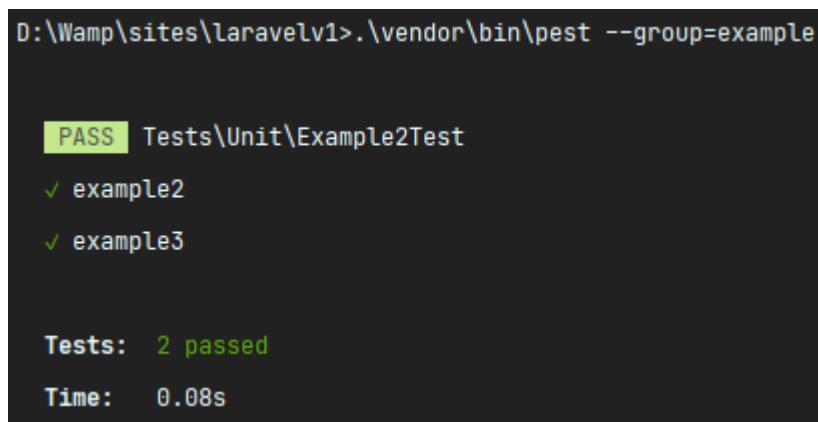
We gebruiken de functie test, waarin we de naam van de test angeven. Verder voeren we dezelfde test uit. En dit is alles wat we nodig hebben.

En nog steeds kan het korter worden geschreven, door middel van een hogere order test. Als we geen afsluiting gebruiken kunnen we een chain achter de eerste methode gebruiken.



```
ExampleTest.php × Example2Test.php ×
1 > <?php
2
3 ▶ test( description: 'example2', function () {
4     $this->assertTrue( condition: true);
5 })->group( ...groups: 'example');
6
7 ▶ test( description: 'example3')->assertTrue( condition: true)->group('example');
```

Example2 en example3 doen precies hetzelfde, maar wel weer minder op te schrijven. Om ervoor te zorgen dat we alleen even deze testen uit kunnen voeren, kan je ze in een group zetten. Hierdoor kan je makkelijker alleen de tests uit voeren die je wilt. Dit kan dan door achter het commando `--group=groepnaam` te zetten.



```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=example
PASS Tests\Unit\Example2Test
✓ example2
✓ example3

Tests: 2 passed
Time: 0.08s
```

Je ziet dan ook dat alleen deze uitgevoerd zijn, waarbij we eerst al 17 testen hadden.

Als we de condition nu is false maken, zie je ook dat het resultaat aangeeft wat er dan precies fout is.

```
test( description: 'example3')->assertTrue( condition: false)->group('example');
```

Het resultaat van de test is dan dit:

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=example

 FAIL Tests\Unit\Example2Test
✓ example2
✗ example3

---

• Tests\Unit\Example2Test > example3
Failed asserting that false is true.

at D:\Wamp\sites\laravelv1\tests\Unit\Example2Test.php:7
 3 | test('example2', function () {
 4 |     $this->assertTrue(true);
 5 | })->group('example');
 6 |
→ 7 | test('example3')->assertTrue(false)->group('example');
 8 |

 1 [internal]:0
    PHPUnit\Framework\Assert::assertTrue()

Tests: 1 failed, 1 passed
Time:  0.09s
```

Je ziet bij de fout meteen wat er niet goed is, zoals nu dat false ongelijk aan true is. Verder zie je ook de code van de test even, zodat je exact weet wat er getest wordt. De pijl bij regel 7 geeft aan dat het bij dat onderdeel fout gaat. Om ervoor te zorgen dat we niet steeds de foutmelding krijgen bij deze test, zetten we de conditie weer op true.

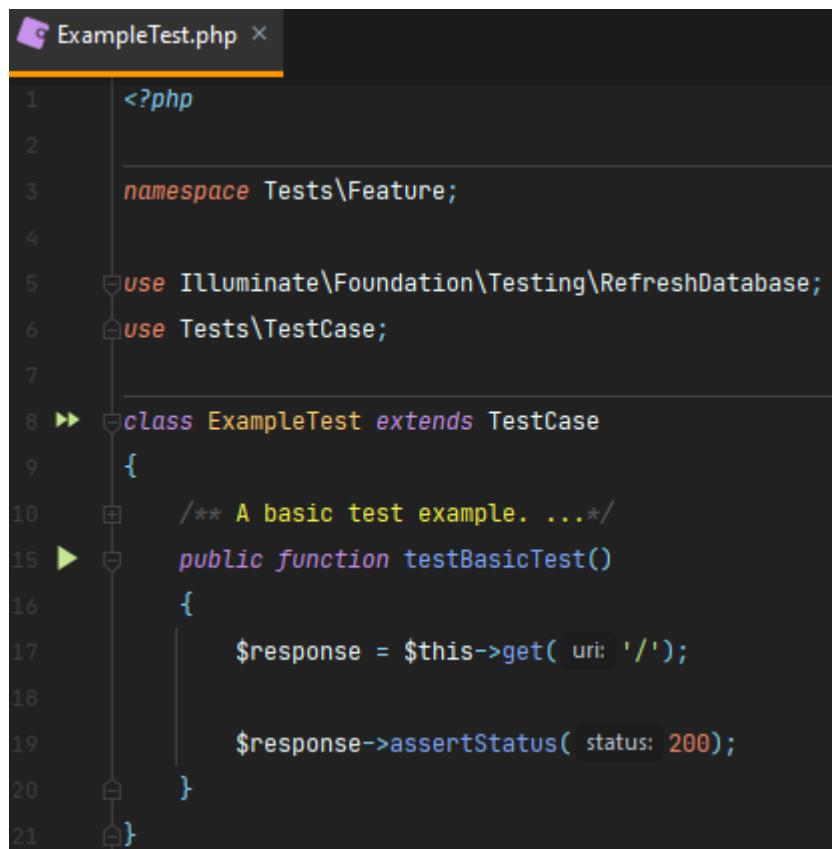
```
test( description: 'example3')->assertTrue( condition: true)->group('example');
```

## 6.5. Feature test

Nu we de unittest een klein beetje hebben bekeken, gaan we kijken hoe het werkt met een feature test. Maar wat is nu precies het verschil tussen een unit test en een feature test. Eigenlijk is het heel simpel:

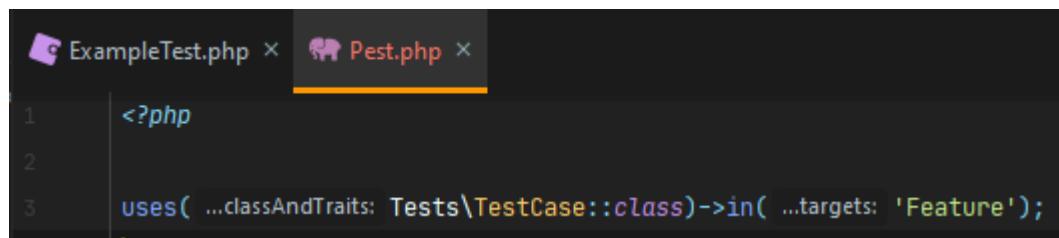
- Bij een feature test, test je de applicatie zoals een echte gebruiker het zou uitvoeren. Dus op linkjes klikken, formulier invullen etc. Het gaat er dus om wat een gebruiker kan zien.
- Bij een unit test ga je eigenlijk 1 bepaalde functie testen of dit werkt. Het gaat dan ook om de logica van een onderdeel, ook wel unit genoemd.

Als we gaan kijken naar ons voorbeeld in Feature tests, zien we dit:



```
ExampleTest.php
1 <?php
2
3 namespace Tests\Feature;
4
5 use Illuminate\Foundation\Testing\RefreshDatabase;
6 use Tests\TestCase;
7
8 class ExampleTest extends TestCase
9 {
10     /**
11      * A basic test example. ...
12     */
13    public function testBasicTest()
14    {
15        $response = $this->get('/');
16
17        $response->assertStatus(status: 200);
18    }
19
20 }
21
```

Voor een feature test hebben we wel alles nodig van Laravel. Eigenlijk heb je een boot nodig van Laravel. Gelukkig is Pest zo gemaakt dat dit niet meer hoeft. In Pest.php staat dat we de class TestCase altijd moeten gebruiken.



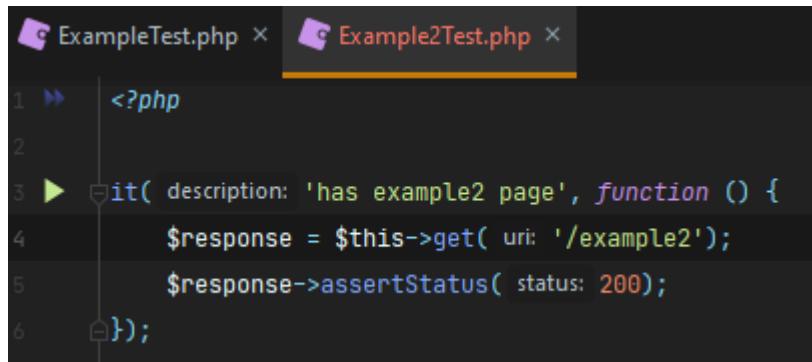
```
Pest.php
1 <?php
2
3 uses( ...classAndTraits: Tests\TestCase::class)->in( ...targets: 'Feature');
```

Als we een feature test willen aanmaken gebruiken we `php artisan pest:test Example2Test`.

```
D:\Wamp\sites\laravelv1>php artisan pest:test Example2Test

[OK] `tests/Feature/Example2Test.php` created successfully.
```

We hebben nu een nieuwe feature test met de naam Example2Test. Hier staat al het voorbeeld in hoe je een feature test kan schrijven.



```
ExampleTest.php x Example2Test.php x

1 >>> <?php
2
3 >>> it( description: 'has example2 page', function () {
4     $response = $this->get( uri: '/example2');
5     $response->assertStatus( status: 200);
6 });

```

We kunnen dus de gehele test omschrijven voor wat bij Pest nodig is. Dit kan dan zo.

```
it( description: 'has welcome page', function () {
    $response = $this->get( uri: '/');
    $response->assertStatus( status: 200);
});
```

En ja, tuurlijk kan je hier ook de chain gebruiken

```
it( description: 'has welcome page', function () {
    $response = $this->get( uri: '/');
    $response->assertStatus( status: 200);
})->group( ...groups: 'exampleFeature');

it( description: 'has welcome2 page')
    :>get( uri: '/')
    ->assertStatus( status: 200)->group('exampleFeature');
```

Je ziet hier wel dat ik geen test() gebruik maar it(). Ze doen precies hetzelfde, alleen staat er bij it() nog in de tekst it erbij. Ook nu weer even in een group gezet, zodat we alleen deze kunnen testen.

Als we deze test dan uitvoeren zie je dit:

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=exampleFeature

FAIL Tests\Feature\Example2Test
✖ it has welcome page
✖ it has welcome2 page
```

Beide tests komen er niet doorheen, wat logisch is want er is nog steeds een redirect naar de login voor als je niet bent ingelogd. Zouden we testen op de status 302 zie je dat de test wel er doorheen komt.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=exampleFeature

PASS Tests\Feature\Example2Test
✓ it has welcome page
✓ it has welcome2 page

Tests: 2 passed
Time: 0.24s
```

Die statussen zijn soms even wennen. Bij status 302 stond namelijk dit:

### 302 Found

de data kan niet opgevraagd worden onder het huidige adres op dit moment. Je kan later wel weer terecht op dit adres voor de data

Het klopt ook wel, want je komt niet bij het adres. Zouden we de redirect niet in onze code hebben maar bijv alleen een foutmelding dat we niet zijn ingelogd, kunnen we weer een andere status krijgen.

Bij de feature test heb je nu wel gezien, dat we op een bepaalde pagina komen en daar wat testen. Je zal later zien wat voor soort tests hier allemaal kunnen.

## 6.6. Opdracht 21: Schrijven van een Featuretest

Maak in de web.php een route aan die rechtstreeks naar de view About gaat. De view van de about moet in de volgende map zitten en ook de correcte naam hebben:

- Resources/views/public/about.blade.php

De view moet vanuit de route /about naar de pagina worden gestuurd.

Schrijf een feature test die controleert:

- De pagina wordt vanuit /about ingeladen
- De view staat in resources/views/public/about.blade.php
- Response status = 200
- Testgroup = PublicViews

## 6.7. Dataset

Nu we toch nog even bezig zijn met de example, kunnen we even wat andere onderdelen van testen bekijken. Als eerst gaan we aan de slag met een dataset. Stel je voor dat we een aantal email adressen willen controleren.

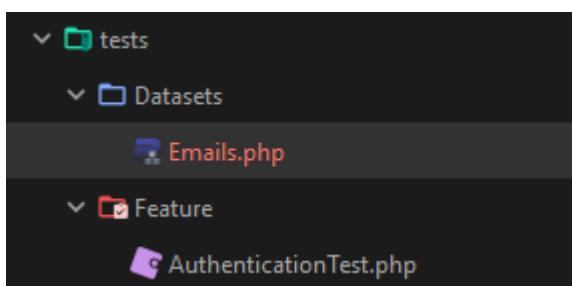
Als je een dataset eenmaal wilt gebruiken kan je deze inline meegeven.

```
it( description: 'has only emails', function ($email) {
    expect($email)->not->toBeEmpty();
})->with( ['admin@tcrmbo.nl', 'sales@tcrmbo.nl']);
```

Nu is dit eigenlijk ook nog niet netjes. Stel je voor dat ik hier een array van 15 mailadressen wil checken. De code wordt dan weer niet ok. Hiervoor kunnen we met Pest een dataset aanmaken met `php artisan pest:dataset emails`

```
D:\Wamp\sites\laravelv1>php artisan pest:dataset Emails
[OK] `tests/Datasets/Emails.php` created successfully.
```

Er wordt een map Datasets aangemaakt in de tests map, met daarin een bestand Emails.php



In Emails.php staat nu al een dataset voorbeeld

```
Emails.php x
1  <?php
2
3  dataset( name: 'emails', function () {
4      return ['email A', 'email B'];
5  });
6
```

Dit kunnen we aanpassen naar bijvoorbeeld emails die we ook bij gebruikers hebben:

```
dataset( name: 'emails', function () {
    return ['admin@tcrmbo.nl', 'sales@tcrmbo.nl', 'customer@tcrmbo.nl'];
});
```

Nu kunnen we deze dataset in onze test gebruiken

```
it( description: 'has emails', function ($email) {
    expect($email)->not->toBeEmpty();
})->with( data: 'emails');
```

Deze manier gebruik je als je de tests schoon wil houden en dus datasets apart in een bestand zet. Je kan zo'n dataset bij meerdere tests gebruiken. Je kan een uitgebreide array hebben als test, met bijvoorbeeld name en email.

```
▶ it( description: 'has emails', function ($email) {
    expect($email)->not->toBeEmpty();
})->with( data: 'emails');

▶ it( description: 'has only emails', function ($email) {
    expect($email)->not->toBeEmpty();
})->with( ['admin@tcrmbo.nl', 'sales@tcrmbo.nl']);

▶ it( description: 'has emails with name', function ($name, $email) {
    expect($email)->not->toBeEmpty();
})->with([
    ['admin', 'admin@tcrmbo.nl'],
    ['sales', 'sales@tcrmbo.nl']
]);
```

Als we deze tests ook nog even in de groep 'exampleFeature' zetten, kunnen we deze feature tests makkelijk apart testen. Je ziet alle tests dan verschijnen. Misschien merk je wel al dat de omschrijving van de test erg van belang is om te weten wat je precies test.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=exampleFeature

PASS Tests\Feature\Example2Test
✓ it has welcome page
✓ it has welcome2 page
✓ it has emails with ('admin@tcrmbo.nl')
✓ it has emails with ('sales@tcrmbo.nl')
✓ it has emails with ('customer@tcrmbo.nl')
✓ it has only emails with ('admin@tcrmbo.nl')
✓ it has only emails with ('sales@tcrmbo.nl')
✓ it has emails with name with ('admin', 'admin@tcrmbo.nl')
✓ it has emails with name with ('sales', 'sales@tcrmbo.nl')

Tests: 9 passed
Time: 0.58s
```

## 6.8. Gebruik van een database

Binnen het testen kan je natuurlijk ook gebruik maken van een database. Let wel even op, dat dit normaal gesproken een aparte database is voor alleen de testen. Voordat we de testdatabase kunnen gebruiken moeten we wat opties overwegen. Standaard is de testdatabase in phpunit een sqlite database. Dan moet je wel sqlite installeren. Dit is een optie, maar omdat we al zo gewend zijn aan een mysql database kies ik hiervoor. Verder, als je verder gaat zoeken zal je dit soort dingen vinden:

However, it's not all flowers and sunshine. There were some issues **where certain SQL differs between our testing and production databases.**

That's not good because the test becomes a false positive. If you don't have good coverage during QA, it might be that you will only discover these nuances when your code fails in production.

Dus voor testen en productie dezelfde type database gebruiken. De database die we nu gebruiken staan gewoon in onze .env bestand:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravelv1
DB_USERNAME=wamp
DB_PASSWORD=
```

Bij wamp.net wordt er een gebruiker voor de database aangemaakt genaamd wamp, zonder wachtwoord. De user root is ook nog beschikbaar, die ook geen wachtwoord heeft.

Hier is dus van belang dat we de verbinding mysql gebruiken en de database workshop. Om voor onze test een andere database te gebruiken, kunnen we de testconfiguratie in de *phpunit.xml* gaan neerzetten, die je kan vinden in de root map van je project. Je ziet daar nu in:

```
<php>
    <server name="APP_ENV" value="testing"/>
    <server name="BCRYPT_ROUNDS" value="4"/>
    <server name="CACHE_DRIVER" value="array"/>
    <!-- <server name="DB_CONNECTION" value="sqlite"/> -->
    <!-- <server name="DB_DATABASE" value=":memory:"/> -->
    <server name="MAIL_MAILER" value="array"/>
    <server name="QUEUE_CONNECTION" value="sync"/>
    <server name="SESSION_DRIVER" value="array"/>
    <server name="TELESCOPE_ENABLED" value="false"/>
</php>
</phpunit>
```

Hier staat als commentaar een sqlite database. Deze gaan we niet gebruiken, maar we maken gewoon een tweede mysql database aan.

Dit wordt dan:

```
20 <php>
21     <server name="APP_ENV" value="testing"/>
22     <server name="BCRYPT_ROUNDS" value="4"/>
23     <server name="CACHE_DRIVER" value="array"/>
24     <!-- <server name="DB_CONNECTION" value="sqlite"/> -->
25     <!-- <server name="DB_DATABASE" value=":memory:"/> -->
26     <server name="DB_CONNECTION" value="mysql"/>
27     <server name="DB_DATABASE" value="laraveltest"/>
28     <server name="MAIL_MAILER" value="array"/>
29     <server name="QUEUE_CONNECTION" value="sync"/>
30     <server name="SESSION_DRIVER" value="array"/>
31     <server name="TELESCOPE_ENABLED" value="false"/>
32 </php>
```

Dus in phpunit.xml zorgen we ervoor dat DB\_CONNECTION de waarde van mysql krijgt, zodat we dezelfde verbinding gebruiken. Daarna geven we bij DB\_DATABASE aan welke database we willen gebruiken. Hiervoor ga ik de laraveltest database gebruiken. Binnen phpMyAdmin kunnen we dan een database aanmaken. Deze gaan we dan *testworkshop* noemen. (vergeet niet op aanmaken te klikken)

The screenshot shows the phpMyAdmin dashboard with the server set to 127.0.0.1. The 'Databases' tab is selected. A modal window is open for creating a new database. In the input field, 'laraveltest' is typed. Next to it, the character set dropdown is set to 'latin1\_swedish\_ci'. At the bottom right of the modal, there is a grey button labeled 'Aanmaken' (Create).

In mysql hebben we dan 2 databases, namelijk laravelv1 en laraveltest. Omdat er met testen naar de phpunit.xml wordt gekeken, zullen we netjes de testworkshop database gaan gebruiken.

Nu zijn we bijna klaar om de methode binnen onze test te gaan schrijven. Het enige wat we nu nog missen is dat de functie niet zomaar weet dat de database gebruikt moet worden. Omdat we dit standaard bij testen willen gaan gebruiken, gaan we dit in Pest.php zetten. Pest.php kan je vinden in de root map van tests. In Pest.php staat nu dit:

```
Pest.php ×
1 <?php
2
3 uses( ...classAndTraits: Tests\TestCase::class)->in( ...targets: 'Feature');
```

We gaan hier zorgen dat we de RefreshDatabase gaan gebruiken, wat bij testen normaal is. Let op, we gebruiken een class, dus moeten we ook even

```
1 <?php
2
3 use \Illuminate\Foundation\Testing\RefreshDatabase;
4 /*...*/
14
15 uses( ...classAndTraits: Tests\TestCase::class, RefreshDatabase::class)->in( ...targets: 'Feature');
16
```

Bij de featuretest gaan we een nieuwe functie erbij maken waarbij je bijvoorbeeld een user in een database zet.

```
it( description: 'has users', function(){
    User::factory()->create();
    $this->assertDatabaseHas('users', ['id' => 1]);
})->group( ...groups: 'dbtest');
```

In deze test maken we gebruik van de standaard factory die er al is voor users. Hier maken we dus 1 gebruiker aan. Daarna controleren we in de database of er een user is met id = 1. Let wel op, we gebruiken de model User nu, dus bovenin de test zal je deze moeten toevoegen.

```
1 <?php
2
3 use App\Models\User;
```

Als we de test nu gaan uitvoeren zal je merken dat de test goed verloopt. Voor dit voorbeeld even de groep dbtest gemaakt, zodat de test weer apart te doen is.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=dbtest

  PASS  Tests\Feature\Example2Test
  ✓ it has users

  Tests: 1 passed
  Time:  0.72s
```

Wat misschien wel een beetje vaag is, is als je naar de laraveltest database gaat zie je helemaal niks staan. Dit komt omdat er bij de test wat wordt aangemaakt, maar na de test gaat het meteen weer weg.

Wat je nog wel ziet zijn alle tabellen, terwijl we geen commando hebben gegeven om dit te doen. Dit is automatisch gebeurd.

The screenshot shows the phpMyAdmin interface for the 'laraveltest' database. At the top, there are tabs for 'Structuur', 'SQL', 'Zoeken', and 'Query'. Below the tabs, there is a 'Filters' section with a search input field labeled 'Bevat het woord:' followed by a text input box. The main area displays a table of 12 tables:

Tabel	Actie
categories	<a href="#">Verkennen</a>
failed_jobs	<a href="#">Verkennen</a>
migrations	<a href="#">Verkennen</a>
model_has_permissions	<a href="#">Verkennen</a>
model_has_roles	<a href="#">Verkennen</a>
password_resets	<a href="#">Verkennen</a>
permissions	<a href="#">Verkennen</a>
prices	<a href="#">Verkennen</a>
products	<a href="#">Verkennen</a>
roles	<a href="#">Verkennen</a>
role_has_permissions	<a href="#">Verkennen</a>
users	<a href="#">Verkennen</a>

Below the table, the text '12 tabellen' is displayed. To the right of the table, the word 'Som' is written.

Tijdens de test moet er echt een gebruiker zijn geweest anders was de test er niet doorheen gekomen.

## 6.9. Naamgeving en Setup

In een test kan je maar 1 functie met dezelfde naam hebben natuurlijk. Om dit even te testen gaan we de laatste test 2x uitvoeren.

```
it( description: 'has users', function(){
    User::factory()->create();
    $this->assertDatabaseHas('users', ['id' => 1]);
})->group( ...groups: 'dbtest');

it( description: 'has users', function(){
    User::factory()->create();
    $this->assertDatabaseHas('users', ['id' => 2]);
})->group( ...groups: 'dbtest');
```

Als we dat de test uitvoeren krijgen we ook netjes een melding

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=dbtest

Pest\Exceptions\TestAlreadyExist

A test with the description `it has users` already exist in the filename `D:\Wamp\sites\laravelv1\tests\Feature\Example2Test.php`.
```

Hierdoor gaan we nu eerst de naamgeving van de test veranderen. Natuurlijk kunnen we nu ook voor user met id = 2 testen.

```
it( description: 'has users', function(){
    User::factory()->create();
    $this->assertDatabaseHas('users', ['id' => 1]);
})->group( ...groups: 'dbtest');

it( description: 'has users 2', function(){
    User::factory()->create();
    $this->assertDatabaseHas('users', ['id' => 2]);
})->group( ...groups: 'dbtest');
```

Als we nu de test uitvoeren zie je dat nu alles is zoals we het verwachten.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=dbtest

PASS Tests\Feature\Example2Test
✓ it has users
✓ it has users 2

Tests: 2 passed
Time: 0.81s
```

Wel kunnen we de testen nog optimaliseren. Ik gebruik nog even geen chain, maar we kunnen met beforeEach zorgen dat we bij elke test bijvoorbeeld een user aanmaken.

```
beforeEach(function () {
    User::factory()->create();
});

it( description: 'has users', function(){
    $this->assertDatabaseHas('users', ['id' => 1]);
})->group( ...groups: 'dbtest');

it( description: 'has users 2', function(){
    $this->assertDatabaseHas('users', ['id' => 2]);
})->group( ...groups: 'dbtest');
```

Het resultaat is nu goed. We krijgen netjes de users die we verwachten.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=dbtest

PASS Tests\Feature\Example2Test
✓ it has users
✓ it has users 2

Tests: 2 passed
Time: 0.79s
```

Let wel op, beforeEach zorgt dat bij elke test er een user wordt aangemaakt. Heb je meerdere testen in je bestand staan, gaat die voor elke test gewoon een gebruiker aanmaken. Dit kan ervoor zorgen dat de id die je wilt testen er niet meer is.

## 7. Product Tests

Nu de basis van testen een beetje bekend zijn, kunnen we wat testen schrijven voor het project. Omdat de products klaar zijn wat betreft functionaliteit gaan we daar eerst is naar kijken.

Het eerste wat we doen, is alles in de ExampleTest in de map Unit en Feature als commentaar zetten. Dit zodat we nog wel de voorbeeld code hebben maar dat ze niet meegenomen worden in de testen.

```
<?php
//test('basic')->assertTrue(true);
```

```
<?php
/*
beforeEach(fn () => factory(App\User::class)->create());

it('has users', function(){
    $this->assertDatabaseHas('users', ['id' => 1]);
});

it('has users 2', function(){
    $this->assertDatabaseHas('users', ['id' => 2]);
});
*/
```

## 7.1. Product Index

Als we naar products gaan kijken, gaan we gewoon aan de slag met de 1<sup>e</sup> methode waar we terecht zouden komen. We maken netjes een test hiervoor aan. Dit doen we met php artisan, want voor pest is dat beschikbaar. Omdat we straks veel meer testen gaan hebben is het van belang om dit te ordenen. Hierdoor gaan we voor products een map aanmaken. Dit kunnen we doen bij het aanmaken van de test.

```
D:\Wamp\sites\laravelv1>php artisan pest:test Products/ProductIndexTest
[OK] `tests/Feature/Products/ProductIndexTest.php` created successfully.
```

Als we naar producten gaan kijken hebben we altijd een product nodig. Maar een product hoort bij een categorie. Een product heeft tevens ook een prijs. Dus om ervoor te zorgen dat we dit bij alle testen kunnen gebruiken kunnen we dit al in een beforeEach zetten.

```
use App\Models\Category;
use App\Models\Product;
use App\Models\Price;

beforeEach(function () {
    $this->category = Category::factory()->create();
    $this->product = Product::factory()->create();
    $this->price = Price::factory()->create();
});
```

Let even op dat we dit moeten doen, omdat we een lege testdatabase standaard hebben. Nu kunnen we namelijk controleren bij elke methode of er wel een categorie, product en prijs zijn.

Wat we ook nodig hebben, is dat we ingelogd zijn als een bepaalde gebruiker. De product pagina's zijn allemaal beveiligd door de authenticatie met rollen. Hiervoor moeten we dus de RoleAndPermissionSeeder en UserSeeder uitvoeren om ervoor te zorgen dat die bestaan in de testdatabase.

```
beforeEach(function () {
    $this->seed( class: 'RoleAndPermissionSeeder');
    $this->seed( class: 'UserSeeder');

    $this->category = Category::factory()->create();
    $this->product = Product::factory()->create();
    $this->price = Price::factory()->create();
});
```

Nog wel even voor de zekerheid, de permissies die worden aangemaakt bij de test voor product zijn dus dit:

```
// permissions for product CRUD
Permission::create(['name' => 'index product']);
Permission::create(['name' => 'show product']);
Permission::create(['name' => 'create product']);
Permission::create(['name' => 'edit product']);
Permission::create(['name' => 'delete product']);

// customer role
$customer = Role::create(['name' => 'customer']);

// sales role
$sales = Role::create(['name' => 'sales'])
    ->givePermissionTo(['index category', 'show category', 'create category', 'edit category',
        'index product', 'show product', 'create product', 'edit product']);

// admin role
$admin = Role::create(['name' => 'admin'])
    ->givePermissionTo(Permission::all());
```

Customer heeft geen permissie, sales en admin wel, al kan sales niet bij de delete komen. In de UserSeeder worden de 3 gebruikers aangemaakt, waarbij customer id=1 krijgt, sales id=2 en admin id=3.

Nu kunnen we de test werkelijk gaan schrijven. Het eerste wat we gaan testen is of de admin bij de product index kan komen, en of die hierop ook een product ziet staan

```
> test( description: 'admin can see the product index page', function()
{
    $admin = User::find(3);
    Laravel\be($admin)
        ->get(route( name: 'products.index'))
        ->assertSee($this->product->name)
        ->assertSee($this->product->category->name)
        ->assertSee($this->product->latest_price->price);

})->group( ...groups: 'ProductIndex');
```

Nu zal je zien dat we de User model nodig hebben om de gebruiker te vinden. Testen kunnen we inloggen als admin met de methode be(). Deze methode zit in Pest, waardoor we dit ook nog moeten inladen.

```
use App\Models\User;
use App\Models\Category;
use App\Models\Product;
use App\Models\Price;
use \Pest\Laravel;
```

We hebben dus nu 4 models ingeladen en Laravel Pest methodes.

Je ziet dus :

- dat we eerst de admin opzoeken
- dan inloggen met deze user
- naar de route 'product index' gaan
- en kijken of daar de product name, category name en price op staat.

Het checken of er iets te vinden is doen we met assertSee(). Hiermee kunnen we kijken of er in het resultaat een bepaalde tekst staat. De test zit in de group ProductIndex, zodat we dit apart kunnen testen. Als we de test uitvoeren zien we dat het allemaal goed gaat.

```
D:\wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductIndex

PASS Tests\Feature\Products\ProductIndexTest
✓ admin can see the product index page

Tests: 1 passed
Time: 1.42s
```

Als je wilt uitproberen of het werkelijk goed is, kan je altijd user met id=1 ophalen (de customer), want die zou geen rechten moeten hebben. Je ziet dan bij de foutomschrijving meteen een heel stuk html staan, omdat we voor tekst kijken met assertSee.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductIndex

FAIL Tests\Feature\Products\ProductIndexTest
✖ admin can see the product index page

---

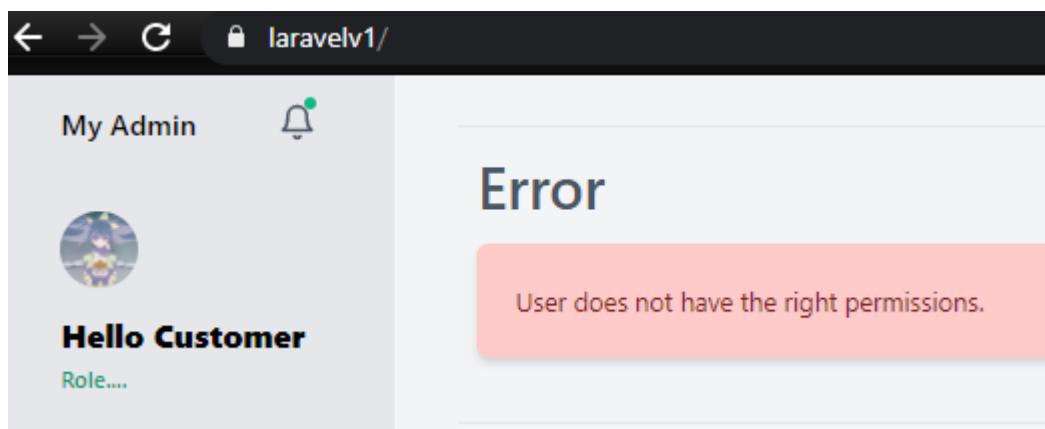
• Tests\Feature\Products\ProductIndexTest > admin can see the product index page
Failed asserting that '<!doctype html>\r\n<html lang="en">\r\n<head>\r\n</head>\r\n<body>\r\n</body>\r\n</html>' matches '...<head><meta charset="UTF-8"/><meta name="viewport" content="width=device-width, initial-scale=1.0"/><title>Product Index</title></head><body><h1>Product Index</h1><ul><li>Product 1</li><li>Product 2</li><li>Product 3</li></ul></body></html>'
```

Als we als admin het nu kunnen testen, zouden we ook als sales en customer moeten testen in de product index test. Voor sales, die de pagina ook gewoon kan zien is de test zo goed als gelijk. We halen id=2 op. Zelf vind ik het prettig om dat ook de user variabel de goede naam te geven.

```
test( description: 'sales can see the product index page', function()
{
    $sales = User::find(2);
    Laravel\be($sales)
        ->get(route( name: 'products.index'))
        ->assertSee($this->product->name)
        ->assertSee($this->product->category->name)
        ->assertSee($this->product->latest_price->price);

})->group( ...groups: 'ProductIndex');
```

De customer zou er niet bij mogen komen. Als we het nu proberen op de site, krijgen we een error pagina dat we niet de goede rol hebben.



Bij het testen kunnen we dus zeggen, dat je er niet bij mag komen als customer.

```
test( description: 'customer can not see the product index page', function()
{
    $customer = User::find(1);
    Laravel\be($customer)
        ->get(route( name: 'products.index'))
        ->assertForbidden();

})->group( ...groups: 'ProductIndex');
```

Als we nu de test doen krijgen we helaas een foutmelding

```
D:\wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductIndex

FAIL Tests\Feature\Products\ProductIndexTest
✓ admin can see the product index page
✗ sales can see the product index page
✗ customer can not see the product index page

---

• Tests\Feature\Products\ProductIndexTest > sales can see the product index page
TypeError

Pest\Laravel\be(): Argument #1 ($user) must be of type Illuminate\Contracts\Auth\Authenticatable, null given,
```

Toch zijn de testen goed geschreven. Helaas kan je niet meerdere keren be() gebruiken in een test document, als je met RefreshDatabase werkt. Binnen Pest.php zullen we wat moeten veranderen. We moeten geen RefreshDatabase meer gebruiken, maar DatabaseMigrations. Hieronder zie je in commentaar wat de oude code was, en in code wat het nu moet worden.

```
Pest.php × ProductIndexTest.php ×

1  <?php
2
3  //use \Illuminate\Foundation\Testing\RefreshDatabase;
4  use \Illuminate\Foundation\Testing\DatabaseMigrations;
5  /*...*/
15
16  //uses(TestCase::class, RefreshDatabase::class)->in('Feature');
17  uses( ...classAndTraits: TestCase::class, DatabaseMigrations::class)->in( ...targets: 'Feature');
```

Als je nu nog een keer de test uit voert, zie je dat de test nu geheel goed gaat.

```
D:\wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductIndex

PASS Tests\Feature\Products\ProductIndexTest
✓ admin can see the product index page
✓ sales can see the product index page
✓ customer can not see the product index page

Tests: 3 passed
Time: 4.48s
```

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest

    PASS TestsFeatureProductIndexTest
✓ admin can see the product index page
✓ sales can see the product index page
✓ customer can not see the product index page

Tests: 3 passed
Time: 2.015
```

De laatste test is dat voor iemand die niet is ingelogd. Als je niet ingelogd bent, wordt je doorverwezen naar de login pagina.

```
->test( description: 'guest can not see the product index page', function(){
    $this->get(route( name: 'products.index'))
        ->assertRedirect(route( name: 'login'));
})->group( ...groups: 'ProductIndex');
```

Je ziet de assertRedirect als controle of je dus wel meteen naar een andere pagina wordt verwezen. De named route van login is erg makkelijk, dit is namelijk gewoon login. Als je het even niet weet kan je met artisan ook de routes opvragen met `php artisan route:list`. Je krijgt dan heel veel informatie zoals je hieronder ziet. (ja, het is erg klein, maar je krijgt ook zoveel info)

Domain	Method	URI	Name	Action	Middleware
	GET HEAD	/		App\Http\Controllers\Admin\CategoryController@index	web
	GET HEAD	_debugbar/assets/javascript	debugbar.assets.js	Barryvdh\Debugbar\Controllers\AssetController@js	App\Http\Middleware\Authenticate Spatie\Permission\Middlewares\PermissionMiddleware@index category
	GET HEAD	_debugbar/assets/stylesheets	debugbar.assets.css	Barryvdh\Debugbar\Controllers\AssetController@css	Barryvdh\Debugbar\Middleware\DebugbarEnabled Closure
	DELETE	_debugbar/cache/{key}/{tags?}	debugbar.cache_delete	Barryvdh\Debugbar\Controllers\CacheController@delete	Barryvdh\Debugbar\Middleware\DebugbarEnabled Closure
	GET HEAD	_debugbar/clockwork/{id}	debugbar.clockwork	Barryvdh\Debugbar\Controllers\OpenHandlerController@clockwork	Barryvdh\Debugbar\Middleware\DebugbarEnabled Closure
	GET HEAD	_debugbar/open	debugbar.openhandler	Barryvdh\Debugbar\Controllers\OpenHandlerController@handle	Barryvdh\Debugbar\Middleware\DebugbarEnabled Closure
	GET HEAD	_debugbar/telescope/{id}	debugbar.telescope	Barryvdh\Debugbar\Controllers\TelescopeController@show	Barryvdh\Debugbar\Middleware\DebugbarEnabled Closure
	GET HEAD	admin/categories	categories.index	App\Http\Controllers\Admin\CategoryController@index	web Spatie\Permission\Middlewares\RoleMiddleware@customer sales admin App\Http\Middleware\Authenticate
	POST	admin/categories	categories.store	App\Http\Controllers\Admin\CategoryController@store	Spatie\Permission\Middlewares\PermissionMiddleware@index category web Spatie\Permission\Middlewares\RoleMiddleware@customer sales admin App\Http\Middleware\Authenticate
	GET HEAD	admin/categories/create	categories.create	App\Http\Controllers\Admin\CategoryController@create	Spatie\Permission\Middlewares\RoleMiddleware@customer sales admin web Spatie\Permission\Middlewares\PermissionMiddleware:create category App\Http\Middleware\Authenticate
	GET HEAD	admin/categories/{category}	categories.show	App\Http\Controllers\Admin\CategoryController@show	Spatie\Permission\Middlewares\RoleMiddleware@customer sales admin web Spatie\Permission\Middlewares\PermissionMiddleware:create category Spatie\Permission\Middlewares\RoleMiddleware@customer sales admin

Wat ook handig is, is dat je meteen zien hoe de permissies zijn geregeld met middleware.

Het resultaat van de test kunnen we natuurlijk ook nog even bekijken.

```
D:\wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductIndex

  PASS  Tests\Feature\Products\ProductIndexTest
    ✓ admin can see the product index page
    ✓ sales can see the product index page
    ✓ customer can not see the product index page
    ✓ guest can not see the product index page

  Tests:  4 passed
  Time:   5.46s
```

Alle 4 de testen zijn nu ok. We missen alleen nog 1 ding, wat straks wel cruciaal is voor het testen of een pagina is zoals je het verwacht. Admin en Sales mogen er namelijk bij. We testen nu wel of we het zien, maar we kunnen ook de status checken. We hebben al eerder wat gezien over een status 200 en status 302 met de login.

We doen dus op de 2 testen van admin en sales nog een kleine aanvulling.

```
  test( description: 'admin can see the product index page', function()
{
    $admin = User::find(3);
    Laravel\be($admin)
        ->get(route( name: 'products.index'))
        ->assertSee($this->product->name)
        ->assertSee($this->product->category->name)
        ->assertSee($this->product->latest_price->price)
        ->assertStatus( status: 200);
})->group( ...groups: 'ProductIndex');

  test( description: 'sales can see the product index page', function()
{
    $sales = User::find(2);
    Laravel\be($sales)
        ->get(route( name: 'products.index'))
        ->assertSee($this->product->name)
        ->assertSee($this->product->category->name)
        ->assertSee($this->product->latest_price->price)
        ->assertStatus( status: 200);
})->group( ...groups: 'ProductIndex');
```

Als we checken wat het resultaat nu is, zien we dat alle testen nog steeds ok zijn.

```
D:\wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductIndex

  PASS  Tests\Feature\Products\ProductIndexTest
    ✓ admin can see the product index page
    ✓ sales can see the product index page
    ✓ customer can not see the product index page
    ✓ guest can not see the product index page

  Tests:  4 passed
  Time:   4.78s
```

Het testen van de product index is nu klaar. We hebben namelijk voor gasten, customer, sales en admin getest of ze erbij kunnen. Als ze erbij kunnen hebben we ook getest of ze een overzicht zien met de bepaalde data.

## 7.2. Product Create

Eerst maken we het testbestand aan in de map tests/Feature/Products. We noemen het bestand ProductCreateTest. Dit doen we met `php artisan pest:test Products/ProductCreateTest`

```
D:\wamp\sites\laravelv1>php artisan pest:test Products/ProductCreateTest
```

```
[OK] `tests/Feature/Products/ProductCreateTest.php` created successfully.
```

Daarna gaan we eerst nadenken wat we nodig zouden hebben. De guest en customer mogen er nog steeds niet bij. Deze tests zullen dus bijna hetzelfde zijn als bij de index.

De admin en sales mogen er wel bij. Deze moeten dan een formulier zien om een product in te vullen. Hier zullen we wel nog even moeten nadenken hoe we dit kunnen testen. Eerst dus even de standaard onderdelen in de test zetten en de tests voor guest en customer, want die zijn nu simpel.

```
use App\Models\User;
use App\Models\Category;
use \Pest\Laravel;

beforeEach(function (){
    $this->seed('RoleAndPermissionSeeder');
    $this->seed('UserSeeder');
    $this->category = Category::factory()->create();
});
```

We gaan de users en permissions weer nodig hebben. Een product create pagina heeft een formulier waarbij je moet aangeven welke categorie je wilt, dus er moet wel een categorie zijn. Er hoeft verder nog geen product of prijs te zijn om het formulier te zien.

De testen voor de customer en guest kunnen we kopieren vanuit de index test en aanpassen waar nodig. Deze tests zetten we in ProductCreate groep.

```
test( description: 'customer can not see the product create page', function(){
{
    $customer = User::find(1);
    Laravel\be($customer)
        ->get(route( name: 'products.create'))
        ->assertForbidden();
})->group( ...groups: 'ProductCreate');

test( description: 'guest can not see the product create page', function(){
    $this->get(route( name: 'products.create'))
        ->assertRedirect(route( name: 'login'));
})->group( ...groups: 'ProductCreate');
```

Als we gaan testen krijgen we een goed resultaat:

```
D:\wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductCreate

  PASS  Tests\Feature\Products\ProductCreateTest
    ✓ customer can not see the product create page
    ✓ guest can not see the product create page

  Tests:  2 passed
  Time:   2.75s
```

Nu nog de test voor de admin en sales. We kunnen controleren of we de goede view zien. Voorlopig is dit even genoeg.

```
test( description: 'admin can see the product create page', function(){
    $admin = User::find(3);
    Laravel\be($admin)
        ->get(route( name: 'products.create'))
        ->assertViewIs( value: 'admin.products.create')
        ->assertStatus( status: 200);;
})->group( ...groups: 'ProductCreate');

test( description: 'sales can see the product create page', function(){
    $sales = User::find(2);
    Laravel\be($sales)
        ->get(route( name: 'products.create'))
        ->assertViewIs( value: 'admin.products.create')
        ->assertStatus( status: 200);;
})->group( ...groups: 'ProductCreate');
```

Je ziet bij deze test even iets nieuws, namelijk assertViewIs. Hiermee kunnen we kijken of we de view zien die we verwachten. Bij de index hadden we dit ook kunnen doen, maar daar moest sowieso al gecontroleerd worden of er bepaalde data zichtbaar was.

Voor de create page is de test nu klaar. Je kan ervoor kiezen om de create en store tests bij elkaar te zetten, dit omdat ze aan elkaar verbonden zijn. Ik doe dat liever niet, omdat ik graag alle tests zo klein mogelijk houd. Als er dan ergens een test niet goed gaat, weet ik precies waar de fout zit.

We hebben dit namelijk al een keer gezien. De product was gemaakt en het verwijderen van een categorie moet aangepast worden. Als je de testen zo maakt dat je alle functionaliteiten apart test kan je zo achterhalen wat de fout is en wat je dus moet aanpassen.

Als we de test nu uitvoeren zien we dat alle 4 de testen er goed doorheen komen.

```
D:\wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductCreate

PASS  Tests\Feature\Products\ProductCreateTest
✓ customer can not see the product create page
✓ guest can not see the product create page
✓ admin can see the product create page
✓ sales can see the product create page

Tests: 4 passed
Time: 5.35s
```

### 7.3. Product Store

Bij de product store gaan we tests krijgen die wat groter worden. Dit omdat we allerlei dingen in het formulier moeten invullen en het formulier moeten opstellen voordat we iets kunnen met het resultaat.

We doen eerst even de users die er sowieso niet bij mogen. De beforeEach die we bij de andere tests hebben gebruikt moet natuurlijk er ook gewoon bij. Het gaat nu wel om de volledige before zoals we bij de index hadden. Het is handig om te controleren of we 2 producten met dezelfde naam mogen gebruiken. We gaan kijken naar de store, zullen we ook moeten testen of we erbij kunnen met de POST methode.

Let wel op, dat je niet dubbele dingen gaat testen. Er is al een test of een gebruiker bij de create kan, dus test dit niet meer bij de store. Bij de store ga je er vanuit dat er iets gepost wordt.

Als eerst de test aanmaken met `php artisan pest:test Products/ProductStoreTest`

```
D:\wamp\sites\laravelv1>php artisan pest:test Products/ProductStoreTest

[OK] `tests/Feature/Products/ProductStoreTest.php` created successfully.
```

We halen alles weg uit de ProductStoreTest en zetten hetzelfde begin als ProductIndexTest erin.

```
<?php

use App\Models\User;
use App\Models\Category;
use App\Models\Product;
use App\Models\Price;
use \Pest\Laravel;

beforeEach(function (){
    $this->seed('RoleAndPermissionSeeder');
    $this->seed('UserSeeder');
    $this->category = Category::factory()->create();
    $this->product = Product::factory()->create();
    $this->price = Price::factory()->create();
});

});
```

De gebruikers met permissies zijn er nu. Ook de categorie, product en prijs zitten in de database. Nu eerst de makkelijkste, namelijk die van guest en customer. Ze lijken op die van de index en create, maar er zit wel een klein verschil in.

```
test( description: 'guest can not create an product in the admin', function () {
    $this->postJson(route( name: 'products.store'))
        ->assertStatus(401);
})->group( ...groups: 'ProductStore');

test( description: 'customer can not create an product in the admin', function () {
    $customer = User::find(1);
    Laravel\be($customer)
        ->postJson(route( name: 'products.store'))
        ->assertForbidden();
})->group( ...groups: 'ProductStore');
```

Bij de guest zie je dat we met Json een post maken. Dit omdat Laravel dit namelijk verwacht bij een test. We hoeven niet eens data mee te sturen nog, want hij zou zonder data al moeten zien dat er geen permissie is, waardoor de 401 status komt.

We zetten deze tests in de groep ProductStore, zodat we het met die groep weer apart kunnen testen.

Als we dit dan nu testen zien we dan alles goed gaat.

```
D:\wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductStore

  PASS  Tests\Feature\Products\ProductStoreTest
    ✓ guest can not create an product in the admin
    ✓ customer can not create an product in the admin

Tests:  2 passed
Time:  2.90s
```

Nu de volgende, namelijk dat de admin een product kan toevoegen.

We hebben in de before al een categorie, product en prijs gemaakt. Deze kunnen we nu niet helemaal gebruiken, omdat we moeten testen of we het juist kunnen opslaan vanuit een form. Dit doen we met make(). Make maakt het wel aan maar slaat het nog niet op in de database, wat de create methode wel doet.

```
test( description: 'admin can create a product in the admin', function () {
    $admin = User::find(3);
    $product = Product::factory()->make();

    Laravel\be($admin)
        ->postJson(route('products.store'), $product->toArray())
        ->assertRedirect(route('products.index'));

    $this->assertDatabaseHas('products', [
        'name' => $product->name,
        'description' => $product->description
    ]);
})->group(...groups: 'ProductStore');
```

We zorgen dat we ingelogd zijn als admin en posten de waarden naar de store. We verwachten daarna een redirect richting de index. Als het goed is staat het product wat we hebben gepost dan in de database.

Als we het gaan testen merken we dat het er helaas niet doorheen komt.

```
D:\wamp\sites\laravelv1.\vendor\bin\pest --group=ProductStore

 FAIL Tests\Feature\Products\ProductStoreTest
 ✓ guest can not create an product in the admin
 ✓ customer can not create an product in the admin
 ✘ admin can create a product in the admin

---

• Tests\Feature\Products\ProductStoreTest > admin can create a product in the admin
Response status code [422] is not a redirect status code.
Failed asserting that false is true.

at D:\wamp\sites\laravelv1\tests\Feature\Products\ProductStoreTest.php:35
```

We krijgen een response status van 422, wat betekent dat de validatie niet is gelukt. Als we dan nog even naar de Request kijken die we hebben gemaakt:

```
public function rules()
{
    return [
        'name' => 'required|max:45|unique:products',
        'description' => 'required',
        'price' => 'required|numeric|max:999999.99'
    ];
}
```

Name en description hebben we, maar de price niet. Hierdoor komt het dus niet door de test. We moeten dus een prijs aanmaken en die ook meesturen.

```
test('admin can create a product in the admin', function () {
    $admin = User::find(3);
    $product = Product::factory()->make();
    $price = Price::factory()->make();

    Laravel\be($admin)
        ->postJson(route('products.store'), array_merge($product->toArray(), $price->toArray()))
        ->assertRedirect(route('products.index'));

    $this->assertDatabaseHas('products', [
        'name' => $product->name,
        'description' => $product->description
    ]);
})->group(...groups: 'ProductStore');
```

Je ziet dat de product en prijs worden aangemaakt. Bij het posten gebruiken we een array\_merge() om de arrays samen te voegen, zodat alle data in 1 keer komt, wat ook wordt verwacht door de validatie.

Als we nu weer de test uitvoeren zien we dat het wel goed gaat. We zijn nu dus wel door de validatie heen gekomen en een redirect gekregen naar de index.

```
D:\wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductStore

  PASS  Tests\Feature\Products\ProductStoreTest
    ✓ guest can not create an product in the admin
    ✓ customer can not create an product in the admin
    ✓ admin can create a product in the admin

  Tests:  3 passed
  Time:   4.27s
```

De vraag is wel, moeten we de prijs in een aparte test controleren of moet dat bij de product store? Het antwoord van de vraag is simpel. Zorgen we ervoor dat de prijs wordt opgeslagen in de product store?

Ja dus, dus we moeten dan ook controleren of deze prijs die we hebben aangemaakt ook in de database staat.

```
  test( description: 'admin can create a product in the admin', function () {
    $admin = User::find(3);
    $product = Product::factory()->make();
    $price = Price::factory()->make();

    Laravel\be($admin)
      ->postJson(route( name: 'products.store'), array_merge($product->toArray(),$price->toArray()))
      ->assertRedirect(route( name: 'products.index'));

    $this->assertDatabaseHas( table: 'products',[

      'name' => $product->name,
      'description' => $product->description
    ]);
    $this->assertDatabaseHas( table: 'prices',[

      'price' => $price->price
    ]);
  })->group( ...groups: 'ProductStore');
```

Let wel even op, dat je niet de effdate ook nog checked of die erin staat. Dit omdat we in de methode Carbon gebruiken voor de huidige datum en tijd.

```
$price = new Price();
$price->price = $request->price;
$price->effdate = Carbon::now();
$price->product_id = $product->id;
$price->save();
```

Als we dus eerst het opslaan en daarna een check doen, zal de datum net iets verschillen waardoor er altijd een error zal zijn. Je zal dus altijd goed moeten kijken wat je wel of niet kan testen. Als we nu weer de test uitvoeren zie je dat nog steeds alles goed is.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductStore

PASS Tests\Feature\Products\ProductStoreTest
✓ guest can not create an product in the admin
✓ customer can not create an product in the admin
✓ admin can create a product in the admin

Tests: 3 passed
Time: 2.58s
```

Nu de admin klaar is, zal de sales makkelijk zijn. Het enige verschil is namelijk de user.

```
test( description: 'sales can create a product in the admin', function () {
    $sales = User::find(2);
    $product = Product::factory()->make();
    $price = Price::factory()->make();

    Laravel\be($sales)
        ->postJson(route( name: 'products.store'), array_merge($product->toArray(),$price->toArray()))
        ->assertRedirect(route( name: 'products.index'));

    $this->assertDatabaseHas( table: 'products', [
        'name' => $product->name,
        'description' => $product->description
    ]);
    $this->assertDatabaseHas( table: 'prices', [
        'price' => $price->price
    ]);
})->group( ...groups: 'ProductStore');
```

En als we de test uitvoeren zie je dat we 4 geslaagde testen hebben.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductStore

  PASS Tests\Feature\Products\ProductStoreTest
    ✓ guest can not create an product in the admin
    ✓ customer can not create an product in the admin
    ✓ admin can create a product in the admin
    ✓ sales can create a product in the admin

  Tests: 4 passed
  Time: 3.07s
```

Als we nu alle testen uitvoeren, wat je soms wel tussendoor moet doen, zie je dat alle testen slagen.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest

  PASS Tests\Feature\AuthenticationTest
    ✓ login screen can be rendered
    ✓ users can authenticate using the login screen
    ✓ users can not authenticate with invalid password

  PASS Tests\Feature\EmailVerificationTest
    ✓ email verification screen can be rendered
    ✓ email can be verified
    ✓ email is not verified with invalid hash

  PASS Tests\Feature>PasswordConfirmationTest
    ✓ confirm password screen can be rendered
    ✓ password can be confirmed
    ✓ password is not confirmed with invalid password

  PASS Tests\Feature>PasswordResetTest
    ✓ reset password link screen can be rendered
    ✓ reset password link can be requested
    ✓ reset password screen can be rendered
    ✓ password can be reset with valid token

  PASS Tests\Feature\RegistrationTest
    ✓ registration screen can be rendered
    ✓ new users can register

  PASS Tests\Feature\Products\ProductCreateTest
    ✓ customer can not see the product create page
    ✓ guest can not see the product create page
    ✓ admin can see the product create page
    ✓ sales can see the product create page

  PASS Tests\Feature\Products\ProductIndexTest
    ✓ admin can see the product index page
    ✓ sales can see the product index page
    ✓ customer can not see the product index page
    ✓ guest can not see the product index page

  PASS Tests\Feature\Products\ProductStoreTest
    ✓ guest can not create an product in the admin
    ✓ customer can not create an product in the admin
    ✓ admin can create a product in the admin
    ✓ sales can create a product in the admin

  Tests: 27 passed
  Time: 10.68s
```

Zoals je ziet is het van belang om testen in een groep te zetten, zodat je de groep even tussendoor kan testen. Dus hebben we nu de volgende groepen:

- ProductIndex
- ProductCreate
- ProductStore

Let op, dat de beforeEach zelf geen test is. Deze kunnen we niet in een group zetten.

## 7.4. ProductStoreCheck

Door middel van deze testen hebben we nu getest dat als we netjes het formulier invullen we de data in de database krijgen als we dit doen met admin & sales. De enige vraag nu nog is, wat als we bijvoorbeeld de prijs niet invullen? Wordt het product er dan wel ingezet of niet? Dit soort testen zijn ook zeker van belang, want je gaat op dat moment heel erg naar je validatie kijken.

Dit kan gewoon met Pest, zoals je hieronder als voorbeeld ziet. Alleen is het van belang dat je niet alleen weet hoe Pest werkt, maar ook dat je gewoon met PHPUnit moet kunnen werken.

```
test( description: 'a product requires a name', function(){
    $admin = User::find(3);
    $product = Product::factory()->make(['name' => null]);
    $price = Price::factory()->make();

    Laravel\be($admin)
        ->postJson(route('products.store'), array_merge($product->toArray(),$price->toArray()))
        ->assertStatus( status: 422);

    $this->assertDatabaseMissing('products', [
        'name' => $product->name,
        'description' => $product->description
    ]);
    $this->assertDatabaseMissing('prices', [
        'price' => $price->price
    ]);
})->group( ...groups: 'ProductStoreCheck');
```

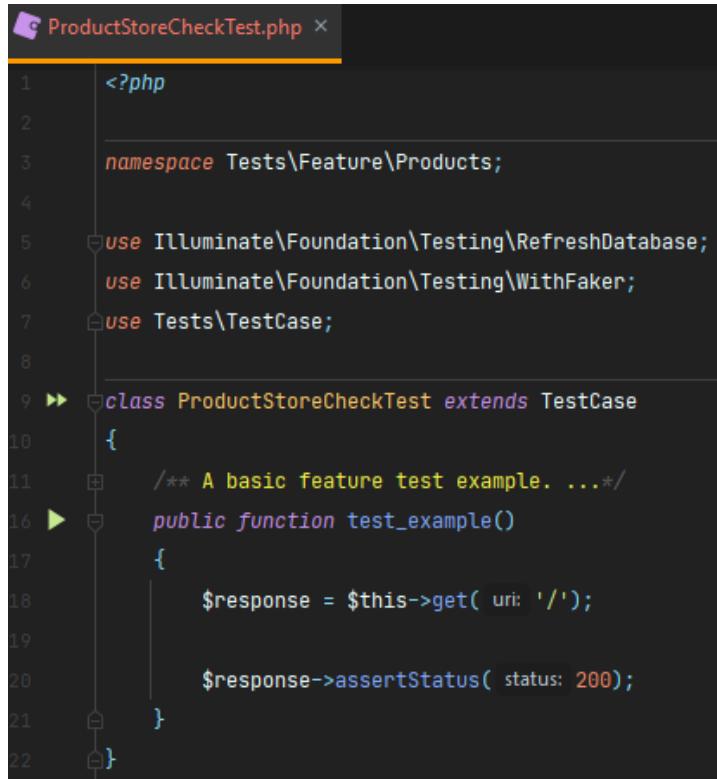
Gelukkig kan je PHPUnit en Pest bij elkaar gebruiken. We gaan dus nu een aantal tests met PHPUnit schrijven. Hierdoor leer je ook beide manieren, wat natuurlijk beter is.

Omdat we nu met PHPUnit gaan werken, hebben we dus een class nodig etc. Dus alles wat we bij Pest konden weglaten hebben we nu wel nodig. We maken dan ook de test aan met:

```
php artisan make:test Products/ProductStoreCheckTest
```

```
D:\Wamp\sites\laravelv1>php artisan make:test Products/ProductStoreCheckTest
Test created successfully.
```

De test ziet er nu zo uit



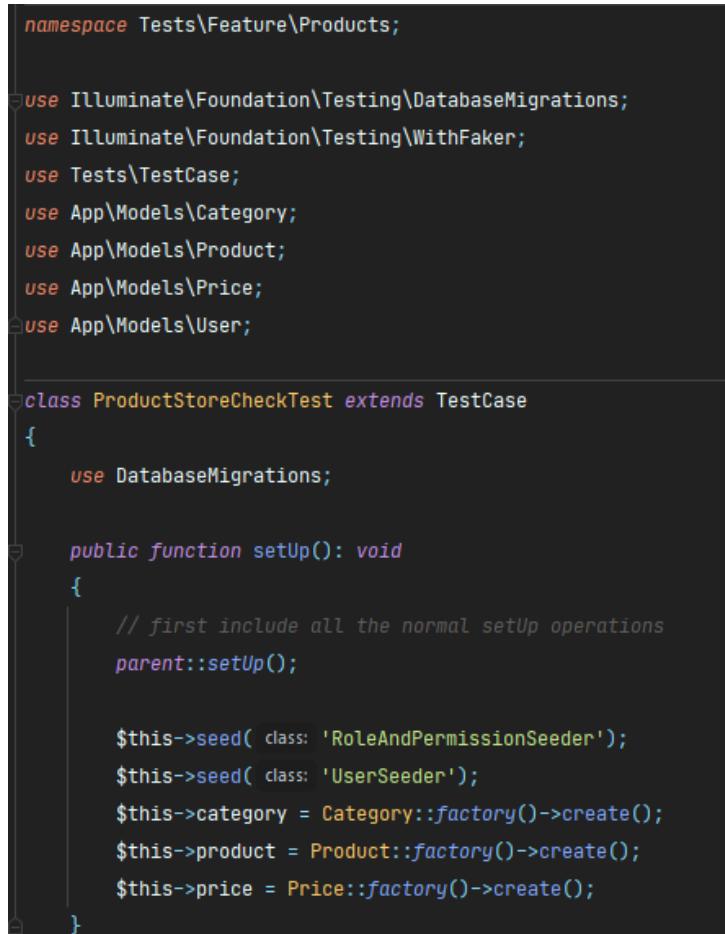
```
<?php

namespace Tests\Feature\Products;

use Illuminate\Foundation\Testing\RefreshDatabase;
use Illuminate\Foundation\Testing\WithFaker;
use Tests\TestCase;

class ProductStoreCheckTest extends TestCase
{
    /** A basic feature test example. ...*/
    public function test_example()
    {
        $response = $this->get( uri: '/');
        $response->assertStatus( status: 200);
    }
}
```

We gaan dit aanpassen, omdat we bijvoorbeeld niet met RefreshDatabase werken etc. De basis van de test in phpunit is dan dit.



```
namespace Tests\Feature\Products;

use Illuminate\Foundation\Testing\DatabaseMigrations;
use Illuminate\Foundation\Testing\WithFaker;
use Tests\TestCase;
use App\Models\Category;
use App\Models\Product;
use App\Models\Price;
use App\Models\User;

class ProductStoreCheckTest extends TestCase
{
    use DatabaseMigrations;

    public function setUp(): void
    {
        // first include all the normal setUp operations
        parent::setUp();

        $this->seed( class: 'RoleAndPermissionSeeder');
        $this->seed( class: 'UserSeeder');
        $this->category = Category::factory()->create();
        $this->product = Product::factory()->create();
        $this->price = Price::factory()->create();
    }
}
```

We gebruiken nu gewoon de database migrations en niet refreshdatabase. Wat we normaal in de beforeEach hadden, zetten we nu in de setup. We zitten nu in een class, dus kunnen netjes de use Category, Product en Price toevoegen.

De testExample er heel even ingelaten zodat we kunnen kijken of het werkt. Wel de assertStatus op 302 gezet, omdat we een redirect krijgen vanwege het niet ingelogd zijn.

```
/** A basic feature test example. ...*/
public function test_example()
{
    $response = $this->get( uri: '/');

    $response->assertStatus( status: 302);
}
```

De test staat gewoon bij de uitvoer.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest
```

```
PASS | Tests\Feature\AuthenticationTest
✓ login screen can be rendered
✓ users can authenticate using the login screen
✓ users can not authenticate with invalid password
```

```
PASS | Tests\Feature\Products\ProductStoreCheckTest
✓ example
```

```
PASS | Tests\Feature\Products\ProductStoreTest
✓ guest can not create an product in the admin
✓ customer can not create an product in the admin
✓ admin can create a product in the admin
✓ sales can create a product in the admin

Tests: 28 passed
Time: 11.45s
```

Het maakt Pest dus niet uit, of je het met phpunit schrijft, of met Pest. Dit is natuurlijk erg handig, want we kunnen dus de testen die op dit moment nog niet met Pest goed gaan schrijven met phpunit.

In de validatie worden er een aantal dingen getest, die we in verschillende testen steeds kunnen beschrijven. Hierdoor wordt het makkelijk als we met een standaard functie komen om een POST te doen van ons product, waarbij we kunnen aangeven welke aanpassing we willen maken.

```

public function postProduct($overrides = [])
{
    $product = Product::factory()->make($overrides);
    $price = Price::factory()->make();
    //return $this->post(route('products.store'), $product->toArray());
    return $this->postJson(route('products.store'), array_merge($product->toArray(), $price->toArray()));
}

```

Omdat we toch in een class zitten kunnen we deze methode gewoon erin hebben. Deze wordt niet zomaar uitgevoerd, want met phpunit wordt de test (functie) pas uitgevoerd als je aangeeft dat het een test is. Dit doe je met `/** @test */` ervoren.

```

/** @test */
function a_product_requires_a_name()

```

Nog even terug naar de functie die net geschreven is voor `postProduct`. Dan zie je dus dat ik netjes de `make` functie gebruik, en de variabel `$overrides` zullen we straks gebruiken voor de uitzonderingen op onze factory. Bijvoorbeeld geen naam, description of prijs.

Binnen de `ProductStoreCheckTest` gaan we dan ook de test schrijven. Let op dat je boven de test `/** @test */` hebt staan, anders wordt de test overgeslagen.

```

/** @test
 * @group ProductStoreCheck
 */
function a_product_requires_a_name()
{
    $admin = User::find(3);
    $this->actingAs($admin);
    $this->postProduct(['name' => NULL])->assertStatus(status: 422);

}

```

We testen nu dat een product een naam nodig heeft. Volgens de validatie namelijk wel. We halen de admin gebruiker op zodat we er bij kunnen, daarna maken we de product aan met `postProduct`. Hier zorgen we voor de uitzondering dat `name = null`, een onbekende waarde. Ook is de `be()` functie die we in Pest gebruikte nu een `actingAs`, maar heeft precies dezelfde functionaliteit.

Volgens de http status zou er dan 422 uit moeten komen, want we komen als het goed is niet door de validatie heen.

Heb je ook op de naam van de functie gelet. In Pest deden we dit in een string, alleen in PHPUnit doen we dit in de functienaam. Het moet nog steeds duidelijk zijn wat we testen!

Als we de test nu uitvoeren kunnen we dit gewoon met Pest weer doen.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductStoreCheck

  PASS  Tests\Feature\Products\ProductStoreCheckTest
    ✓ a product requires a name

  Tests:  1 passed
  Time:  0.96s
```

Yes, opzet geslaagd. De test is gemaakt met phpUnit!

Het gebruik maken van een functie die dan wordt gebruikt in een test, kan dit nu ook in Pest ????  
Natuurlijk kan dit. Dit zou op deze manier moeten:

```
function postProduct($overrides = [])
{
    $product = Product::factory()->make($overrides);
    $price = Price::factory()->make();
    return Laravel\postJson(route('products.store'), array_merge($product->toArray(), $price->toArray()));
}

test('a product requires a name', function() {
    $admin = User::find(3);

    Laravel\be($admin);
    postProduct(['name' => null])
        ->assertStatus(status: 422);
})->group(...groups: 'ProductStoreCheck');
```

Je ziet dus, het lijkt zoveel op elkaar. Soms is alleen Pest wat makkelijk te schrijven. Maar voor de ProductStoreCheckTest gaan we wel door met PhpUnit. Maar voordat we verder gaan, is het goed om is te kijken naar waarom we die status 422 krijgen. Dit kunnen we mooi doen door middel van de geïnstalleerde debugbar.

## 7.5. ProductStoreCheck met debugbar

Om de test na te doen, loggen we in als admin en gaan we naar de products create pagina. Ik heb wel even bij name de required in de html weggehaald, zodat we de test kunnen uitvoeren.

The screenshot shows a Laravel application's admin interface. The URL in the browser is `laravelv1/admin/products/create`. The page title is "Product Admin". On the left sidebar, there are links for "My Admin", "Hello Admin" (with a placeholder "Role...."), "Category Admin", and "Product Admin". The main content area contains form fields for "ProductName", "Description", "Price", and "Category". The "ProductName" field is empty. The "Description" field contains the text "test zonder name". The "Price" field contains the value "1.00". The "Category" field contains the value "Louisa Hessel". A green "Submit" button is located at the bottom of the form.

We vullen tevens de description en price correct in, maar laten de productname leeg. Als we dan op submit klikken zien we netjes de validatie zijn werk doen. We krijgen een melding dat de name field verplicht is.

The screenshot shows the "Product Admin" page. A red box at the top displays the validation error message: "The name field is required." Below the error message, the form fields for "ProductName", "Description", "Price", and "Category" are visible. The "ProductName" field is empty.

ProductName

Laten we verder kijken met de debugbar. Als we op het mapje rechts onder klikken krijgen we dit te zien

DATE	METHOD	URL
2021-04-16 20:21:02	POST	/admin/products
2021-04-16 20:21:02	GET	/admin/products/create
2021-04-16 20:19:17	GET	/admin/products/create
2021-04-16 20:19:16	GET	/admin/products

Je ziet hier de geschiedenis wat je hebt gedaan. We zaten eerst op de products/create met de GET. Daarna hebben we een POST gedaan naar /products. Nu kan je op /products klikken om meer info hierover te krijgen.

Als we dan naar Exceptions gaan zien we dit. (exceptions is nu een rood vlak met het een 1 erbij)

The given data was invalid.  
D:\Wamp\sites\laravelv1\vendor\Laravel\framework\src\Illuminate\Foundation\Http\FormRequest.php#137

```
*/  
protected function failedValidation(Validator $validator)  
{  
    throw (new ValidationException($validator))  
        ->errorBag($this->errorBag)  
        ->redirectTo($this->getRedirectUrl());  
}
```

Bij de exception staat, dat de data niet correct was. Tevens zie je ook de methode failedValidation.

Dit zou betekenen, dat we een 422 moeten krijgen.... Wel weten we dat we dit onderdeel van de test verder goed hebben geschreven, want we krijgen ook een 422.

Als we naar de Gate gaan, zie je hoe het zit met permissies

```
★ array:4 [▼  
    "ability" => "create product"  
    "result" => true  
    "user" => 3  
    "arguments" => "["]  
]
```

Je ziet dat je de permissie 'create product' moet hebben, en dat de gebruiker (3 = admin) toegang heeft.

Nu is kijken bij de request.

Request	
path_info	/admin/products
status_code	302
status_text	Found
format	html
content_type	text/html; charset=UTF-8
request_query	[]
request_request	<pre>array:5 [▼     "_token" =&gt; "pbHjDuzs3shFUMcX4gVv65SMGieFfdiEdIcN8bZc"     "name" =&gt; null     "description" =&gt; "test zonder name"     "price" =&gt; "1.00"     "category_id" =&gt; "1" ]</pre>

Daar zien we opeens de status\_code 302. Hoe kunnen we nu een status code 422 krijgen uit de test, maar hier ook nog een status code 302. Dit is nu de reden waarom we met de test een postJson doen, anders zouden we namelijk de 302 gaan krijgen uit onze error.

Om dit even te checken kunnen we in de test even onze functie wijzigen:

```
public function postProduct($overrides = [])
{
    $product = Product::factory()->make($overrides);
    $price = Price::factory()->make();
    return $this->post(route('products.store'), array_merge($product->toArray(), $price->toArray()));
}
```

Als we de test dan uitvoeren:

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductStoreCheck

FAIL Tests\Feature\Products\ProductStoreCheckTest
  × a product requires a name
  --
  • Tests\Feature\Products\ProductStoreCheckTest > a product requires a name
    Expected status code 422 but received 302.
    Failed asserting that 422 is identical to 302.
```

En ja hoor, we krijgen opeens een 302 eruit, wat ook in de debugbar er staat.

We veranderen natuurlijk het wel weer even terug:

```
public function postProduct($overrides = [])
{
    $product = Product::factory()->make($overrides);
    $price = Price::factory()->make();
    return $this->postJson(route('products.store'), array_merge($product->toArray(), $price->toArray()));
}
```

Nog even een keer kijken naar de debugbar request:

	Messages	Timeline	Exceptions 1	Views 0	Route	Queries 5	Models 2	Mails	Gate 1	Session	Request	
path_info					/admin/products							
status_code					302							
status_text					Found							
format					html							
content_type					text/html; charset=UTF-8							
request_query					[]							
request_request					array:5 [▼ "_token" => "pbHjDuzs3shFUMcX4gVv65SMGieFfdiEdIcN8bZc" "name" => null "description" => "test zonder name" "price" => "1.00" "category_id" => "1" ]							

Je ziet hier duidelijk dat bij name de null is gepost, maar ook bijv dat er een token mee wordt gestuurd (de csrf token). Al dit soort dingen zijn handig als je een bepaalde foutmelding tegen komt.

## 7.6. Verder met ProductStoreCheck

Nu we dit resultaat hebben kunnen we natuurlijk redelijk wat testen maken. Dit zullen allerlei dingen zijn om de volledige validatie te checken. Hieronder de volgende onderdelen:

- Product naam maximaal 45 characters
- Product naam moet uniek zijn
- Product heeft een beschrijving nodig

```
/** @test
 *  @group ProductStoreCheck
 */
function a_product_name_can_be_max_45_characters()
{
    $admin = User::find(3);
    $this->actingAs($admin);
    $this->postProduct(['name' => '1234567890123456789012345678901234567890123456'])
        ->assertStatus( status: 422);
}

/** @test
 *  @group ProductStoreCheck
 */
function a_product_name_can_must_be_unique()
{
    $admin = User::find(3);
    $product = Product::find(1);
    $this->actingAs($admin);
    $this->postProduct(['name' => $product->name])
        ->assertStatus( status: 422);
}

/** @test
 *  @group ProductStoreCheck
 */
function a_product_requires_a_description()
{
    $admin = User::find(3);
    $this->actingAs($admin);
    $this->postProduct(['description' => null])
        ->assertStatus( status: 422);
}
```

Daarna nog deze:

- Product heeft een prijs nodig
- Product prijs moet een getal zijn
- Product prijs mag maximaal 999.999,99 zijn

Dit heeft wel met prijs te maken, en dan hebben we met de gemaakte code nog een uitdaging.

```
public function postProduct($overrides = [])
{
    $product = Product::factory()->make($overrides);
    $price = Price::factory()->make();
    return $this->postJson(route('products.store'), array_merge($product->toArray(), $price->toArray()));
}
```

Zoals je ziet staat \$overrides bij het maken van product. Als we de prijs willen testen zullen we de postProduct aan moeten passen zodat we ook wat voor price aanpassingen kunnen toevoegen. In code moet het altijd duidelijk zijn wat de bedoeling is, dus dit wordt het dan:

```
public function postProduct($overridesProduct = [], $overridesPrice = [])
{
    $product = Product::factory()->make($overridesProduct);
    $price = Price::factory()->make($overridesPrice);
    return $this->postJson(route('products.store'), array_merge($product->toArray(), $price->toArray()));
}
```

Je ziet dat we nu 2 variabelen in postProduct kunnen zetten. De namen zijn ook anders geworden. We hebben het zo gemaakt, dat we de eerste testen niet anders hoeven te maken. De laatste 3 testen gaan we nu schrijven voor de prijs.

```
/** @test
 * @group ProductStoreCheck
 */
function a_product_requires_a_price()
{
    $admin = User::find(3);
    $this->actingAs($admin);
    $this->postProduct([], ['price' => null])
        ->assertStatus(status: 422);
}
```

Let even goed op bij deze test, we zetten nu in postProduct 2 arrays. Eerst een lege array omdat we niks willen aanpassen in product en dan de wijziging in de 2<sup>e</sup> array om prijs aan te passen.

De laatste 2 testen zien er dan zo uit:

```
/** @test
*  @group ProductStoreCheck
*/
function a_product_price_should_be_a_number()
{
    $admin = User::find(3);
    $this->actingAs($admin);
    $this->postProduct([], ['price' => 'abc'])
        ->assertStatus( status: 422);
}

/** @test
*  @group ProductStoreCheck
*/
function a_product_price_can_be_max_999999_99()
{
    $admin = User::find(3);
    $this->actingAs($admin);
    $this->postProduct([], ['price' => 1000000.00])
        ->assertStatus( status: 422);
}
```

Alle testen zitten netjes in de groep ProductStoreCheck. Als we nu de test uitvoeren zal je zien dat inderdaad alle testen correct zijn.

```
D:\Wamp\sites\laravelv1.\vendor\bin\pest --group=ProductStoreCheck

  PASS | Tests\Feature\Products\ProductStoreCheckTest
  ✓ a product requires a name
  ✓ a product name can be max 45 characters
  ✓ a product name can must be unique
  ✓ a product requires a description
  ✓ a product requires a price
  ✓ a product price should be a number
  ✓ a product price can be max 999999 99

  Tests:  7 passed
  Time:  5.10s
```

## 7.7. Product edit

Nu de product store met validatie checks gedaan zijn, kunnen we verder met de edit. Je zal merken dat de tests van de edit natuurlijk erg lijken op de store. Hierdoor gaan we alleen de nieuwe punten even behandelen. Wel zal de code van de gehele test er staan, zodat je een totaal beeld hebt van een crud test.

Wel gaan we dit nu weer met Pest doen (leuk heh, Pest en phpunit door elkaar gebruiken). Als eerst maken we de test aan.

```
D:\Wamp\sites\laravelv1>php artisan pest:test Products/ProductEditTest
```

```
[OK] `tests/Feature/Products/ProductEditTest.php` created successfully.
```

Als eerst de opzet van de edit test. We moeten een bestaand product wijzigen, dus in de before each hebben we nu wel nodig dat we dit allemaal aanmaken.

```
use App\Models\User;
use App\Models\Category;
use App\Models\Product;
use App\Models\Price;
use \Pest\Laravel;

beforeEach(function (){
    $this->seed( class: 'RoleAndPermissionSeeder');
    $this->seed( class: 'UserSeeder');
    $this->category = Category::factory()->create();
    $this->product = Product::factory()->create();
    $this->price = price::factory()->create();
});

test( description: 'customer can not see the product edit page', function()
{
    $customer = User::find(1);
    Laravel\be($customer)
        ->get(route( name: 'products.edit', ['product' => $this->product->id]))
        ->assertForbidden();
})->group( ...groups: 'ProductEdit');

test( description: 'guest can not see the product edit page', function(){
    $this->get(route( name: 'products.edit', ['product' => $this->product->id]))
        ->assertRedirect(route( name: 'login'));
})->group( ...groups: 'ProductEdit');
```

Daarnaast ziet je dat we in de route netjes de product->id gebruiken van het bestaande product, om bij de pagina voor edit te komen.

De admin en sales kunnen er natuurlijk wel bij. We kunnen dan meteen meenemen of we wel de informatie in het formulier zien die we verwachten.

```
test( description: 'admin can see the product edit page', function(){
    $admin = User::find(3);
    Laravel\be($admin)
        ->get(route( name: 'products.edit', ['product' => $this->product->id]))
        ->assertViewIs( value: 'admin.products.edit')
        ->assertSee($this->category->name)
        ->assertSee($this->product->name)
        ->assertSee($this->product->description)
        ->assertSee($this->product->latest_price->price)
        ->assertStatus( status: 200);
})->group( ...groups: 'ProductEdit');

test( description: 'sales can see the product edit page', function(){
    $sales = User::find(2);
    Laravel\be($sales)
        ->get(route( name: 'products.edit', ['product' => $this->product->id]))
        ->assertViewIs( value: 'admin.products.edit')
        ->assertSee($this->category->name)
        ->assertSee($this->product->name)
        ->assertSee($this->product->description)
        ->assertSee($this->product->latest_price->price)
        ->assertStatus( status: 200);
})->group( ...groups: 'ProductEdit');
```

Binnen de beforeEach hebben we natuurlijk \$this->category en \$this->product. De prijs zou de laatste prijs zijn, dus net zoals in de view kunnen we erbij met de relatie die in de model staat.

Als we de test uitvoeren zien we dat het netjes allemaal werkt

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductEdit

  PASS  Tests\Feature\Products\ProductEditTest
    ✓ customer can not see the product edit page
    ✓ guest can not see the product edit page
    ✓ admin can see the product edit page
    ✓ sales can see the product edit page

  Tests:  4 passed
  Time:   3.04s
```

De ProductEditTest is klaar.

## 7.8. Product update

Bij de product update gaan we kijken of de permissies weer goed werken en of de update goed werkt als we ons netjes aan de regels houden die in de validatie staan. Als eerst de test aanmaken.

```
D:\Wamp\sites\laravelv1>php artisan pest:test Products/ProductUpdateTest
```

```
[OK] `tests/Feature/Products/ProductUpdateTest.php` created successfully.
```

Ook hier weer eerst de beforeEach, en dat de guest en customer er niet mogen komen.

```
use App\Models\User;
use App\Models\Category;
use App\Models\Product;
use App\Models\Price;
use \Pest\Laravel;

beforeEach(function (){
    $this->seed( class: 'RoleAndPermissionSeeder');
    $this->seed( class: 'UserSeeder');
    $this->category = Category::factory()->create();
    $this->product = Product::factory()->create();
    $this->price = price::factory()->create();
});

test( description: 'customer can not update a product', function () {
    $customer = User::find(1);
    $product = Product::find(1);
    $price = Price::find(1);
    Laravel\be($customer)
        ->patchJson(route( name: 'products.update', ['product' => $this->product->id]),
            array_merge($product->toArray(),$price->toArray()))
        ->assertForbidden();
})->group( ...groups: 'ProductUpdate');

test( description: 'guest can not update a product', function () {
    $product = Product::find(1);
    $price = Price::find(1);
    $this->patchJson(route( name: 'products.update', ['product' => $this->product->id]),
        array_merge($product->toArray(),$price->toArray()))
    ->assertStatus( status: 401);
})->group( ...groups: 'ProductUpdate');
```

Deze keer iets uitgebreider, zodat er echt een product wordt opgezocht en proberen te updaten, al maakt het niet uit want er zal geen resultaat zijn.

Let op dat we nu met een update werken. Binnen Laravel is een update geen post, maar een put of patch. Omdat we al bij de edit test kijken of we bij het formulier mogen komen hoeft dat nu niet meer. (dit was dus ook zo bij de store met create)

Als we wel zijn ingelogd, maar niet de rechten hebben krijgen we Forbidden mee. Zijn we niet ingelogd maar proberen wel wat te updaten, krijgen we een unauthorized melding, wat de status 401 heeft.

Als we wel iets kunnen updaten wordt de test wel wat lastiger en groter. Van de categorie die we in de beforeEach maken, gaan we alles veranderen. Dus de name, description, price en category\_id. Voor de categorie moeten we dus een nieuwe categorie aanmaken in de test.

```
test( description: 'admin can update a product', function () {
    $admin = User::find(3);
    $newcategory = Category::factory()->create(['name' => 'TestCategory']);
    Laravel\be($admin)
        ->patchJson(route('products.update', ['product' => $this->product->id]),
            ['name' => 'Een productnaam',
             'description' => 'Blablabla',
             'price' => 1.00,
             'category_id' => $newcategory->id]
        );
    $this->product = $this->product->fresh();

    $this->get(route('products.index'))
        ->assertSee('value: Een productnaam')
        ->assertSee('value: 1.00')
        ->assertSee($newcategory->name);

    $this->get(route('products.index'))
        ->assertSee($this->product->name)
        ->assertSee($this->product->category->name)
        ->assertSee($this->product->latest_price->price);
})->group(...groups: 'ProductUpdate');
```

Bij de patch zullen we een array moeten meegeven met de waardes die naar de update methode gestuurd wordt. Hier zetten we dus alle veranderingen in. Daarna laden we opnieuw \$this->product in met ->fresh(). Dit zorgt ervoor dat het product gewoon weer uit de database wordt gehaald. Als het goed is, staan hier de nieuwe gegevens in. We controleren dan of deze gegevens er werkelijk staan.

Als we de test uitvoeren zal je zien dat de test nu werkt.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductUpdate

  PASS  Tests\Feature\Products\ProductUpdateTest
    ✓ customer can not update a product
    ✓ guest can not update a product
    ✓ admin can update a product

  Tests:  3 passed
  Time:   2.33s
```

Voor Sales lijkt de test natuurlijk wel op die van de admin. De waardes die we meegeven voor de wijziging zijn alleen wat anders.

```
  test( description: 'sales can update a product', function () {
    $sales = User::find(2);
    $newcategory = Category::factory()->create(['name' => 'Nog een categorie']);
    Laravel\be($sales)
      ->patchJson(route('products.update', ['product' => $this->product->id]),
        ['name' => 'Nog een product',
         'description' => 'Lorem ipsum',
         'price' => 2.00,
         'category_id' => $newcategory->id]
      );

    $this->product = $this->product->fresh();

    $this->get(route('products.index'))
      ->assertSee('value: Nog een product')
      ->assertSee('value: 2.00')
      ->assertSee($newcategory->name);

    $this->get(route('products.index'))
      ->assertSee($this->product->name)
      ->assertSee($this->product->category->name)
      ->assertSee($this->product->latest_price->price);
  })->group(...groups: 'ProductUpdate');
```

En ook nu zien we als de test wordt uitgevoerd dat alles goed is.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductUpdate

  PASS  Tests\Feature\Products\ProductUpdateTest
    ✓ customer can not update a product
    ✓ guest can not update a product
    ✓ admin can update a product
    ✓ sales can update a product

  Tests:  4 passed
  Time:   3.07s
```

## 7.9. ProductUpdateCheck

Nu de ProductUpdateTest klaar is, gaan we aan de slag om de validatie ervan uit te testen. De vorige keer met een check hebben we het in PHPUnit gemaakt. Laten we het nu in Pest maken.

```
D:\Wamp\sites\laravelv1>php artisan pest:test Products/ProductUpdateCheckTest

[OK] `tests/Feature/Products/ProductUpdateCheckTest.php` created successfully.
```

De basis is als eerst de use en de beforeEach. We maken alles standaard aan.

```
use App\Models\User;
use App\Models\Category;
use App\Models\Product;
use App\Models\Price;
use \Pest\Laravel;

beforeEach(function () {
    $this->seed( class: 'RoleAndPermissionSeeder' );
    $this->seed( class: 'UserSeeder' );
    $this->category = Category::factory()->create();
    $this->product = Product::factory()->create();
    $this->price = Price::factory()->create();
});
```

Daarna komt het lastige, we moeten nu de functie maken die we steeds gaan gebruiken.

```
function patchProduct($overridesProduct = [], $overridesPrice = []){  
    $product = Product::find(1)->make($overridesProduct);  
    $price = Price::find(1)->make($overridesPrice);  
    return Laravel\patchJson(route('products.update', ['product' => 1]),  
        array_merge($product->toArray(), $price->toArray()));  
}
```

Let op, dat bij een phpunit test we in een class zitten, dus objecten kunnen benaderen via \$this. Nu kan dit niet, waardoor het soms wat lastiger is om een variabel te verkrijgen. Ook hier zullen we weer product en price aparte overrides moeten geven. We pakken steeds product met id=1, dus hierdoor wordt het wat makkelijker.

Dezelfde tests voeren we uit als bij de store, alleen dan met patchProduct.

```
test('a product requires a name', function(){  
    $admin = User::find(3);  
    Laravel\be($admin);  
    patchProduct(['name' => null])  
        ->assertStatus(status: 422);  
})->group(...groups: 'ProductUpdateCheck');  
  
test('a product name can be max 45 characters', function(){  
    $admin = User::find(3);  
    Laravel\be($admin);  
    patchProduct(['name' => '1234567890123456789012345678901234567890123456'])  
        ->assertStatus(status: 422);  
})->group(...groups: 'ProductUpdateCheck');  
  
test('a product name must be unique', function(){  
    $admin = User::find(3);  
    $product = Product::factory()->create(['name' => 'Product1']);  
    Laravel\be($admin);  
    patchProduct(['name' => 'Product1'])  
        ->assertStatus(status: 422);  
})->group(...groups: 'ProductUpdateCheck');  
  
test('a product requires a description', function(){  
    $admin = User::find(3);  
    Laravel\be($admin);  
    patchProduct(['description' => null])  
        ->assertStatus(status: 422);  
})->group(...groups: 'ProductUpdateCheck');
```

Je ziet, als je eenmaal weet hoe het moet, valt het op zich erg mee om testen te schrijven. Hieronder het gedeelte van de price.

```
test( description: 'a product requires a price', function(){
    $admin = User::find(3);
    Laravel\be($admin);
    patchProduct([], ['price' => null])
        ->assertStatus( status: 422);
})->group( ...groups: 'ProductUpdateCheck');

test( description: 'a product price should be a number', function(){
    $admin = User::find(3);
    Laravel\be($admin);
    patchProduct([], ['price' => 'abc'])
        ->assertStatus( status: 422);
})->group( ...groups: 'ProductUpdateCheck');

test( description: 'a product price can be max 999999.99', function(){
    $admin = User::find(3);
    Laravel\be($admin);
    patchProduct([], ['price' => 1000000.00])
        ->assertStatus( status: 422);
})->group( ...groups: 'ProductUpdateCheck');
```

Als we de testen gaan uitvoeren zien we dit. Het werkt toch niet, want ovaal zijn errors nu.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductUpdateCheck

FAIL Tests\Feature\Products\ProductUpdateCheckTest
× a product requires a name
× a product name can be max 45 characters
× a product name must be unique
× a product requires a description
× a product requires a price
× a product price should be a number
× a product price can be max 999999.99

---
• Tests\Feature\Products\ProductUpdateCheckTest > a product requires a name
Illuminate\Database\Eloquent\MassAssignmentException

Add [name] to fillable property to allow mass assignment on [App\Models\Product].
```

Maar, als je de errors leest, zie je dat het gaat om mass assignment. Dit kunnen we oplossen door de Models aan te passen. In de product model zetten we dan netjes de fillable.

```
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
    use HasFactory;

    protected $fillable = ['name'];
```

En tevens in de price model

```
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Price extends Model
{
    use HasFactory;

    protected $fillable = ['price'];
```

Als we nu de test uitvoeren zie je dat alles goed gaat. De ProductUpdateCheck is klaar.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductUpdateCheck

  PASS  Tests\Feature\Products\ProductUpdateCheckTest
    ✓ a product requires a name
    ✓ a product name can be max 45 characters
    ✓ a product name must be unique
    ✓ a product requires a description
    ✓ a product requires a price
    ✓ a product price should be a number
    ✓ a product price can be max 999999.99

  Tests:  7 passed
  Time:  5.10s
```

## 7.10. Product Delete

Voor de product delete zal de test op zich simpel zijn. Dit omdat de delete erg veel lijkt op de edit, alleen zijn de inputs disabled. Wel moeten we er even op letten dat alleen een admin een product mag deleten. De sales heeft dus geen permissie om naar de delete pagina te gaan.

Als eerst het aanmaken van de test.

```
D:\Wamp\sites\laravelv1>php artisan pest:test Products/ProductDeleteTest
[OK] `tests/Feature/Products/ProductDeleteTest.php` created successfully.
```

Daarna de opzet van de test met de use en de beforeEach.

```
use App\Models\User;
use App\Models\Category;
use App\Models\Product;
use App\Models\Price;
use \Pest\Laravel;

beforeEach(function (){
    $this->seed( class: 'RoleAndPermissionSeeder');
    $this->seed( class: 'UserSeeder');
    $this->category = Category::factory()->create();
    $this->product = Product::factory()->create();
    $this->price = price::factory()->create();
});
```

Voor de guest is het net zoals bij de edit

```
test( description: 'guest can not see the product delete page', function(){
    $this->get(route( name: 'products.delete', [ 'product' => $this->product->id]))
        ->assertRedirect(route( name: 'login'));
})->group( ...groups: 'ProductDelete');
```

De sales mag niet erbij komen, dus die zou hetzelfde als de customer behandeld moet worden.

```
test( description: 'customer can not see the product delete page', function(){
{
    $customer = User::find(1);
    Laravel\be($customer)
        ->get(route( name: 'products.delete', ['product' => $this->product->id]))
        ->assertForbidden();
})->group( ...groups: 'ProductDelete');

test( description: 'sales can not see the product delete page', function(){
    $sales = User::find(2);
    Laravel\be($sales)
        ->get(route( name: 'products.delete', ['product' => $this->product->id]))
        ->assertForbidden();
})->group( ...groups: 'ProductDelete');
```

De admin kan het natuurlijk wel zien. Daardoor kunnen we controleren of we gegevens van het product nu kunnen zien.

```
test( description: 'admin can see the product delete page', function(){
    $admin = User::find(3);
    Laravel\be($admin)
        ->get(route( name: 'products.delete', ['product' => $this->product->id]))
        ->assertViewIs( value: 'admin.products.delete')
        ->assertSee($this->category->name)
        ->assertSee($this->product->name)
        ->assertSee($this->product->description)
        ->assertSee($this->product->latest_price->price)
        ->assertStatus( status: 200);
})->group( ...groups: 'ProductDelete');
```

Als we nu deze tests allemaal uitvoeren, zien we dat alles naar behoren werkt. De ProductDeleteTest is klaar.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductDelete

PASS Tests\Feature\Products\ProductDeleteTest
✓ guest can not see the product delete page
✓ customer can not see the product delete page
✓ sales can not see the product delete page
✓ admin can see the product delete page

Tests: 4 passed
Time: 3.24s
```

## 7.11. Product Destroy

Bij de destroy hebben we niet met validatie te maken. We kunnen dus nu alleen testen of de gebruiker het wel of niet mag. Hiervoor zullen we weer Json nodig hebben om de correcte status eruit te krijgen.

Als eerst maken we de test aan.

```
D:\Wamp\sites\laravelv1>php artisan pest:test Products/ProductDestroyTest
```

```
[OK] `tests/Feature/Products/ProductDestroyTest.php` created successfully.
```

En nu dan de opzet. Als eerst de customer nu gedaan. Ingelogd als customer, en daarna een json gestuurd met methode DELETE naar de goede route. Het resultaat zou dan moeten zijn dat deze gebruiker niet bij de route mag komen en dus een assertForbidden zou moeten krijgen.

```
use App\Models\User;
use App\Models\Category;
use App\Models\Product;
use App\Models\Price;
use \Pest\Laravel;

beforeEach(function (){
    $this->seed( class: 'RoleAndPermissionSeeder');
    $this->seed( class: 'UserSeeder');
    $this->category = Category::factory()->create();
    $this->product = Product::factory()->create();
    $this->price = price::factory()->create();
});

test( description: 'customer can not destroy a product', function () {
    $customer = User::find(1);
    Laravel\be($customer);
    $this->json( method: 'DELETE', route( name: 'products.destroy', ['product' => $this->product->id]))
        ->assertForbidden();
})->group( ...groups: 'ProductDestroy');
```

Bij het uitproberen van de test zien we dat het werkt.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductDestroy

  PASS  Tests\Feature\Products\ProductDestroyTest
    ✓ customer can not destroy a product

  Tests:  1 passed
  Time:   0.95s
```

Nu is de test voor de guest en sales erg makkelijk om te maken.

```
  test( description: 'sales can not destroy a product', function () {
    $sales = User::find(2);
    Laravel\be($sales);
    $this->json( method: 'DELETE', route( name: 'products.destroy', ['product' => $this->product->id]))
      ->assertForbidden();
  })->group( ...groups: 'ProductDestroy');

  test( description: 'guest can not destroy a product', function () {
    $this->json( method: 'DELETE', route( name: 'products.destroy', ['product' => $this->product->id]))
      ->assertStatus( status: 401);
  })->group( ...groups: 'ProductDestroy');
```

We zien ook dat de test ook werkt.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductDestroy

  PASS  Tests\Feature\Products\ProductDestroyTest
    ✓ customer can not destroy a product
    ✓ sales can not destroy a product
    ✓ guest can not destroy a product

  Tests:  3 passed
  Time:   2.33s
```

De admin zal net even wat anders zijn. We moeten namelijk wel controleren of het product ook werkelijk verwijderd is.

```
test( description: 'admin can destroy a product', function(){
    $admin = User::find(3);
    Laravel\be($admin);
    $this->json( method: 'DELETE', route( name: 'products.destroy', ['product' => $this->product->id]));
    $this->assertDatabaseMissing( table: 'products', ['id' => $this->product->id]);
})->group( ...groups: 'ProductDestroy');
```

Je zal zien, het product bestaat niet meer. We komen dus de test goed door.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductDestroy

PASS Tests\Feature\Products\ProductDestroyTest
✓ customer can not destroy a product
✓ sales can not destroy a product
✓ guest can not destroy a product
✓ admin can destroy a product

Tests: 4 passed
Time: 2.99s
```

Maar nu is de vraag, staan de prijzen van het product nog steeds in de database. Een extra controle dus:

```
test( description: 'admin can destroy a product', function(){
    $admin = User::find(3);
    Laravel\be($admin);
    $this->json( method: 'DELETE', route( name: 'products.destroy', ['product' => $this->product->id]));
    $this->assertDatabaseMissing( table: 'products', ['id' => $this->product->id]);
    $this->assertDatabaseMissing( table: 'prices', ['id' => $this->price->id]);
})->group( ...groups: 'ProductDestroy');
```

En ook hier komen we doorheen

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductDestroy

PASS Tests\Feature\Products\ProductDestroyTest
✓ customer can not destroy a product
✓ sales can not destroy a product
✓ guest can not destroy a product
✓ admin can destroy a product

Tests: 4 passed
Time: 3.17s
```

De prijzen gaan automatisch weg, omdat we in de relatie een cascade hebben aangegeven bij het aanmaken van de tabel. Je ziet dus hoe belangrijk de database is en dat je goed moet nadenken over de relaties in de database.

```
class CreatePricesTable extends Migration
{
    /**
     * Run the migrations. ....
     */
    public function up()
    {
        Schema::create('prices', function (Blueprint $table) {
            $table->id();
            $table->decimal('price', 8, 2);
            $table->dateTime('effdate');
            $table->foreignId('product_id')->constrained()
                ->onUpdate('cascade')->onDelete('cascade');
            $table->timestamps();
        });
    }
}
```

Dit betekent dat de gehele test voor de product crud klaar is. Toch nog even een check of alle testen nu nog goed werken. Ik laat alleen even het resultaat zien van het totaal, niet meer van de individuele testen.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest

  PASS Tests\Feature\AuthenticationTest
    ✓ login screen can be rendered
    ✓ users can authenticate using the login screen
    ✓ users can not authenticate with invalid password

  PASS Tests\Feature\Products\ProductUpdateTest
    ✓ customer can not update a product
    ✓ guest can not update a product
    ✓ admin can update a product
    ✓ sales can update a product

  Tests: 57 passed
  Time: 31.32s
```

Alle testen werken nu. Mooi, dit betekent wel dat je ondertussen weet hoe je om moet gaan met Feature Tests. We zijn vooral met Pest bezig geweest, maar ook even met phpunit. Alle testen zou je op beide manieren kunnen uitwerken.

## 7.12. Unit Testing

Nu we klaar zijn met de feature test kunnen we kijken wat we verder kunnen testen bij products. Dit zal met unit tests gaan gebeuren. Bij unit testing gaan we niet doen alsof we een gebruiker zijn en iets willen uitvoeren, maar juist of componenten die we gebruiken werken zoals we het verwachten.

Als eerst moeten we met Pest ook alle functionaliteiten kunnen gebruiken die we bij de feature tests beschikbaar hadden. Hiervoor moeten we in Pest.php aangeven dat we ook met 'Unit' willen werken.

```
uses( ...classAndTraits: Tests\TestCase::class, DatabaseMigrations::class) ->in( ...targets: 'Feature', 'Unit');
```

Een unit test kan gewoon gemaakt worden met Pest. Om een unit test voor product te maken, gaan we dus in de map *tests/Unit* het bestand *ProductTest.php* aanmaken. Omdat we de meeste testen al in feature hebben voor product en het niet al te uitgebreid is, kunnen we alle tests voor product in 1 test kwijt. Het aanmaken kan door middel van *php artisan pest:test ProductTest --unit*

```
D:\Wamp\sites\laravelv1>php artisan pest:test ProductTest --unit
[OK] `tests/Unit/ProductTest.php` created successfully.
```

Ook voor de unittest hebben we een basis setup nodig voor de test, want we hebben een categorie, product en prijs nodig. We hebben geen users of permissies nodig voor product. Er staat wel al bij dat we de class Carbon nodig hebben, dit ivm de created\_at en updated\_at.

```
use Carbon\Carbon;
use App\Models\Category;
use App\Models\Product;
use App\Models\Price;
use \Pest\Laravel;

beforeEach(function (){
    $this->category = Category::factory()->create();
    $this->product = Product::factory()->create();
    $this->price = Price::factory()->create();
});
```

Daarna gaan we kijken of de relaties kloppen zoals we verwachten. Een prijs is gekoppeld met een product en ook is product een onderdeel van een categorie. Dit kunnen we testen met assertInstanceOf(). Hierbij test je of de prijs een instantie is van product en dat product een instantie is van categorie. assertInstanceOf() is een functie binnen phpUnit.

```
test( description: 'a product has a price', function(){
    // $this->assertInstanceOf(Price::class, $this->product->latest_price); // PHPUnit Style
    expect($this->product->latest_price)->toBeInstanceOf( class: Price::class); // Pest expectations
})->group( ...groups: 'ProductUnit');
```

Zelf vind ik de manier van schrijven in Pest prettiger. Hier zeg je gewoon, ik verwacht ... om dit te zijn. Om ervoor te zorgen dat je beide kan zien, zal ik bij de unitests steeds beide voordoen. Ik voer wel steeds de Pest manier uit en de phpUnit manier zet ik in comments.

Als we dan controleren of product in een categorie zit

```
test( description: 'a product is inside a category', function(){
    // $this->assertInstanceOf(Category::class, $this->product->category); // PHPUnit Style
    expect($this->product->category)->toBeInstanceOf( class: Category::class); // Pest expectations
})->group( ...groups: 'ProductUnit');
```

Deze 2 unit testen gaan we uitproberen:

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductUnit

PASS Tests\Unit\ProductTest
✓ a product has a price
✓ a product is inside a category

Tests: 2 passed
Time: 1.41s
```

Zo te zien werken ze beide. We kunnen verder de attributen testen van product. Product heeft zelf 2 attributen naast de standaard binnen Laravel, namelijk name en description. We verwachten dat beide attributen een string zijn, wat we kunnen testen met assertIsString() ofwel toBeString()

```
test( description: 'a product name is a string', function(){
    // $this->assertIsString($this->product->name); // PHPUnit Style
    expect($this->product->name)->toBeString(); // Pest expectations
})->group( ...groups: 'ProductUnit');

test( description: 'a product description is a string', function(){
    // $this->assertIsString($this->product->description); // PHPUnit Style
    expect($this->product->description)->toBeString(); // Pest expectations
})->group( ...groups: 'ProductUnit');
```

We kunnen dan ook nog de standaard attributen testen. De id is natuurlijk een integer.

```
└ test( description: 'a product id is an int', function(){
    // $this->assertIsInt($this->product->id); // PHPUnit Style
    expect($this->product->id)->toBeInt(); // Pest expectations
})->group( ...groups: 'ProductUnit');

└ test( description: 'a product created at is a datetime', function(){
    // $this->assertInstanceOf(Carbon::class, $this->product->created_at); // PHPUnit Style
    expect($this->product->created_at)->toBeInstanceOf( class: Carbon::class); // Pest expectations
})->group( ...groups: 'ProductUnit');

└ test( description: 'a product updated at is a datetime', function(){
    // $this->assertInstanceOf(Carbon::class, $this->product->updated_at); // PHPUnit Style
    expect($this->product->updated_at)->toBeInstanceOf( class: Carbon::class); // Pest expectations
})->group( ...groups: 'ProductUnit');
```

Created\_at en updated\_at zijn wat lastiger. Er is namelijk niet een assertIsDateTime. Hiervoor moeten we dus controleren of het een onderdeel is van Carbon. Carbon wordt namelijk gebruikt om datetime aan te maken.

Als we nu de gehele unittest uitvoeren zien we dat alles goed is

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=ProductUnit

  PASS | Tests\Unit\ProductTest
  ✓ a product has a price
  ✓ a product is inside a category
  ✓ a product name is a string
  ✓ a product description is a string
  ✓ a product id is an int
  ✓ a product created at is a datetime
  ✓ a product updated at is a datetime

  Tests:  7 passed
  Time:   4.21s
```

De unit test is dus best snel klaar. Prijs zou een aparte unit test krijgen, omdat het nu echt over product gaat en we zijn niet afhankelijk wat er in een bepaalde methode gebeurd.

## 8.TDD met orders

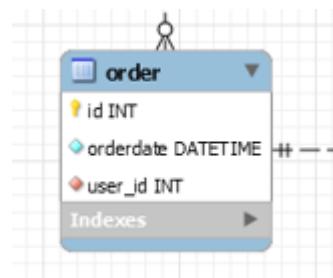
Nu we weten wat tests voor ons kunnen betekenen gaan we nog een stap verder, namelijk met Test Driven Development, ofwel TDD. Niet iedereen houd ervan, maar het zorgt er wel voor dat je heel gestructureerd gaat programmeren.

Met test driven development ga je eerst de tests schrijven voordat je de functionaliteit maakt. Waar we eerder juist tests schreven voor bestaande functionaliteit is bij TDD het omgekeerde. De moeilijkheid is wel dat je goed moet weten wat je wilt maken om van te voren je test te schrijven. Hiervoor gaan we dit dan ook stap voor stap doen. We pakken hierbij in ons project nog een makkelijk stukje, namelijk de order.

En je kan denken, de order is juist moeilijk!

Maar de order heeft de velden id, orderdate en user\_id.

We gaan dus niet meteen ervoor zorgen dat je producten in de order kan plaatsen etc, dat komt later wel. Wat we dus gaan doen is dat je een order kan plaatsen. Dit doen we wel voor een admin en nog niet voor een klant, zodat we het makkelijk in de site die we nu hebben kunnen plaatsen.

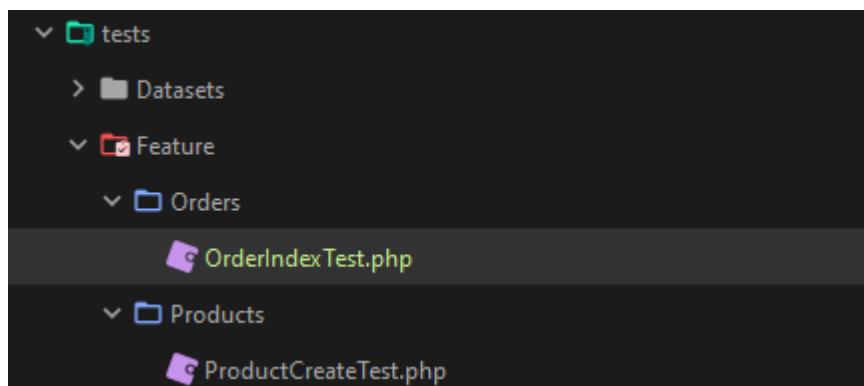


### 8.1. Maken van testen voor Order Index

We gaan simpel beginnen, namelijk met de test van de index. Hiervoor zullen we in het commando aangeven dat er een map orders moet komen in de feature map.

```
D:\Wamp\sites\laravelv1>php artisan pest:test Orders/OrderIndex
```

```
[OK] `tests/Feature/Orders/OrderIndex.php` created successfully.
```



De tests zijn redelijk vergelijkbaar met de test van de product index. Wel moeten we even nadenken wat we voor de index aangemaakt moeten hebben. We hebben users nodig en we hebben orders nodig.

Dus het eerste stuk van onze test wordt de beforeEach()

```
use App\Models\User;
use App\Models\Order;
use \Pest\Laravel;

beforeEach(function () {
    $this->seed( class: 'RoleAndPermissionSeeder');
    $this->seed( class: 'UserSeeder');
    $this->order = Order::factory()->create();
});

});
```

Ook al hebben we nog geen Order model, dit maakt op dit moment niet uit. Dat komt waarschijnlijk straks uit de test. Wat verwachten we verder op de index pagina te zien als bijvoorbeeld de admin. Bij de orders willen we de id, orderdatum en degene die de order heeft geplaatst zien.

```
test( description: 'admin can see the order index page', function()
{
    $this->withoutExceptionHandling();
    $admin = User::find(3);
    Laravel\be($admin)
        ->get(route( name: 'orders.index'))
        ->assertViewIs( value: 'admin.orders.index')
        ->assertSee($this->order->id)
        ->assertSee($this->order->orderdate)
        ->assertSee($this->order->user->name)
        ->assertStatus( status: 200);
})->group( ...groups: 'OrderIndex');
```

Je ziet hier al dat ik verwacht dat er named routes zullen zijn. Die zullen we dus straks ook moeten gaan maken. Verder zullen we de relatie tussen order en user gemaakt moeten hebben om de naam van de user straks te krijgen. De test is ook meteen in de groep OrderIndex gezet, zodat we deze apart kunnen testen. Verder moeten we de exception handling uit zetten om de errors die we krijgen altijd uit de test te krijgen. De test voor de sales zal gelijk zijn, omdat ze hetzelfde moeten kunnen zien.

```
test( description: 'sales can see the order index page', function()
{
    $this->withoutExceptionHandling();
    $sales = User::find(2);
    Laravel\be($sales)
        ->get(route( name: 'orders.index'))
        ->assertViewIs( value: 'admin.orders.index')
        ->assertSee($this->order->id)
        ->assertSee($this->order->orderdate)
        ->assertSee($this->order->user->name)
        ->assertStatus( status: 200);
})->group( ...groups: 'OrderIndex');
```

De customer mag het niet zien, dus krijg te zien dat hij geen permissies heeft. Iemand die niet is ingelogd krijgt een redirect naar de login. Bij deze 2 tests hebben we wel gewoon de exception handling nodig, want hier testen we juist op.

```
test( description: 'customer can not see the order index page', function()
{
    $customer = User::find(1);
    Laravel\be($customer)
        ->get(route( name: 'orders.index'))
        ->assertForbidden();
})->group( ...groups: 'OrderIndex');

test( description: 'guest can not see the order index page', function(){
    $this->get(route( name: 'orders.index'))
        ->assertRedirect(route( name: 'login'));
})->group( ...groups: 'OrderIndex');
```

Nu we deze tests hebben geschreven kunnen we de test uitvoeren. We gaan hier wel meteen de optie gebruiken dat als we een fout tegenkomen, de tests meteen gestopt worden. Dit zodat we eerst het probleem kunnen oplossen en daarna weer verder kunnen testen. Dit doen we met de optie --stop-on-failure

## 8.2. Starten met test van Order Index

Het resultaat van de test is dit. Zie ook meteen dat door de optie stop-on-failure er 1 failed staat en 3 pending.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderIndex --stop-on-failure

FAIL Tests\Feature\Orders\OrderIndexTest
  × admin can see the order index page

Error

Class "App\Models\Order" not found

at D:\Wamp\sites\laravelv1\tests\Feature\Orders\OrderIndexTest.php:10
  6 |
  7 | beforeEach(function () {
  8 |     $this->seed('RoleAndPermissionSeeder');
  9 |     $this->seed('UserSeeder');
→ 10 |     $this->order = Order::factory()->create();
  11 | });
  12 |
  13 | test('admin can see the order index page', function()
  14 | {



Tests: 1 failed, 3 pending
Time: 0.90s
```

Ok, de class App\Models\Order is er niet. Dit is de model van Order, die we dus nu gaan aanmaken.

```
D:\Wamp\sites\laravelv1>php artisan make:model Order  
Model created successfully.
```

Daarna voeren we de test weer uit. Doordat we steeds de errors moeten bekijken zal ik een klein stukje van de error laten zien. Juist net wat we nodig hebben.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderIndex --stop-on-failure  
  
FAIL Tests\Feature\Orders\OrderIndexTest  
  × admin can see the order index page  
  
Error  
  
Class "Database\Factories\OrderFactory" not found
```

Uit de test blijkt dat er geen factory is voor order. De factory gaan we dus aanmaken

```
D:\Wamp\sites\laravelv1>php artisan make:factory --model=Order OrderFactory  
Factory created successfully.
```

Factory is aangemaakt. Nu de test weer.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderIndex --stop-on-failure  
  
FAIL Tests\Feature\Orders\OrderIndexTest  
  × admin can see the order index page  
  
  Illuminate\Database\QueryException  
  Illuminate\Database\QueryException  
  
SQLSTATE[42S02]: Base table or view not found: 1146 Table 'laraveltest.orders' doesn't exist (SQL: insert into `orders`(`updated_at`, `created_at`) values (2021-04-21 19:39:16, 2021-04-21 19:39:16))
```

De tabel bestaat niet. Hiervoor moeten we een migration aanmaken en ook migrate uitvoeren.

```
D:\Wamp\sites\laravelv1>php artisan make:migration --create=orders create_orders_table  
Created Migration: 2021_04_21_194043_create_orders_table
```

Let even op, we maken het bestand aan maar veranderen nog niks aan de inhoud.

Voor de zekerheid meteen een migrate + seed. Niet nodig voor de test, maar wel voor live omgeving.

```
D:\Wamp\sites\laravelv1>php artisan migrate:fresh --seed
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (15.54ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (13.86ms)
```

Hierna weer de test uitvoeren

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderIndex --stop-on-failure

FAIL Tests\Feature\Orders\OrderIndexTest
  × admin can see the order index page

    Symfony\Component\Routing\Exception\RouteNotFoundException

    Route [orders.index] not defined.
```

Er is geen route voor orders. Voor de admin deden we dit met een resource route, dus deze gaan we wel toevoegen.

```
Route::resource('name: /admin/orders', 'OrderController');
```

Ik wil natuurlijk wel, omdat het een crud wordt, gewoon in de admin maps hebben. Niet alleen de controller, maar ook eigenlijk de url. Dit doen we maar meteen, want we gebruiken toch named routes. Verder zorgen we ook dat de class bij de include staat.

```
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\Admin\CategoryController;
use App\Http\Controllers\Admin\ProductController;
use App\Http\Controllers\Admin\OrderController;
```

De test weer

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderIndex --stop-on-failure

 FAIL Tests\Feature\Orders\OrderIndexTest
 × admin can see the order index page

 Illuminate\Contracts\Container\BindingResolutionException

 Target class [App\Http\Controllers\Admin\OrderController] does not exist.
```

De ordercontroller bestaat nog niet. Deze gaan we dan ook aanmaken.

```
D:\Wamp\sites\laravelv1>php artisan make:controller --model=Order Admin\OrderController
Controller created successfully.
```

De test weer

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderIndex --stop-on-failure

 FAIL Tests\Feature\Orders\OrderIndexTest
 × admin can see the order index page

 The response is not a view.
```

We moeten dus in de methode index aangeven dat een view getoond moet worden. Dit doen we in de controller.

```
class OrderController extends Controller
{
    /**
     * Display a listing of the resource. ....
     */
    public function index()
    {
        return view('admin.orders.index');
    }
}
```

De test weer

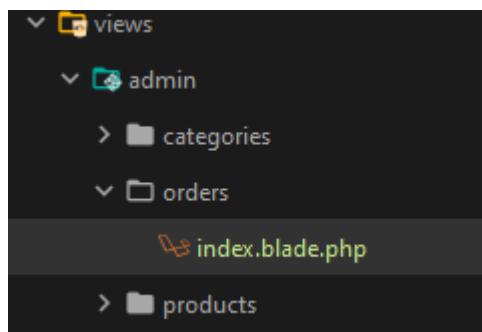
```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderIndex --stop-on-failure

FAIL Tests\Feature\Orders\OrderIndexTest
✖ admin can see the order index page

InvalidArgumentException

View [admin.orders.index] not found.
```

Er is geen view aangemaakt, dus deze moeten we in de map admin/orders gaan aanmaken.



De test weer

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderIndex --stop-on-failure

FAIL Tests\Feature\Orders\OrderIndexTest
✖ admin can see the order index page

Failed asserting that '' contains "1".
```

Er staat nu niet goed bij hoe of wat, maar als we verder kijken bij het testresultaat gaat het erom dat de order id niet te zien is.

```
at D:\Wamp\sites\laravelv1\tests\Feature\Orders\OrderIndexTest.php:19
15 |     $this->withoutExceptionHandling();
16 |     $admin = User::find(3);
17 |     Laravel\be($admin)
18 |         ->get(route('orders.index'))
→ 19 |         ->assertSee($this->order->id)
20 |         ->assertSee($this->order->orderdate)
21 |         ->assertSee($this->order->user->name)
22 |         ->assertStatus(200);
23 | })->group('OrderIndex');
```

De id zal 1 zijn, omdat we maar 1 order hebben. Dus hij ziet geen order id. Logisch natuurlijk, want we roepen wel in view aan, maar er wordt geen data opgehaald en naar de view gestuurd. In de view staat er niks om die data ook werkelijk te zien. Dit gaan we dan ook netjes maken. Nu kunnen we het zo maken.

```
class OrderController extends Controller
{
    /**
     * Display a listing of the resource. ...
     */
    public function index()
    {
        $orders = Order::all();
        return view('admin.orders.index', compact('orders'));
    }
}
```

De vraag is, moet dit ook zo. We moeten zo wel de username er ook bij krijgen, maar eager loading zal je niet uit een test kunnen halen. Dit is alleen maar dat het sneller wordt geladen. We gaan het dan maar meteen netjes doen:

```
class OrderController extends Controller
{
    /**
     * Display a listing of the resource. ...
     */
    public function index()
    {
        $orders = Order::with('user')->get();
        return view('admin.orders.index', compact('orders'));
    }
}
```

Als we opnieuw gaan testen krijgen we dit.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderIndex --stop-on-failure

FAIL Tests\Feature\Orders\OrderIndexTest
  × admin can see the order index page

    Illuminate\Database\Eloquent\RelationNotFoundException

    Call to undefined relationship [user] on model [App\Models\Order].
```

Logisch, want we hebben met eager loading verteld dat we met een relatie willen werken die user heet.

De relatie maken we dan ook netjes aan in de Order model.

```
class Order extends Model
{
    use HasFactory;

    public function user()
    {
        return $this->belongsTo( related: User::class);
    }
}
```

De volgende test geeft dan dit.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderIndex --stop-on-failure

FAIL Tests\Feature\Orders\OrderIndexTest
× admin can see the order index page

Failed asserting that '' contains "1".
```

Dit is nu dezelfde melding als eerst. Dezelfde fout kregen we namelijk terug toen we nog niet een ingevulde index methode hadden. Dit komt omdat de id nog steeds niet op de pagina staat. We zullen dus nu de view moeten gaan invullen zodat het er wel op komt te staan. Alleen even het tabelstuk hier, met alleen id die we laten zien.

```
<tbody class="bg-white divide-y divide-gray-200">
@foreach($orders as $order)
<tr>
    <td class="px-6 py-4 whitespace nowrap text-sm font-medium text-gray-900">
        {{ $order->id }}
    </td>
    <td class="px-6 py-4 whitespace nowrap text-sm font-medium text-gray-900">
    </td>
    <td class="px-6 py-4 whitespace nowrap text-sm font-medium text-gray-900">
    </td>
    <td class="px-6 py-4 whitespace nowrap text-sm text-gray-500">
        <a href="{{ route('orders.show', ['order' => $order->id]) }}">Details</a>
    </td>
    <td class="px-6 py-4 whitespace nowrap text-sm text-gray-500">
        <a href="{{ route('orders.edit', ['order' => $order->id]) }}">Edit</a>
    </td>
    @can('delete order')
        <td class="px-6 py-4 whitespace nowrap text-sm text-gray-500">
            <a href="{{ route('orders.delete', ['order' => $order->id]) }}">Delete</a>
        </td>
    @endcan
    </tr>
@endforeach
</tbody>
```

Bij de volgende test komt dit er uit.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderIndex --stop-on-failure

 FAIL Tests\Feature\Orders\OrderIndexTest
 × admin can see the order index page

ErrorException

Attempt to read property "name" on null

at D:\Wamp\sites\laravelv1\tests\Feature\Orders\OrderIndexTest.php:22
 18 |         ->get(route('orders.index'))
 19 |         ->assertViewIs('admin.orders.index')
 20 |         ->assertSee($this->order->id)
 21 |         ->assertSee($this->order->orderdate)
→ 22 |         ->assertSee($this->order->user->name)
 23 |         ->assertStatus(200);
 24 |     }->group('OrderIndex');
```

Nu is dit wat ingewikkelder. We zien in de foutmelding dat we nu name opvragen en dat het geen object is. We hebben wel in de model al de relatie gelegd, dus hier ligt het niet aan.

```
7  class Order extends Model
8  {
9      //
10     public function user()
11     {
12         return $this->belongsTo( related: User::class);
13     }
14 }
```

Nu hebben we natuurlijk de user\_id nog niet in de database staan. Hierdoor is er dus nog geen koppeling met de tabel users en kunnen we dus geen object ophalen. De migration gaan we dus wijzigen zodat de foreign key erin zit.

```
/** Run the migrations. ...*/
public function up()
{
    Schema::create('orders', function (Blueprint $table) {
        $table->id();
        $table->foreignId('user_id')->constrained()
            ->onUpdate('cascade')->onDelete('cascade');
        $table->timestamps();
    });
}
```

De volgende test geeft dit.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderIndex --stop-on-failure

 FAIL Tests\Feature\Orders\OrderIndexTest
 × admin can see the order index page

 Illuminate\Database\QueryException

SQLSTATE[HY000]: General error: 1364 Field 'user_id' doesn't have a default value (SQL: insert into `orders` (`updated_at`, `created_at`) values (2021-04-21 20:16:32, 2021-04-21 20:16:32))
```

Geen standaard waarde voor user\_id. Een standaard waarde geven we nu mee in de factory. In de factory zetten we dus een standaard waarde voor user\_id. Wel ervoor zorgen dat de class wordt ingeladen in de factory.

```
namespace Database\Factories;

use App\Models\Order;
use App\Models\User;
use Illuminate\Database\Eloquent\Factories\Factory;

class OrderFactory extends Factory
{
    /** The name of the factory's corresponding model. */
    protected $model = Order::class;

    /** Define the model's default state. ...*/
    public function definition()
    {
        return [
            'user_id' => User::all()->random()->id
        ];
    }
}
```

Als we nu testen krijgen we in de test

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderIndex --stop-on-failure

 FAIL Tests\Feature\Orders\OrderIndexTest
 × admin can see the order index page

 Failed asserting that '<!doctype html>\n<html lang="en">\n<head>\n
```

Er staat heel veel html in het resultaat van de test. Er wordt dan iets niet gevonden. Op het eind zien we dit.

```
\n
' contains "Customer".

at D:\Wamp\sites\laravelv1\tests\Feature\Orders\OrderIndexTest.php:22
18 |     ->get(route('orders.index'))
19 |     ->assertViewIs('admin.orders.index')
20 |     ->assertSee($this->order->id)
21 |     ->assertSee($this->order->orderdate)
→ 22 |     ->assertSee($this->order->user->name)
23 |     ->assertStatus(200);
24 | })->group('OrderIndex');
25 |
26 | test('sales can see the order index page', function()
```

Dit betekent dat de Customer, bij ons een naam van een user, niet op de pagina staat. Het kan zijn dat je wel door de test door komt, omdat de naam ook in het linkermenu staat voor degene die is ingelogd. Als dat dezelfde gebruiker is krijg je een andere fout. Voer gewoon is meerdere malen de test uit en je ziet een verschil. We zullen dus de index.blade moeten aanpassen zodat deze erop komt te staan.

```
@foreach($orders as $order)
    <tr>
        <td class="px-6 py-4 whitespace nowrap text-sm font-medium text-gray-900">
            {{ $order->id }}
        </td>
        <td class="px-6 py-4 whitespace nowrap text-sm font-medium text-gray-900">
            {{ $order->user->name }}
        </td>
```

Als we testen komt er dit uit.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderIndex --stop-on-failure

 FAIL Tests\Feature\Orders\OrderIndexTest
 ✓ admin can see the order index page
 ✓ sales can see the order index page
 ✘ customer can not see the order index page

 Response status code [200] is not a forbidden status code.
 Failed asserting that false is true.
```

Hij komt nu dus door de admin en sales test heen. Dit terwijl de orderdate nog niet in de database staat en ook niet in de layout. Wat er nu gebeurt is namelijk, dat \$this->order->orderdate gelijk is aan null. Als je dan assertSee(null) gebruikt, komt deze er altijd door, want het is gewoon een lege string. Helaas is dus onze test niet voldoende om nu verder te gaan. We hebben een unitest nodig zoals we met product hebben gedaan.

Als we even terug kijken naar de unit test van product stond dit erin.

```
31     test( description: 'a product created at is a datetime', function(){
32         $this->assertInstanceOf(Carbon::class, $this->category->created_at);
33     })->group( ...groups: 'ProductUnit');
34
35     test( description: 'a product updated at is a datetime', function(){
36         $this->assertInstanceOf(Carbon::class, $this->category->updated_at);
37     })->group( ...groups: 'ProductUnit');
```

Dit zou er dus voor zorgen dat de orderdate bij ons ook van dit type is, waardoor er geen null in kan komen te staan.

## 8.3. Order Unit Test

Bij de Unit tests maken we dan een OrderTest aan.

```
D:\Wamp\sites\laravelv1>php artisan pest:test OrderTest --unit  
[OK] `tests/Unit/OrderTest.php` created successfully.
```

In de OrderTest zullen we Carbon nodig hebben vanwege de datums. Verder hebben we een user nodig en een order voordat de testen beginnen.

```
1  <?php  
2  
3      use Carbon\Carbon;  
4  
5      beforeEach(function () {  
6          $this->user = factory( class: App\User::class)->create();  
7          $this->order = factory( class: App\Order::class)->create();  
8      });
```

Daarna kunnen we de attributen testen die we nodig hebben in order.

```
test( description: 'a order orderdate is a datetime', function () {  
    expect($this->order->orderdate)->toBeInstanceOf( class: Carbon::class);  
})->group( ...groups: 'OrderUnit');  
  
test( description: 'An order belongs to a user', function(){  
    expect($this->order->user)->toBeInstanceOf( class: User::class);  
})->group( ...groups: 'ProductUnit');  
  
test( description: 'a order user id is an int', function () {  
    expect($this->order->user_id)->toBeInt();  
})->group( ...groups: 'OrderUnit');  
  
test( description: 'a order id is an int', function () {  
    expect($this->order->id)->toBeInt();  
})->group( ...groups: 'OrderUnit');
```

De order date moet van Carbon zijn. Een order moet gekoppeld zijn aan een user, waardoor ook de foreign key een int moet zijn. De primary key is ook een int.

Natuurlijk hebben we ook altijd nog de created\_at en updated\_at

```
└─ test( description: 'a order created at is a datetime', function () {
    expect($this->order->created_at)->toBeInstanceOf( class: Carbon::class);
})->group( ...groups: 'OrderUnit');

└─ test( description: 'a order updated at is a datetime', function () {
    expect($this->order->updated_at)->toBeInstanceOf( class: Carbon::class);
})->group( ...groups: 'OrderUnit');
```

De order unit tests zijn nu in de groep OrderUnit gezet, zodat we deze samen met de feature test kunnen uitvoeren.

#### 8.4. Verder met TDD van Index

Nu we de unit test hebben, kunnen we deze samen met de OrderIndex test uitvoeren. Je ziet bij het commando nu, dat we meerdere groepen tegelijk kunnen testen met gewoon een komma ertussen.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderIndex,OrderUnit --stop-on-failure

FAIL Tests\Unit\OrderTest
× a order orderdate is a datetime

Failed asserting that null is an instance of class "Carbon\Carbon".
```

We krijgen meteen het gewenste resultaat, want we zien dat orderdate null is en geen instance van Carbon. Nu hebben we dus vanuit de test de reden gekregen om dit voor elkaar te krijgen. We gaan dus de standaardwaarde van orderdate als datetime zetten.

```
use App\Models\Order;
use App\Models\User;
use Carbon\Carbon;
use Illuminate\Database\Eloquent\Factories\Factory;

class OrderFactory extends Factory
{
    /** The name of the factory's corresponding model. ....*/
    protected $model = Order::class;

    /** Define the model's default state. ....*/
    public function definition()
    {
        return [
            'user_id' => User::all()->random()->id,
            'orderdate' => Carbon::now()
        ];
    }
}
```

Nu krijgen we met de test meteen de fout dat orderdate niet in de tabel staat

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderIndex,OrderUnit --stop-on-failure

FAIL Tests\Unit\OrderTest
✖ a order orderdate is a datetime

Illuminate\Database\QueryException

SQLSTATE[42S22]: Column not found: 1054 Unknown column 'orderdate' in 'field list' (SQL: insert into `orders` (`user_id`, `orderdate`, `updated_at`, `created_at`) values (1, 2021-04-22 07:09:23, 2021-04-22 07:09:23, 2021-04-22 07:09:23))
```

We gaan dus de migration aanpassen dat we orderdate in de tabel hebben

```
class CreateOrdersTable extends Migration
{
    /**
     * Run the migrations. ...
     */
    public function up()
    {
        Schema::create('orders', function (Blueprint $table) {
            $table->id();
            $table->dateTime('orderdate');
            $table->foreignId('user_id')->constrained()
                ->onUpdate('cascade')->onDelete('cascade');
            $table->timestamps();
        });
    }
}
```

Met de test krijgen we nu dat de unittest geheel werkt. Dus alle data is er met de correcte type.  
We krijgen wel een fout met de admin die de index page moet zien.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderIndex,OrderUnit --stop-on-failure

PASS Tests\Unit\OrderTest
✓ a order orderdate is a datetime
✓ a order user id is an int
✓ a order id is an int
✓ a order created at is a datetime
✓ a order updated at is a datetime

FAIL Tests\Feature\Orders\OrderIndexTest
✖ admin can see the order index page

Failed asserting that '<!doctype html>
<html lang="en">'
```

Hier komt weer een heel stuk html in voor, dus we missen iets in de view. Onderaan zie je wat we missen, namelijk de orderdate.

```
\n\n' contains "2021-04-22 07:11:13".\n\nat D:\Wamp\sites\laravelv1\tests\Feature\Orders\OrderIndexTest.php:21\n17 |     Laravel::be($admin)\n18 |         ->get(route('orders.index'))\n19 |         ->assertViewIs('admin.orders.index')\n20 |         ->assertSee($this->order->id)\n→ 21 |         ->assertSee($this->order->orderdate)\n22 |         ->assertSee($this->order->user->name)\n23 |         ->assertStatus(200);\n24 | })->group('OrderIndex');
```

Deze zetten we netjes in de foreach.

```
<tbody class="bg-white divide-y divide-gray-200">\n@foreach($orders as $order)\n    <tr>\n        <td class="px-6 py-4 whitespace nowrap text-sm font-medium text-gray-900">\n            {{ $order->id }}\n        </td>\n        <td class="px-6 py-4 whitespace nowrap text-sm font-medium text-gray-900">\n            {{ $order->user->name }}\n        </td>\n        <td class="px-6 py-4 whitespace nowrap text-sm font-medium text-gray-900">\n            {{ $order->orderdate }}\n        </td>\n    </tr>\n@endforeach
```

Als we weer testen krijgen we dit

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderIndex,OrderUnit --stop-on-failure

  PASS  Tests\Unit\OrderTest
    ✓ a order orderdate is a datetime
    ✓ a order user id is an int
    ✓ a order id is an int
    ✓ a order created at is a datetime
    ✓ a order updated at is a datetime

  FAIL  Tests\Feature\Orders\OrderIndexTest
    ✓ admin can see the order index page
    ✓ sales can see the order index page
    ✗ customer can not see the order index page

      Response status code [200] is not a forbidden status code.
      Failed asserting that false is true.
```

Je ziet dat alle unit testen goed gaan. De admin en sales gaan ook goed. Bij de customer krijgen we nu een status 200. De customer kan er nu dus bij. We hadden een forbidden verwacht. Dit betekent dus dat de permissies geregeld moeten worden voor de controller.

In de RoleAndPermissionSeeder gaan we als eerst permissies toevoegen. Omdat we nu echt heel precies moeten werken anders werkt TDD niet, zullen we de index ook moeten beveiligen in de controller.

```
// permissions for order CRUD
Permission::create(['name' => 'index order']);
Permission::create(['name' => 'show order']);
Permission::create(['name' => 'create order']);
Permission::create(['name' => 'edit order']);
Permission::create(['name' => 'delete order']);
```

Deze permissies geven we aan de sales

```
// sales role
$sales = Role::create(['name' => 'sales'])
    ->givePermissionTo(['index category', 'show category', 'create category', 'edit category',
        'index product', 'show product', 'create product', 'edit product',
        'index order', 'show order', 'create order', 'edit order']);
```

De admin heeft sowieso alle permissies, dus hoeven we niet apart te koppelen. Wel zullen we de permissies moeten koppelen aan de methodes in de controller.

```
class OrderController extends Controller
{
    /** Set permissions on methods ...*/
    public function __construct()
    {
        $this->middleware( middleware: 'auth');
        $this->middleware( middleware: 'permission:index order', ['only' => ['index']]);
        $this->middleware( middleware: 'permission:show order', ['only' => ['show']]);
        $this->middleware( middleware: 'permission:create order', ['only' => ['create', 'store']]);
        $this->middleware( middleware: 'permission:edit order', ['only' => ['edit', 'update']]);
        $this->middleware( middleware: 'permission:delete order', ['only' => ['delete', 'destroy']]);
    }
}
```

Als we de test nu uitvoeren zien we dit

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderIndex,OrderUnit --stop-on-failure
```

```
PASS Tests\Unit\OrderTest
✓ a order orderdate is a datetime
✓ a order user id is an int
✓ a order id is an int
✓ a order created at is a datetime
✓ a order updated at is a datetime

FAIL Tests\Feature\Orders\OrderIndexTest
✗ admin can see the order index page

Facade\Ignition\Exceptions\ViewException

Route [orders.delete] not defined. (View: D:\Wamp\sites\laravelv1\resources\views\admin\orders\index.blade.php)
```

We merken nu dat de route orders.delete nog niet bestaat. Dit klopt, want alleen de resource route was geregeld. Dit komt natuurlijk omdat in de view deze route al wordt gebruikt, want we gebruiken de methode zelf nog niet. Het aanpassing van de routes is dan zo:

```
Route::get( uri: 'admin/orders/{order}/delete', [OrderController::class, 'delete'])
    ->name( name: 'orders.delete');
Route::resource( name: '/admin/orders', controller: OrderController::class);
```

Nu kijken wat de test gaat doen.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderIndex,OrderUnit --stop-on-failure

  PASS  Tests\Unit\OrderTest
    ✓ a order orderdate is a datetime
    ✓ a order user id is an int
    ✓ a order id is an int
    ✓ a order created at is a datetime
    ✓ a order updated at is a datetime

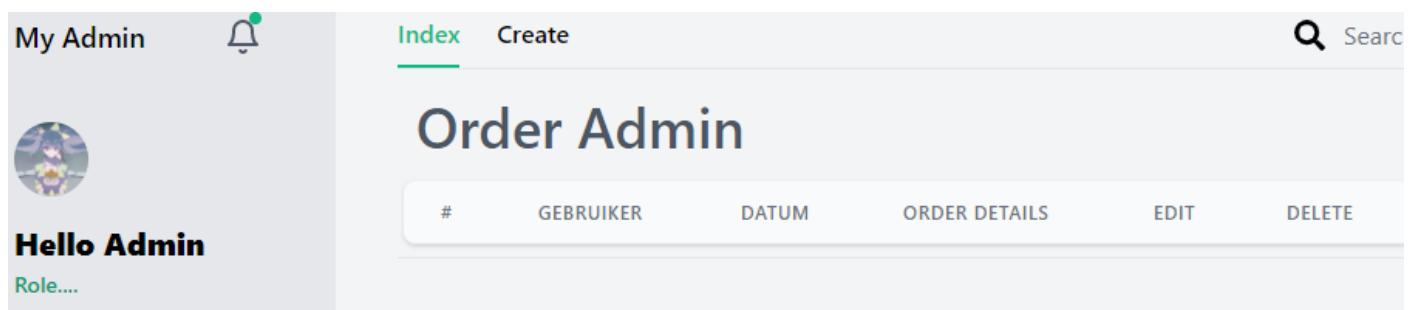
  PASS  Tests\Feature\Orders\OrderIndexTest
    ✓ admin can see the order index page
    ✓ sales can see the order index page
    ✓ customer can not see the order index page
    ✓ guest can not see the order index page

  Tests:  9 passed
  Time:   6.21s
```

Alles werkt zoals het zou moeten. Voordat we het in de browser gaan testen, zal je wel even je database moeten refreshen.

```
D:\Wamp\sites\laravelv1>php artisan migrate:fresh --seed
Dropped all tables successfully.
Migration table created successfully.
```

We kunnen nu testen, bijvoorbeeld als admin.



We zien een order admin zonder orders. Logisch, want we seeden niet standaard de order tabel nog, maar in de test deden we dat steeds per test. Als je niet bent ingelogd wordt je naar de login verwezen. Dit komt omdat we alle methodes hebben beveiligd in de controller.

We zijn er bijna, want het zou wel zo netjes zijn dat in de route al de controle wordt gedaan of je bent ingelogd. Dit kan je niet apart testen helaas. We gaan dus nog even de route aanpassen dat het in de group middleware komt te staan voor de rollen sales en admin.

```
Route::group(['middleware' => ['role:customer|sales|admin']], function () {
    Route::get('admin/categories/{category}/delete', [CategoryController::class, 'delete'])
        ->name('categories.delete');
    Route::resource('name: /admin/categories', controller: CategoryController::class);

    Route::get('uri: admin/products/{product}/delete', [ProductController::class, 'delete'])
        ->name('products.delete');
    Route::resource('name: /admin/products', controller: ProductController::class);

    Route::get('uri: admin/orders/{order}/delete', [OrderController::class, 'delete'])
        ->name('orders.delete');
    Route::resource('name: /admin/orders', controller: OrderController::class);
});
```

Nu nog even kijken of alles nog werkt.

```
D:\Wamp\sites\laravelv1>.\\vendor\\bin\\pest

PASS Tests\\Feature\\AuthenticationTest
✓ login screen can be rendered
✓ users can authenticate using the login screen
✓ users can not authenticate with invalid password
```

```
PASS Tests\\Feature\\Products\\ProductUpdateTest
✓ customer can not update a product
✓ guest can not update a product
✓ admin can update a product
✓ sales can update a product

Tests: 74 passed
Time: 45.49s
```

Door middel van Test Driven Development kan je dus heel goed een product maken en testen of bepaalde functionaliteit erin zit. Wel zal je heel nauwkeurig de tests moeten schrijven zodat je zeker weet dat je uiteindelijk het goede product maakt. Daar zit dan ook meteen de moeilijkheid in.

Als je eenmaal gewend bent om testen te schrijven zal je merken dat het steeds makkelijker af gaat.

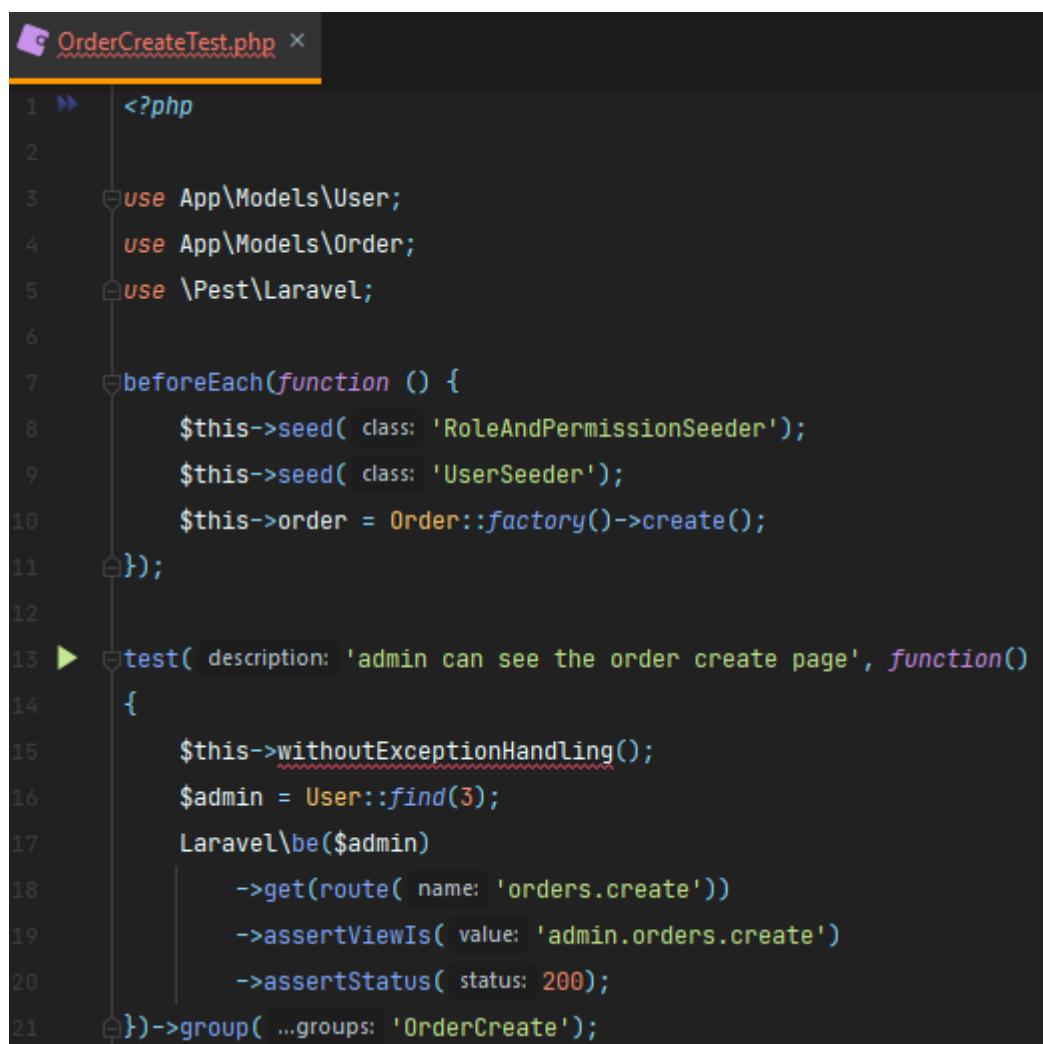
## 8.5. Create

Laten we kijken hoe TDD nu gaat bij de create van orders. Nu denk je, weer zoveel pagina's doorlopen. Maar er is nu al van alles in ons project beschikbaar. De aantal stappen die we moeten doorlopen zal daardoor nu veel minder zijn. Zeker omdat we al een resource route, resource controller en van alles in de database hebben geregeld. De permissions zijn ook meteen al gedaan.

Als eerst de test voor OrderCreate aanmaken.

```
D:\Wamp\sites\laravelv1>php artisan pest:test Orders\OrderCreateTest
[OK] `tests/Feature/Orders/OrderCreateTest.php` created successfully.
```

We hebben net als bij de index permissions, users en een order nodig. We gaan net als bij products alle rollen af om te checken of ze er wel of niet bij kunnen. Bij admin en sales wel exception handling uit zetten.



The screenshot shows a code editor window with the file 'OrderCreateTest.php' open. The code is a Pest test for the OrderCreateController. It includes imports for User, Order, and Pest, and a beforeEach block that seeds the database with RoleAndPermissionSeeder and UserSeeder, creates an order, and sets the current user to an admin. The test itself checks if an admin can see the order create page, using withoutExceptionHandling() to skip exception handling logic. It asserts the view is 'admin.orders.create', the status is 200, and groups the test under 'OrderCreate'.

```
<?php

use App\Models\User;
use App\Models\Order;
use \Pest\Laravel;

beforeEach(function () {
    $this->seed( class: 'RoleAndPermissionSeeder');
    $this->seed( class: 'UserSeeder');
    $this->order = Order::factory()->create();
});

test( description: 'admin can see the order create page', function()
{
    $this->withoutExceptionHandling();
    $admin = User::find(3);
    Laravel\be($admin)
        ->get(route( name: 'orders.create'))
        ->assertViewIs( value: 'admin.orders.create')
        ->assertStatus( status: 200);
})->group( ...groups: 'OrderCreate');
```

Bij een formulier hoef je niet te controleren of bepaalde waarden beschikbaar zijn, dus de create is wat simpeler dan de index.

Voor de sales is het natuurlijk hetzelfde

```
test( description: 'sales can see the order create page', function()
{
    $this->withoutExceptionHandling();
    $sales = User::find(2);
    Laravel\be($sales)
        ->get(route( name: 'orders.create'))
        ->assertViewIs( value: 'admin.orders.create')
        ->assertStatus( status: 200);
})->group( ...groups: 'OrderCreate');

test( description: 'customer can not see the order create page', function()
{
    $customer = User::find(1);
    Laravel\be($customer)
        ->get(route( name: 'orders.create'))
        ->assertForbidden();
})->group( ...groups: 'OrderCreate');

test( description: 'guest can not see the order create page', function(){
    $this->get(route( name: 'orders.create'))
        ->assertRedirect(route( name: 'login'));
})->group( ...groups: 'OrderCreate');
```

De customer en guest mogen er niet bij. Als we dan de test gaan doen, gebruiken we de OrderCreate, maar ook de OrderUnit. Dit even voor de zekerheid omdat we bij de index dit ook nodig hadden. Waarschijnlijk niet echt nodig bij de create, maar je kan beter teveel testen dan te weinig.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderCreate,OrderUnit --stop-on-failure

PASS Tests\Unit\OrderTest
✓ a order orderdate is a datetime
✓ a order user id is an int
✓ a order id is an int
✓ a order created at is a datetime
✓ a order updated at is a datetime

FAIL Tests\Feature\Orders\OrderCreateTest
✗ admin can see the order create page

The response is not a view.
```

De eerste melding is dat we geen view krijgen. Dit klopt, want de controller is nog leeg. We zullen daar een view moeten returnen.

```
/** Show the form for creating a new resource. ...*/
public function create()
{
    //
    return view( view: 'admin.orders.create');
}
```

Natuurlijk hebben we straks de users nodig, omdat deze aan een order gekoppeld zijn. Dit stuur ik nog niet mee. We zullen later hierin nog iets moeten gaan wijzigen. Opnieuw de test.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderUnit,OrderCreate --stop-on-failure

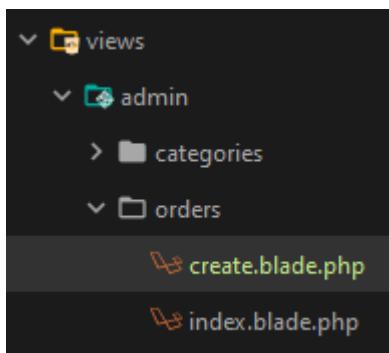
  PASS  Tests\Unit\OrderTest
    ✓ a order orderdate is a datetime
    ✓ a order user id is an int
    ✓ a order id is an int
    ✓ a order created at is a datetime
    ✓ a order updated at is a datetime

  FAIL  Tests\Feature\Orders\OrderCreateTest
    ✗ admin can see the order create page

      InvalidArgumentException

        View [admin.orders.create] not found.
```

De view bestaat nog niet, dus dit klopt. Deze maken we dus aan.



Als we daarna testen zien we dat alles werkt.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderCreate,OrderUnit --stop-on-failure

  PASS  Tests\Unit\OrderTest
    ✓ a order orderdate is a datetime
    ✓ a order user id is an int
    ✓ a order id is an int
    ✓ a order created at is a datetime
    ✓ a order updated at is a datetime

  FAIL  Tests\Feature\Orders\OrderCreateTest
    ✗ admin can see the order create page

      Failed asserting that '' contains "Customer".
```

We zien de customer niet. Dit hebben we in de test gezet om te checken of je een customer kan selecteren in het formulier. Hiervoor moeten we 2 stappen doen, namelijk de users ophalen en naar de view sturen. Daarna in de view laten zien.

```
/** Show the form for creating a new resource. ...*/
public function create()
{
    $users = User::all();
    return view( view: 'admin.orders.create', compact( var_name: 'users'));
}
```

Nu de users laten zien. Natuurlijk hebben we nog geen formulier in de create. We zouden in de test nog kunnen schrijven dat we bepaalde onderdelen willen zien als een POST ofzo. Maar als je ziet dat je een view moet maken voor een create snap je natuurlijk ook wel dat er wat op moet staan.

Zou je het verder willen testen, kan je een browsertest doen. Deze is er ook voor Laravel, namelijk met Laravel Dusk.

## # Introduction

Laravel Dusk provides an expressive, easy-to-use browser automation and testing API. By default, Dusk does not require you to install JDK or Selenium on your machine. Instead, Dusk uses a standalone [ChromeDriver](#) installation. However, you are free to utilize any other Selenium compatible driver you wish.

Dit gaan we alleen nu niet doen.

Wel even netjes het formulier gemaakt, want we willen natuurlijk wel dat de website ook gewoon straks ok is.

```
<form id="form" class="bg-white shadow-md rounded-lg px-8 pt-6 pb-8 mb-4"
      action="{{ route('orders.store') }}" method="POST">
    @csrf
    <div class="mb-4">
      <label class="block text-gray-700 text-sm font-bold mb-2" for="orderdate">
        Datum
      </label>
      <input
        class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
        focus:outline-none focus:shadow-outline @error('orderdate') border-red-500 @enderror" name="orderdate"
        id="orderdate" value="{{ old('orderdate') }}" type="datetime-local">
    </div>
    <div class="mb-4">
      <label class="block text-gray-700 text-sm font-bold mb-2" for="user_id">
        User
      </label>
      <select
        class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
        focus:outline-none focus:shadow-outline" name="user_id" id="user_id">
        @foreach($users as $user)
          <option value="{{ $user->id }}"
            @if( old('user_id') == $user->id)
              selected
            @endif
            >{{ $user->name }}</option>
        @endforeach
      </select>
    </div>
    <div class="flex items-center justify-between">
      <button id="submit"
        class="bg-green-500 hover:bg-green-700 text-white font-bold py-2 px-4 rounded
        focus:outline-none focus:shadow-outline" type="submit">Submit
      </button>
    </div>
  </form>
```

We zorgen dat we een datum kunnen invullen en een gebruiker kunnen selecteren voor een order. We koppelen nog geen producten.

Het mooie is, is dat als je een foutje maakt of iets vergeet, je dit mooi uit de test kan halen. Als we de test uitvoeren zie je dit:

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderCreate,OrderUnit --stop-on-failure

PASS Tests\Unit\OrderTest
✓ a order orderdate is a datetime
✓ a order user id is an int
✓ a order id is an int
✓ a order created at is a datetime
✓ a order updated at is a datetime

FAIL Tests\Feature\Orders\OrderCreateTest
✖ admin can see the order create page

Error

Class "App\Http\Controllers\Admin\User" not found

at D:\Wamp\sites\laravelv1\app\Http\Controllers\Admin\OrderController.php:42
38 |     * @return \Illuminate\Http\Response
39 |
40 |     public function create()
41 |
→ 42 |     $users = User::all();
43 |     return view('admin.orders.create', compact('users'));
44 |
45 |
46 |     /**
```

Blijkbaar in de create methode van de controller heb ik User gebruikt, maar niet in de use gezet bovenin. Die zetten we er dan nog even bij.

```
namespace App\Http\Controllers\Admin;

use App\Http\Controllers\Controller;
use App\Models\Order;
use App\Models\User;
use Illuminate\Http\Request;

class OrderController extends Controller
{
```

Als je nu test zie je dat alles is gelukt.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderCreate,OrderUnit --stop-on-failure

  PASS  Tests\Unit\OrderTest
    ✓ a order orderdate is a datetime
    ✓ a order user id is an int
    ✓ a order id is an int
    ✓ a order created at is a datetime
    ✓ a order updated at is a datetime

  PASS  Tests\Feature\Orders\OrderCreateTest
    ✓ admin can see the order create page
    ✓ sales can see the order create page
    ✓ customer can not see the order create page
    ✓ guest can not see the order create page

  Tests:  9 passed
  Time:   6.26s
```

De create is nu klaar. Merk je dat we niks meer voor permissies hoeft te regelen. Dit omdat we bij de index dit meteen voor de gehele crud hebben gedaan. Je zou per test het kunnen maken, zodat je per test steeds foutmeldingen krijgt en de aanpassingen moet doorvoeren. Dan zorg je er wel voor dat je geen onnodige code hebt geschreven. Voor nu, is het gewoon even makkelijk.

## 8.6. Store

Bij de store zullen we waarschijnlijk iets meer moeten doen, bijvoorbeeld met de validatie. Er is nog geheel geen validatie geschreven voor de store. Als eerst gaan de testen schrijven zodat we weer met TDD gestuurd worden naar het juiste resultaat.

Als eerst maken we een OrderStoreTest aan.

```
D:\Wamp\sites\laravelv1>php artisan pest:test Orders\OrderStoreTest
[OK] `tests/Feature/Orders/OrderStoreTest.php` created successfully.
```

Hierin zit natuurlijk een beforeEach en gaan we alle gebruikers af of ze er bij kunnen komen.

```
use App\Models\User;
use App\Models\Order;
use \Pest\Laravel;

beforeEach(function () {
    $this->seed( class: 'RoleAndPermissionSeeder');
    $this->seed( class: 'UserSeeder');
    $this->order = Order::factory()->create();
});

test( description: 'guest can not create an order in the admin', function () {
    $this->postJson(route( name: 'orders.store'))
        ->assertStatus( status: 401);
})->group( ...groups: 'OrderStore');

test( description: 'customer can not create an order in the admin', function () {
    $customer = User::find(1);
    Laravel\be($customer)
        ->postJson(route( name: 'orders.store'))
        ->assertForbidden();
})->group( ...groups: 'OrderStore');
```

Bij de niet ingelogde krijgen we als het goed is een status 401, dat ze geen bevoegdheid hebben.

Een customer mag er ook niet bij, maar krijgt een andere foutmelding. Namelijk dat hij geen rechten heeft om dit te doen.

Voor de admin en sales lijken ze op elkaar. Wel moeten we even de exceptionhandling weer uit zetten.

We posten een user, die ongelijk is aan waar we mee zijn ingelogd, en een orderdate als string naar store.

De user moet ongelijk zijn als degene waar we mee zijn ingelogd, anders wordt de user in het menu (rechtsboven op de site) meegenomen. De datum moeten we nog even omzetten naar een string, want zo slaan we het op in de database.

Verder testen we of er een redirect is naar de index en of de data in de database staat.

```
test( description: 'admin can create an order in the admin', function () {
    $admin = User::find(3);
    $order = Order::factory()->make(['user_id' => 1, 'orderdate' => Carbon::now()->toDateTimeString()]);

    Laravel\be($admin)
        ->postJson(route( name: 'orders.store'), $order->toArray())
        ->assertRedirect(route( name: 'orders.index'));

    $this->assertDatabaseHas( table: 'orders', [
        'user_id' => 1,
        'orderdate' => $order->orderdate
    ]);
})->group( ...groups: 'OrderStore');

test( description: 'sales can create an order in the admin', function () {
    $sales = User::find(2);
    $order = Order::factory()->make(['user_id' => 1, 'orderdate' => Carbon::now()->toDateTimeString()]);

    Laravel\be($sales)
        ->postJson(route( name: 'orders.store'), $order->toArray())
        ->assertRedirect(route( name: 'orders.index'));

    $this->assertDatabaseHas( table: 'orders', [
        'user_id' => 1,
        'orderdate' => $order->orderdate
    ]);
})->group( ...groups: 'OrderStore');
```

Het kan zijn dat je dan weer de index wil testen of de data er werkelijk staat. Of data er werkelijk staat test je in de index test. Je wilt de testen zo clean mogelijk houden, waardoor je niet op 2 plekken precies hetzelfde test. Je ziet duidelijk als je testen aan het schrijven bent, dat je wel goed het systeem moet kennen waardoor je weet wat er verwacht kan worden. Je zal naast deze tests altijd nog in de browser moeten kijken of het werkt, maar dat hoeft niet constant.

Natuurlijk moeten we ook voor de validatie testen schrijven. Voor product deden we dit met phpunit. Het zal nu een stuk kleiner worden, waardoor we geen lossen functie nodig hebben. We kunnen dus gewoon Pest gebruiken.

Dit doen we dan in de OrderStoreCheckTest

```
D:\Wamp\sites\laravelv1>php artisan pest:test Orders\OrderStoreCheckTest
[OK] `tests/Feature/Orders/OrderStoreCheckTest.php` created successfully.
```

In de test zorgen we ervoor dat alle (on)mogelijkheden worden uitgetest. Let er wel op, er worden ook al dingen in de unit test gedaan, dus ga niet dingen dubbel doen.

```
use App\Models\User;
use App\Models\Order;
use \Pest\Laravel;
use Carbon\Carbon;

beforeEach(function () {
    $this->seed( class: 'RoleAndPermissionSeeder');
    $this->seed( class: 'UserSeeder');
    $this->order = Order::factory()->create();
});

test( description: 'an order requires a user', function () {
    $admin = User::find(3);
    Laravel\be($admin);
    $order = Order::factory()->make(['user_id' => null, 'orderdate' => Carbon::now()->toDateTimeString()]);
    $this->postJson(route( name: 'orders.store'), $order->toArray())
        ->assertStatus( status: 422);
})->group( ...groups: 'OrderStoreCheck');
```

Bij de eerste test controleren we of we een 422 krijgen als we geen user\_id meesturen.

```
test( description: 'an user_id must be an integer', function () {
    $admin = User::find(3);
    Laravel\be($admin);
    $order = Order::factory()->make(['user_id' => 'bla', 'orderdate' => Carbon::now()->toDateTimeString()]);
    $this->postJson(route( name: 'orders.store'), $order->toArray())
        ->assertStatus( status: 422);
})->group( ...groups: 'OrderStoreCheck');
```

Ook controleren we of het gewoon iets random kan zijn.

Als laatst nog controleren of we de datum leeg kunnen laten

```
└ test( description: 'an order requires an orderdate', function () {
    $admin = User::find(3);
    Laravel\be($admin);
    $order = Order::factory()->make(['user_id' => 1, 'orderdate' => null]);
    $this->postJson(route('orders.store'), $order->toArray())
        ->assertStatus(status: 422);
})->group(...groups: 'OrderStoreCheck');
```

De tests voor OrderStore en OrderStoreCheck zijn nu geschreven. Als we nu beginnen met testen krijgen we als eerst dit

```
D:\Wamp\sites\laravelv1>.vendor\bin\pest --group=OrderStore,OrderStoreCheck,OrderUnit --stop-on-failure

PASS Tests\Unit\OrderTest
✓ a order orderdate is a datetime
✓ a order user id is an int
✓ a order id is an int
✓ a order created at is a datetime
✓ a order updated at is a datetime

FAIL Tests\Feature\Orders\OrderStoreCheckTest
✗ an order requires a user

Expected status code 422 but received 200.
Failed asserting that 422 is identical to 200.
```

We testen dus meteen de OrderStore en OrderStoreCheck, met daarbij altijd de OrderUnit test. We krijgen een 422 eruit, wat dus betekent dat de validatie niet goed gaat. Dit was natuurlijk al verwacht.

We maken dus met artisan de request aan

```
D:\Wamp\sites\laravelv1>php artisan make:request OrderStoreRequest
Request created successfully.
```

En vullen dan nu alleen in dat de user\_id verplicht is.

```
namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class OrderStoreRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request. ...
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request. ...
     */
    public function rules()
    {
        return [
            'user_id' => 'required'
        ];
    }
}
```

Ook moeten we ervoor zorgen dat deze request gebruikt gaat worden. In de methode store in de controller voegen we de request toe.

```
/** Store a newly created resource in storage. ...
public function store(OrderStoreRequest $request)
{
    //
}
```

Als we nu gaan testen gaat de order requires a user goed.

```
FAIL Tests\Feature\Orders\OrderStoreCheckTest
✓ an order requires a user
✗ an user_id must be an integer

Expected status code 422 but received 200.

Failed asserting that 422 is identical to 200.
```

Helaas werkt de volgende nog niet. Het moet ook een integer zijn. Dit zetten we dus netjes in de validatie

```
/** Get the validation rules that apply to the request. ...*/
public function rules()
{
    return [
        'user_id' => 'required|integer'
    ];
}
```

Als we nu testen komen we er wel doorheen. De volgende fout zit bij de create van een order.

```
FAIL Tests\Feature\Orders\OrderStoreCheckTest
✓ an order requires a user
✓ an user_id must be an integer
✗ an order requires an orderdate

Expected status code 422 but received 200.

Failed asserting that 422 is identical to 200.
```

De orderdate zetten we dan als required

```
/** Get the validation rules that apply to the request. ...*/
public function rules()
{
    return [
        'user_id' => 'required|integer',
        'orderdate' => 'required'
    ];
}
```

Dan gaan we weer testen en komt er dit uit.

```
PASS  Tests\Feature\Orders\OrderStoreCheckTest
✓ an order requires a user
✓ an user_id must be an integer
✓ an order requires an orderdate

FAIL  Tests\Feature\Orders\OrderStoreTest
✓ guest can not create an order in the admin
✓ customer can not create an order in the admin
✗ admin can create an order in the admin

Response status code [200] is not a redirect status code.
Failed asserting that false is true.
```

De OrderStoreCheckTest is nu geheel goed. Guest en customer ook al, maar de admin gaat nog niet goed. Er wordt een redirect verwacht maar we krijgen een melding dat alles goed is. We moeten dus een redirect toevoegen. Dit doen we in de controller.

```
/** Store a newly created resource in storage. ....*/
public function store(OrderStoreRequest $request)
{
    return redirect()->route('orders.index')->with('status', 'Order toegevoegd');
}
```

We voeren weer de test uit.

```
FAIL  Tests\Feature\Orders\OrderStoreTest
✓ guest can not create an order in the admin
✓ customer can not create an order in the admin
✗ admin can create an order in the admin

Failed asserting that a row in the table [orders] matches the attributes {
    "user_id": 1,
    "orderdate": "2021-04-22 20:54:43"
}.
```

We testen natuurlijk of deze in de database gaan komen. Dit hebben we nog niet in de controller dus zullen we dat moeten maken. De fout is dan ook, dat de order niet getoond wordt want het werd nog niet opgeslagen.

We passen dus de methode aan.

```
/** Store a newly created resource in storage. ...*/
public function store(OrderStoreRequest $request)
{
    $order = new Order();
    $order->user_id = $request->user_id;
    $order->orderdate = $request->orderdate;
    $order->save();

    return redirect()->route('orders.index')->with('status', 'Order toegevoegd');
}
```

Zodra we de methode dan zo maken dat het wordt opgeslagen. Nu weer de test.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderStore,OrderStoreCheck,OrderUnit --stop-on-failure

PASS Tests\Unit\OrderTest
✓ a order orderdate is a datetime
✓ a order user id is an int
✓ a order id is an int
✓ a order created at is a datetime
✓ a order updated at is a datetime

PASS Tests\Feature\Orders\OrderStoreCheckTest
✓ an order requires a user
✓ an user_id must be an integer
✓ an order requires an orderdate

PASS Tests\Feature\Orders\OrderStoreTest
✓ guest can not create an order in the admin
✓ customer can not create an order in the admin
✓ admin can create an order in the admin
✓ sales can create an order in the admin

Tests: 12 passed
Time: 8.59s
```

De gehele store werkt dus zoals we het willen, want alle testen zijn een succes. Als je het in de browser zelf gaat testen zal je zien dat alles ook netjes werkt.

## 8.7. Edit

De tests voor de edit zullen redelijk veel lijken op die van de create, met het verschil dat we data ophalen en die data dus willen zien. We maken de OrderEditTest aan.

```
D:\Wamp\sites\laravelv1>php artisan pest:test Orders\OrderEditTest
[OK] `tests/Feature/Orders/OrderEditTest.php` created successfully.
```

Het begin is niet al te lastig. Deze keer eerst even de guest en customer. Wel alvast Carbon ook ingeladen, want die hebben we zo nodig.

```
use Carbon\Carbon;
use App\Models\User;
use App\Models\Order;
use \Pest\Laravel;

beforeEach(function () {
    $this->seed( class: 'RoleAndPermissionSeeder');
    $this->seed( class: 'UserSeeder');
    $this->order = Order::factory()->create();
});

test( description: 'guest can not see the order edit page', function(){
    $this->get(route( name: 'orders.edit', ['order' => $this->order->id]))
        ->assertRedirect(route( name: 'login'));
})->group( ...groups: 'OrderEdit');

test( description: 'customer can not see the order edit page', function(){
    $customer = User::find(1);
    Laravel\be($customer)
        ->get(route( name: 'orders.edit', ['order' => $this->order->id]))
        ->assertForbidden();
})->group( ...groups: 'OrderEdit');
```

De sales en admin zijn wel wat lastiger

```
test( description: 'sales can see the order edit page', function(){
    $sales = User::find(2);
    $this->withoutExceptionHandling();

    Laravel\be($sales)
        ->get(route( name: 'orders.edit', ['order' => $this->order->id]))
        ->assertViewIs( value: 'admin.orders.edit')
        ->assertSee($this->order->user->name)
        ->assertSee(Carbon::createFromFormat( format: 'Y-m-d H:i:s', $this->order->orderdate)
            ->format( format: 'Y-m-d\TH:i'))
        ->assertStatus( status: 200);
})->group( ...groups: 'OrderEdit');

test( description: 'admin can see the order edit page', function(){
    $admin = User::find(3);
    $this->withoutExceptionHandling();

    Laravel\be($admin)
        ->get(route( name: 'orders.edit', ['order' => $this->order->id]))
        ->assertViewIs( value: 'admin.orders.edit')
        ->assertSee($this->order->user->name)
        ->assertSee(Carbon::createFromFormat( format: 'Y-m-d H:i:s', $this->order->orderdate)
            ->format( format: 'Y-m-d\TH:i'))
        ->assertStatus( status: 200);
})->group( ...groups: 'OrderEdit');
```

Je ziet hier dat we inloggen als de gebruiker, naar de edit gaan (met de order id mee). Checken of we de goede view zien met daarom de username, de orderdatum al ingevuld en status 200 zodat we zien dat de pagina goed is ingeladen. Als we nu beginnen met testen met OrderEdit en OrderUnit zien we dit:

```
D:\Wamp\sites\laravelv1.\vendor\bin\pest --group=OrderEdit,OrderUnit --stop-on-failure

PASS Tests\Unit\OrderTest
✓ a order orderdate is a datetime
✓ a order user id is an int
✓ a order id is an int
✓ a order created at is a datetime
✓ a order updated at is a datetime

FAIL Tests\Feature\Orders\OrderEditTest
✓ guest can not see the order edit page
✓ customer can not see the order edit page
✗ sales can see the order edit page

The response is not a view.
```

De unit test blijft natuurlijk goed gaan. De guest en customer mogen niet bij de pagina, wat klopt. Daarna de sales moet de edit page zien. We roepen geen view aan in de methode, dus de melding klopt. Dit vullen we in bij de edit methode

```
/** Show the form for editing the specified resource. ...*/
public function edit(Order $order)
{
    return view( view: 'admin.orders.edit');
}
```

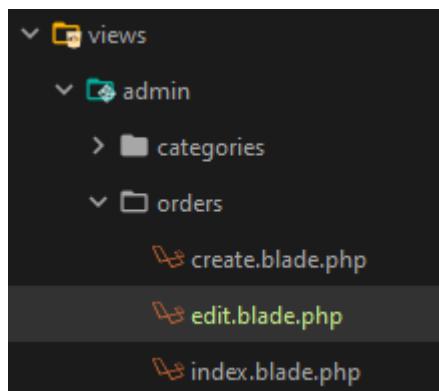
De volgende test

```
FAIL Tests\Feature\Orders\OrderEditTest
✓ guest can not see the order edit page
✓ customer can not see the order edit page
✗ sales can see the order edit page

InvalidArgumentException

View [admin.orders.edit] not found.
```

De view is niet gevonden. Deze maken we dus aan.



Als we weer testen

```
FAIL Tests\Feature\Orders\OrderEditTest
✓ guest can not see the order edit page
✓ customer can not see the order edit page
✗ sales can see the order edit page

Failed asserting that '' contains "Admin".
```

De gebruikersnaam staat er dus niet op. Dit klopt, want we moeten een dropdown hebben met users, net zoals in de create. Deze gegevens sturen we dus nog niet mee. De view is nog geheel leeg ook, dus hier moeten we wat mee gaan doen. De lege quotes geven ook aan dat er helemaal niks in de view staat (we hebben eerder al veel html gezien). We maken netjes de gehele edit.blade

```
<form id="form" class="bg-white shadow-md rounded-lg px-8 pt-6 pb-8 mb-4"
      action="{{ route('orders.update', ['order' => $order->id]) }}" method="POST">
    @method('PUT')
    @csrf
    <div class="mb-4">
      <label class="block text-gray-700 text-sm font-bold mb-2" for="orderdate">
        Datum
      </label>
      <input
        class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
        focus:outline-none focus:shadow-outline @error('orderdate') border-red-500 @enderror" name="orderdate"
        id="orderdate" value="{{ old('orderdate', $order->orderdate) }}" type="datetime-local">
    </div>
    <div class="mb-4">
      <label class="block text-gray-700 text-sm font-bold mb-2" for="user_id">
        User
      </label>
      <select
        class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
        focus:outline-none focus:shadow-outline" name="user_id" id="user_id">
        @foreach($users as $user)
          <option value="{{ $user->id }}"
                 @if( old('user_id', $order->user_id) == $user->id)
                   selected
                 @endif
               >{{ $user->name }}</option>
        @endforeach
      </select>
    </div>
    <div class="flex items-center justify-between">
      <button id="submit"
             class="bg-green-500 hover:bg-green-700 text-white font-bold py-2 px-4 rounded
             focus:outline-none focus:shadow-outline" type="submit">Submit
      </button>
    </div>
  </form>
```

Als we opnieuw gaan testen zien we dit

```
FAIL Tests\Feature\Orders\OrderEditTest
✓ guest can not see the order edit page
✓ customer can not see the order edit page
✗ sales can see the order edit page
```

```
Facade\Ignition\Exceptions\ViewException
```

```
Undefined variable $order (View: D:\Wamp\sites\laravelv1\resources\views\admin\orders\edit.blade.php)
```

De orders sturen we nog niet mee, dus dit wijzigen we in de edit methode.

```
/** Show the form for editing the specified resource. ...*/
public function edit(Order $order)
{
    return view( view: 'admin.orders.edit', compact( var_name: 'order'));
}
```

En opnieuw de test.

```
FAIL Tests\Feature\Orders\OrderEditTest
✓ guest can not see the order edit page
✓ customer can not see the order edit page
✗ sales can see the order edit page
```

```
Facade\Ignition\Exceptions\ViewException
```

```
Undefined variable $users (View: D:\Wamp\sites\laravelv1\resources\views\admin\orders\edit.blade.php)
```

De users voor de select sturen we nog niet op. In de methode moeten we ze dus ophalen en meesturen naar de view.

```
/** Show the form for editing the specified resource. ...*/
public function edit(Order $order)
{
    $users = User::all();
    return view( view: 'admin.orders.edit', compact( var_name: 'order', ...var_names: 'users'));
}
```

Als we nu kijken naar de test

```
FAIL Tests\Feature\Orders\OrderEditTest
✓ guest can not see the order edit page
✓ customer can not see the order edit page
✗ sales can see the order edit page

Failed asserting that '<!doctype html>\n<html lang="en">\n<head>\n
```

Een heel stuk html weer. Als we dan helemaal ondernaan bij de test kijken zien we waar het probleem zit.

```
' contains "2021-04-23T11:46".

at D:\Wamp\sites\laravelv1\tests\Feature\Orders\OrderEditTest.php:35
31 |         ->get(route('orders.edit', ['order' => $this->order->id]))
32 |         ->assertViewIs('admin.orders.edit')
33 |         ->assertSee($this->order->user->name)
34 |         ->assertSee(Carbon::createFromFormat('Y-m-d H:i:s', $this->order->orderdate))
→ 35 |             ->format('Y-m-d\TH:i'))
36 |         ->assertStatus(200);
37 | })->group('OrderEdit');
```

De datum wordt niet goed in het formulier weergegeven. Dit komt, omdat de format van een formulier en wat er in de database komt niet hetzelfde is. We zullen dus de format moeten wijzigen voordat het naar de view gaat.

```
/** Show the form for editing the specified resource. ...*/
public function edit(Order $order)
{
    $users = User::all();
    $order->orderdate = Carbon::createFromFormat( format: 'Y-m-d H:i:s', $order->orderdate)
        ->format( format: 'Y-m-d\TH:i');
    return view( view: 'admin.orders.edit', compact( var_name: 'order', ...var_names: 'users'));
}
```

Wat we doen is, we krijgen een datum in de format Y-m-d H:i:s en hier maken we een Carbon object van. Deze gaan we formatten met Y-m-d\TH:i

De \ wordt gebruikt als escape, zodat de T als letter blijft staan. Verder hebben we geen seconden nodig in het formulier, dus laten we dit weg.

Als we nu testen zien we dit:

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderEdit,OrderUnit --stop-on-failure

  PASS  Tests\Unit\OrderTest
    ✓ a order orderdate is a datetime
    ✓ a order user id is an int
    ✓ a order id is an int
    ✓ a order created at is a datetime
    ✓ a order updated at is a datetime

  PASS  Tests\Feature\Orders\OrderEditTest
    ✓ guest can not see the order edit page
    ✓ customer can not see the order edit page
    ✓ sales can see the order edit page
    ✓ admin can see the order edit page

  Tests:  9 passed
  Time:   6.21s
```

De test is klaar. Als we het uittesten, kunnen we even een order toevoegen en daarna naar de edit gaan.

## Order Admin

Datum

User

**Submit**

We zien netjes een datum + tijd, en de geselecteerde user.

## 8.8. Update

Voor de update gaan we weer de testen schrijven. Dit doen we in OrderUpdateTest om de functionaliteit te testen en in OrderUpdateCheckTest om de validatie te controleren. Als eerst de OrderUpdateTest maken.

```
D:\Wamp\sites\laravelv1>php artisan pest:test Orders\OrderUpdateTest
[OK] `tests/Feature/Orders/OrderUpdateTest.php` created successfully.
```

In de OrderUpdateTest komt dan dit.

```
use Carbon\Carbon;
use App\Models\User;
use App\Models\Order;
use \Pest\Laravel;

beforeEach(function () {
    $this->seed( class: 'RoleAndPermissionSeeder');
    $this->seed( class: 'UserSeeder');
    $this->order = Order::factory()->create();
});

test( description: 'guest can not update an order in the admin', function () {
    $this->patchJson(route( name: 'orders.update', [ 'order' => $this->order->id]))
        ->assertStatus( status: 401);
})->group( ...groups: 'OrderUpdate');

test( description: 'customer can not update an order in the admin', function () {
    $customer = User::find(1);
    Laravel\be($customer)
        ->patchJson(route( name: 'orders.update', [ 'order' => $this->order->id]))
        ->assertForbidden();
})->group( ...groups: 'OrderUpdate');
```

De guest en customer hebben geen mogelijkheid om erbij te komen. Hier veranderd weinig aan.

Voor de sales en admin zijn de testen wat uitgebreider.

```
test( description: 'admin can update an order in the admin', function () {
    $this->withoutExceptionHandling();
    $admin = User::find(3);
    Laravel\be($admin);
    $order = Order::factory()->make(['user_id' => 1, 'orderdate' => Carbon::now()->toDateTimeString()]);
    $newdate = Carbon::now()->addDays( value: 10)->toDateTimeString();
    $this->patchJson(route( name: 'orders.update', ['order' => $this->order->id]),
        ['user_id' => 2, 'orderdate' => $newdate])->assertRedirect(route( name: 'orders.index'));

    $this->assertDatabaseHas( table: 'orders', ['user_id' => 2, 'orderdate' => $newdate]);
})->group( ...groups: 'OrderUpdate');

test( description: 'sales can update an order in the admin', function () {
    $this->withoutExceptionHandling();
    $sales = User::find(2);
    Laravel\be($sales);
    $order = Order::factory()->make(['user_id' => 1, 'orderdate' => Carbon::now()->toDateTimeString()]);
    $newdate = Carbon::now()->addDays( value: 10)->toDateTimeString();
    $this->patchJson(route( name: 'orders.update', ['order' => $this->order->id]),
        ['user_id' => 2, 'orderdate' => $newdate])->assertRedirect(route( name: 'orders.index'));

    $this->assertDatabaseHas( table: 'orders', ['user_id' => 2, 'orderdate' => $newdate]);
})->group( ...groups: 'OrderUpdate');
```

Let op dat je wel de user en orderdate aanpast. We kijken of we een redirect krijgen en dat in de database de nieuwe gegevens staan. We maken ook meteen de test aan voor de OrderUpdateCheckTest, waar we testen of de validatie correct is.

```
D:\Wamp\sites\laravelv1>php artisan pest:test Orders\OrderUpdateCheckTest
[OK] `tests/Feature/Orders/OrderUpdateCheckTest.php` created successfully.
```

Bij deze test ook het standaard begin

```
use Carbon\Carbon;
use App\Models\User;
use App\Models\Order;
use \Pest\Laravel;

beforeEach(function () {
    $this->seed( class: 'RoleAndPermissionSeeder');
    $this->seed( class: 'UserSeeder');
    $this->order = Order::factory()->create();
});
```

Daarna de testen, waarbij user\_id = null, user\_id = bla en orderdate = null wordt uitgeprobeerd. Er wordt dan wel verwacht dat we een status 422 krijgen.

```
test( description: 'an order update requires an user', function () {
    $admin = User::find(3);
    $this->actingAs($admin);
    $order = Order::factory()->make(['user_id' => 1, 'orderdate' => Carbon::now()->toDateTimeString()]);
    $this->patchJson(route('orders.update', ['order' => $this->order->id]),
        ['user_id' => null, 'orderdate' => $order->orderdate])->assertStatus(status: 422);
})->group(...groups: 'OrderUpdateCheck');

test( description: 'an user id must be an integer', function () {
    $admin = User::find(3);
    $this->actingAs($admin);
    $order = Order::factory()->make(['user_id' => 1, 'orderdate' => Carbon::now()->toDateTimeString()]);
    $this->patchJson(route('orders.update', ['order' => $this->order->id]),
        ['user_id' => 'bla', 'orderdate' => $order->orderdate])->assertStatus(status: 422);
})->group(...groups: 'OrderUpdateCheck');

test( description: 'an order requires a orderdate', function () {
    $admin = User::find(3);
    $this->actingAs($admin);
    $order = Order::factory()->make(['user_id' => 1, 'orderdate' => Carbon::now()->toDateTimeString()]);
    $this->patchJson(route('orders.update', ['order' => $this->order->id]),
        ['user_id' => 2, 'orderdate' => null])->assertStatus(status: 422);
})->group(...groups: 'OrderUpdateCheck');
```

Als we gaan testen met OrderUpdate, OrderCheckUpdate en OrderUnit krijgen we dit:

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderUpdate,OrderUpdateCheck,OrderUnit --stop-on-failure

PASS Tests\Unit\OrderTest
✓ a order orderdate is a datetime
✓ a order user id is an int
✓ a order id is an int
✓ a order created at is a datetime
✓ a order updated at is a datetime

FAIL Tests\Feature\Orders\OrderUpdateCheckTest
✗ an order update requires an user

Expected status code 422 but received 200.
Failed asserting that 422 is identical to 200.
```

Hij voert de check test uit, want die staat boven de OrderUpdateTest. We krijgen een ok bij deze test, terwijl we een 442 hadden verwacht. Dit betekent dat er gewoon geen validatie wordt gedaan. Dit klopt ook, want de Update heeft nog geen validatie. Dit veranderen we in de controller. We kunnen dezelfde validatie gebruiken als bij de store.

```
/** Update the specified resource in storage. ...*/
public function update(OrderStoreRequest $request, Order $order)
{
    //
}
```

Als we dan weer testen zien we dit

```
PASS Tests\Feature\Orders\OrderUpdateCheckTest
✓ an order update requires an user
✓ an user id must be an integer
✓ an order requires a orderdate

FAIL Tests\Feature\Orders\OrderUpdateTest
✓ guest can not update an order in the admin
✓ customer can not update an order in the admin
✗ admin can update an order in the admin

Response status code [200] is not a redirect status code.
Failed asserting that false is true.
```

De check testen gaan al goed. Mooi, validatie werkt. Bij de admin krijgen we geen redirect eruit. Dat klopt ook, want we doen nog niks bij de update. We zorgen dus dat we een redirect erin zetten.

```
/** Update the specified resource in storage. ...*/
public function update(OrderStoreRequest $request, Order $order)
{
    return redirect()->route('orders.index')->with(['status' => 'Order gewijzigd']);
}
```

Bij de volgende test zien we dit

```
FAIL Tests\Feature\Orders\OrderUpdateTest
✓ guest can not update an order in the admin
✓ customer can not update an order in the admin
✗ admin can update an order in the admin

Failed asserting that a row in the table [orders] matches the attributes {
    "user_id": 2,
    "orderdate": "2021-05-03 12:59:13"
}.
```

De gegevens die we verwachten staan niet in de database. Dit klopt, want we wijzigen ook nog niks in de controller. We zorgen in de update dat we dus de gegevens wijzigen.

```
/** Update the specified resource in storage. ...*/
public function update(OrderStoreRequest $request, Order $order)
{
    $order->user_id = $request->user_id;
    $order->orderdate = $request->orderdate;
    $order->save();

    return redirect()->route('orders.index')->with('status', 'Order gewijzigd');
}
```

We zijn dan ook meteen klaar met de update, want alle testen gaan nu goed

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderUpdate,OrderUpdateCheck,OrderUnit --stop-on-failure

PASS Tests\Unit\OrderTest
✓ a order orderdate is a datetime
✓ a order user id is an int
✓ a order id is an int
✓ a order created at is a datetime
✓ a order updated at is a datetime

PASS Tests\Feature\Orders\OrderUpdateCheckTest
✓ an order update requires an user
✓ an user id must be an integer
✓ an order requires a orderdate

PASS Tests\Feature\Orders\OrderUpdateTest
✓ guest can not update an order in the admin
✓ customer can not update an order in the admin
✓ admin can update an order in the admin
✓ sales can update an order in the admin

Tests: 12 passed
Time: 8.56s
```

## 8.9. Delete

De delete is eigenlijk erg simpel. We willen exact hetzelfde zien als bij de edit, alleen dan in de route van delete. Dat iets disabled is zou meer voor een browsertest zijn. Dit doen we dan ook niet. Verder mag alleen de admin deleten, dus sales mag er niet bij komen. Eerst maken we de OrderDeleteTest aan.

```
D:\Wamp\sites\laravelv1>php artisan pest:test Orders\OrderDeleteTest
```

```
[OK] `tests/Feature/Orders/OrderDeleteTest.php` created successfully.
```

De OrderDeleteTest ziet er dan zo uit. Customer en Sales zijn nu dezelfde test alleen met andere user.

```
use Carbon\Carbon;
use App\Models\User;
use App\Models\Order;
use \Pest\Laravel;

beforeEach(function () {
    $this->seed( class: 'RoleAndPermissionSeeder');
    $this->seed( class: 'UserSeeder');
    $this->order = Order::factory()->create();
});

test( description: 'guest can not see the order delete page', function(){
    $this->get(route( name: 'orders.delete', ['order' => $this->order->id]))
        ->assertRedirect(route( name: 'login'));
})->group( ...groups: 'OrderDelete');

test( description: 'customer can not see the order delete page', function(){
    $customer = User::find(1);
    Laravel\be($customer)
        ->get(route( name: 'orders.delete', ['order' => $this->order->id]))
        ->assertForbidden();
})->group( ...groups: 'OrderDelete');

test( description: 'sales can not see the order delete page', function(){
    $sales = User::find(2);
    Laravel\be($sales)
        ->get(route( name: 'orders.delete', ['order' => $this->order->id]))
        ->assertForbidden();
})->group( ...groups: 'OrderDelete');
```

De admin test is wel natuurlijk wat groter, maar we gebruiken hetzelfde als bij de OrderEditTest. We zien een formulier met ingevulde gegevens.

```
test( description: 'admin can see the order delete page', function(){
    $admin = User::find(3);
    $this->withoutExceptionHandling();

    Laravel\be($admin)
        ->get(route( name: 'orders.delete', ['order' => $this->order->id]))
        ->assertViewIs( value: 'admin.orders.delete')
        ->assertSee($this->order->user->name)
        ->assertSee(Carbon::createFromFormat( format: 'Y-m-d H:i:s', $this->order->orderdate)
            ->format( format: 'Y-m-d\TH:i'))
        ->assertStatus( status: 200);
})->group( ...groups: 'OrderDelete');
```

Als we de test nu gaan uitvoeren krijgen we dit. De UnitTest heeft bij de delete niet echt veel zin, dus voeren we nu niet uit.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderDelete --stop-on-failure

 FAIL Tests\Feature\Orders\OrderDeleteTest
 ✓ guest can not see the order delete page
 ✓ customer can not see the order delete page
 ✓ sales can not see the order delete page
 ✘ admin can see the order delete page

 ---
 
 • Tests\Feature\Orders\OrderDeleteTest > admin can see the order delete page
 BadMethodCallException

 Method App\Http\Controllers\Admin\OrderController::delete does not exist.
```

De methode bestaat niet in de controller. Deze gaan we dus aanmaken.

De delete methode komt weer tussen de update en destroy.

```
/**  
 * Show the form for deleting the specified resource.  
 *  
 * @param \App\Models\Order $order  
 * @return \Illuminate\Http\Response  
 */  
  
public function delete(Order $order)  
{  
}
```

Als we testen krijgen we dit

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderDelete --stop-on-failure  
  
 FAIL Tests\Feature\Orders\OrderDeleteTest  
✓ guest can not see the order delete page  
✓ customer can not see the order delete page  
✓ sales can not see the order delete page  
✗ admin can see the order delete page  
  
---  
  
• Tests\Feature\Orders\OrderDeleteTest > admin can see the order delete page  
The response is not a view.
```

In de controller moeten we dus een view returnen. Dit doen we in de controller.

```
public function delete(Order $order)  
{  
    return view('admin.orders.delete');  
}
```

Daarna testen we weer.

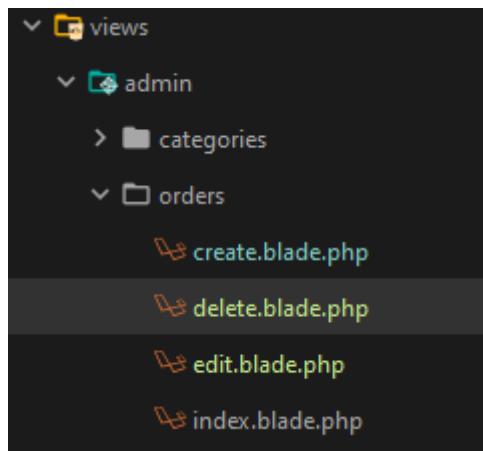
```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderDelete --stop-on-failure

FAIL Tests\Feature\Orders\OrderDeleteTest
✓ guest can not see the order delete page
✓ customer can not see the order delete page
✓ sales can not see the order delete page
✗ admin can see the order delete page

---
• Tests\Feature\Orders\OrderDeleteTest > admin can see the order delete page
  InvalidArgumentException

View [admin.orders.delete] not found.
```

De view is er niet, dus maken we deze aan.



Daarna weer de test

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderDelete --stop-on-failure

FAIL Tests\Feature\Orders\OrderDeleteTest
✓ guest can not see the order delete page
✓ customer can not see the order delete page
✓ sales can not see the order delete page
✗ admin can see the order delete page

---
• Tests\Feature\Orders\OrderDeleteTest > admin can see the order delete page
Failed asserting that '' contains "Customer".
```

Bij De admin gaat het fout, want we checken de inhoud van de html er er staat niks in. We zullen dus de view moeten opmaken, zodat de inhoud erin komt. We zien dat de gebruiker van de order er namelijk niet in staat.

```
<form id="form" class="bg-white shadow-md rounded-lg px-8 pt-6 pb-8 mb-4"
      action="{{ route('orders.destroy', ['order' => $order->id]) }}" method="POST">
  @method('DELETE')
  @csrf
  <div class="mb-4">
    <label class="block text-gray-700 text-sm font-bold mb-2" for="orderdate">
      Datum
    </label>
    <input
      class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
      focus:outline-none focus:shadow-outline @error('orderdate') border-red-500 @enderror" name="orderdate"
      id="orderdate" value="{{ old('orderdate', $order->orderdate) }}" type="datetime-local" disabled>
  </div>
  <div class="mb-4">
    <label class="block text-gray-700 text-sm font-bold mb-2" for="user_id">
      User
    </label>
    <select
      class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
      focus:outline-none focus:shadow-outline" name="user_id" id="user_id" disabled>
      @foreach($users as $user)
        <option value="{{ $user->id }}>
          @if( old('user_id', $order->user_id) == $user->id)
            selected
          @endif
        >{{ $user->name }}</option>
      @endforeach
    </select>
  </div>
  <div class="flex items-center justify-between">
    <button id="submit"
      class="bg-green-500 hover:bg-green-700 text-white font-bold py-2 px-4 rounded
      focus:outline-none focus:shadow-outline" type="submit">Submit
    </button>
  </div>
</form>
```

Let op, dat je de route goed zet en dat de inputs disabled zijn.

Als we nu testen krijgen we dit.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderDelete --stop-on-failure

FAIL Tests\Feature\Orders\OrderDeleteTest
✓ guest can not see the order delete page
✓ customer can not see the order delete page
✓ sales can not see the order delete page
✗ admin can see the order delete page

---
• Tests\Feature\Orders\OrderDeleteTest > admin can see the order delete page
  Facade\Ignition\Exceptions\ViewException

Undefined variable $order (View: D:\Wamp\sites\laravelv1\resources\views\admin\orders\delete.blade.php)
```

De variabelen die we willen hebben worden niet in de controller meegegeven. We zorgen dat we bij de controller alles meegeven

```
public function delete(Order $order)
{
    return view( view: 'admin.orders.delete', compact( var_name: 'order'));
}
```

We testen opnieuw

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderDelete --stop-on-failure

FAIL Tests\Feature\Orders\OrderDeleteTest
✓ guest can not see the order delete page
✓ customer can not see the order delete page
✓ sales can not see the order delete page
✗ admin can see the order delete page

---
• Tests\Feature\Orders\OrderDeleteTest > admin can see the order delete page
  Facade\Ignition\Exceptions\ViewException

Undefined variable $users (View: D:\Wamp\sites\laravelv1\resources\views\admin\orders\delete.blade.php)
```

De users zijn ook niet te vinden. Logisch, want deze hadden we ook nodig bij de edit.

We passen dan de delete methode aan met de user erbij

```
public function delete(Order $order)
{
    $users = User::all();
    return view( view: 'admin.orders.delete', compact( var_name: 'order', ...var_names: 'users'));
}
```

Met opnieuw testen krijgen we dit:

```
D:\Wamp\sites\laravelv1.\vendor\bin\pest --group=OrderDelete --stop-on-failure

FAIL Tests\Feature\Orders\OrderDeleteTest
✓ guest can not see the order delete page
✓ customer can not see the order delete page
✓ sales can not see the order delete page
✗ admin can see the order delete page

-- 

• Tests\Feature\Orders\OrderDeleteTest > admin can see the order delete page
Failed asserting that '<!doctype html>\n<html lang="en">\n
```

Met de html error checken we altijd even aan de onderkant van de test

```
' contains "2021-04-23T13:45".

at D:\Wamp\sites\laravelv1\tests\Feature\Orders\OrderDeleteTest.php:42
38 |         ->get(route('orders.delete', ['order' => $this->order->id]))
39 |         ->assertViewIs('admin.orders.delete')
40 |         ->assertSee($this->order->user->name)
41 |         ->assertSee(Carbon::createFromFormat('Y-m-d H:i:s', $this->order->orderdate)
→ 42 |                 ->format('Y-m-d\TH:i'))
43 |         ->assertStatus(200);
44 | })->group('OrderDelete');
```

De datum komt niet goed in de layout.

Net zoals bij de edit voegen we dan het format van de orderdate erbij

```
public function delete(Order $order)
{
    $users = User::all();
    $order->orderdate = Carbon::createFromFormat('Y-m-d H:i:s', $order->orderdate)
        ->format('Y-m-d\TH:i');
    return view('admin.orders.delete', compact('order', ...$users));
}
```

Nu staat alles zoals we willen. De gehele test werkt nu.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderDelete --stop-on-failure

  PASS Tests\Feature\Orders\OrderDeleteTest
  ✓ guest can not see the order delete page
  ✓ customer can not see the order delete page
  ✓ sales can not see the order delete page
  ✓ admin can see the order delete page

  Tests: 4 passed
  Time: 3.32s
```

## 8.10. Destroy

Als we gaan kijken naar de destroy zal de test erg lijken op die van de product destroy. De basis is weer hetzelfde als bij de andere testen van de order. Eerst gaan we de test aanmaken.

```
D:\Wamp\sites\laravelv1>php artisan pest:test Orders/OrderDestroyTest

[OK] `tests/Feature/Orders/OrderDestroyTest.php` created successfully.
```

De opzet is weer hetzelfde als bij de andere

```
use App\Models\User;
use App\Models\Order;
use \Pest\Laravel;

beforeEach(function () {
    $this->seed(['class' => 'RoleAndPermissionSeeder']);
    $this->seed(['class' => 'UserSeeder']);
    $this->order = Order::factory()->create();
});
```

Bij de testen krijgt de gast een 401 omdat hij niet is ingelogd. De customer en sales krijgen een assertForbidden. Alleen de admin mag erbij. De order zou dan ook weg moeten zijn.

```
test( description: 'guest can not destroy an order', function () {
    $this->json( method: 'DELETE', route( name: 'orders.destroy', ['order' => $this->order->id]))
        ->assertStatus( status: 401);
})->group( ...groups: 'OrderDestroy');

test( description: 'customer can not destroy an order', function () {
    $customer = User::find(1);
    Laravel\be($customer);
    $this->json( method: 'DELETE', route( name: 'orders.destroy', ['order' => $this->order->id]))
        ->assertForbidden();
})->group( ...groups: 'OrderDestroy');

test( description: 'sales can not destroy an order', function () {
    $sales = User::find(2);
    Laravel\be($sales);
    $this->json( method: 'DELETE', route( name: 'orders.destroy', ['order' => $this->order->id]))
        ->assertForbidden();
})->group( ...groups: 'OrderDestroy');

test( description: 'admin can destroy an order', function(){
    $admin = User::find(3);
    Laravel\be($admin);
    $this->json( method: 'DELETE', route( name: 'orders.destroy', ['order' => $this->order->id]))
        ->assertRedirect(route( name: 'orders.index'));
    $this->assertDatabaseMissing( table: 'orders', ['id' => $this->order->id]);
})->group( ...groups: 'OrderDestroy');
```

Als we gaan testen werken de testen bijna. Alleen de admin werkt nog niet.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderDestroy --stop-on-failure

FAIL Tests\Feature\Orders\OrderDestroyTest
✓ guest can not destroy an order
✓ customer can not destroy an order
✓ sales can not destroy an order
✗ admin can destroy an order

---
• Tests\Feature\Orders\OrderDestroyTest > admin can destroy an order
Response status code [200] is not a redirect status code.
Failed asserting that false is true.
```

We krijgen een status 200 terug en niet een redirect. We gaan dus eerst een redirect neerzetten in de destroy methode.

```
public function destroy(Order $order)
{
    return redirect()->route('orders.index')->with('status', 'Order verwijderd');
}
```

Daarna weer testen

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderDestroy --stop-on-failure

 FAIL Tests\Feature\Orders\OrderDestroyTest
 ✓ guest can not destroy an order
 ✓ customer can not destroy an order
 ✓ sales can not destroy an order
 ✘ admin can destroy an order

 ---
 
 • Tests\Feature\Orders\OrderDestroyTest > admin can destroy an order
 Failed asserting that a row in the table [orders] does not match the attributes {
     "id": 1
 }.
```

We hebben ook nog niks in de destroy staan, dus de order wordt niet verwijderd. We gaan dus de order verwijderen in de methode destroy.

```
public function destroy(Order $order)
{
    $order->delete();
    return redirect()->route('orders.index')->with('status', 'Order verwijderd');
}
```

Nu werkt de test al geheel

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --group=OrderDestroy --stop-on-failure

  PASS  Tests\Feature\Orders\OrderDestroyTest
    ✓ guest can not destroy an order
    ✓ customer can not destroy an order
    ✓ sales can not destroy an order
    ✓ admin can destroy an order

  Tests:  4 passed
  Time:   3.31s
```

Als we voor de eindcontrole nog is alle testen uitvoeren zien we dat alles nu goed door de test komt.

```
D:\Wamp\sites\laravelv1>.\vendor\bin\pest --stop-on-failure

  PASS  Tests\Feature\AuthenticationTest
    ✓ login screen can be rendered
    ✓ users can authenticate using the login screen
    ✓ users can not authenticate with invalid password

  PASS  Tests\Feature\Products\ProductUpdateTest
    ✓ customer can not update a product
    ✓ guest can not update a product
    ✓ admin can update a product
    ✓ sales can update a product

  Tests:  104 passed
  Time:   68.06s
```

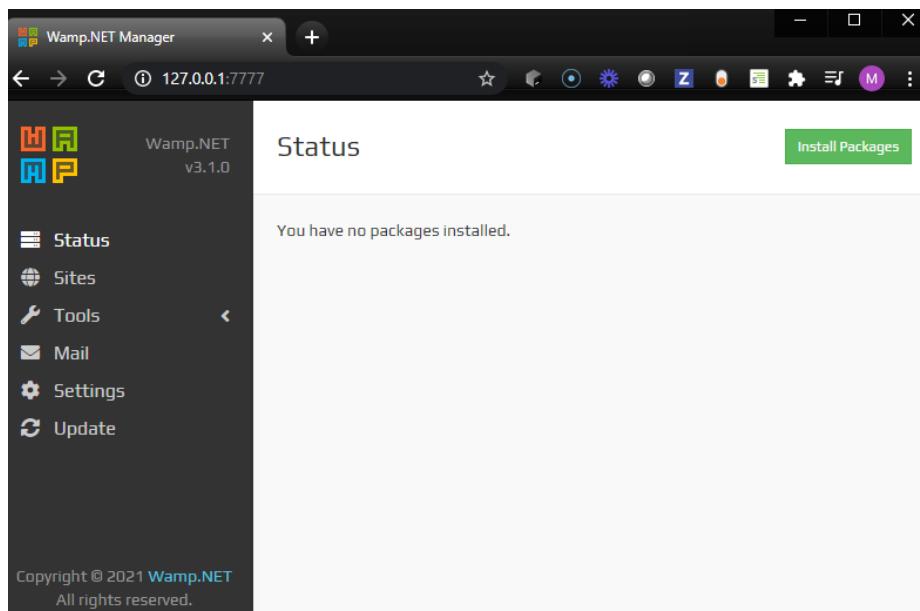
De Order is nu tot zover klaar en dus ook het stuk TDD in deze lessenserie. Je hebt nu alle basics van Feature testen, Unit testen en Test Driven Development gehad.

# 9. Omgeving

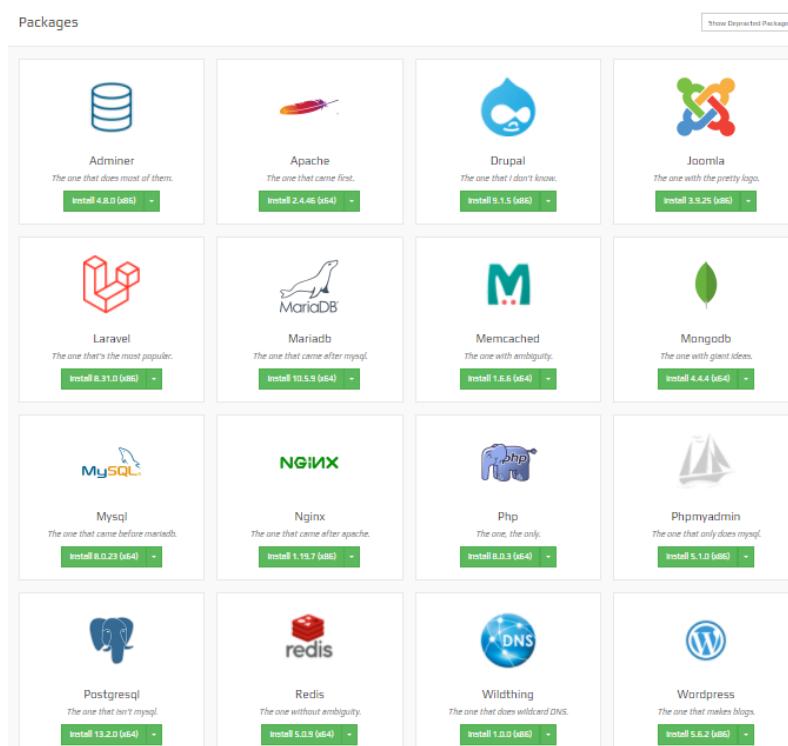
## 9.1. Wamp.net installatie met Laravel

Op <https://wamp.net/> kan je een omgeving downloaden waarmee je allerlei packages kan downloaden, zoals nginx, mariadb, phpmyadmin en laravel. Het kan zijn dat je eerder gewerkt hebt met andere tools, of met een eerdere versie van wamp, maar het ziet er uit dat dit redelijk vernieuwd is met allerlei nieuwe opties, zoals virtual hosts en makkelijk updaten van packages.

Als je wamp.net installeert en opstart krijg je een webomgeving te zien waarbij alle configuratie mogelijkheden staan. Erg clean en overzichtelijk. Standaard wil hij het op c:\wamp installeren, maar zelf heb ik het op d:\wamp geinstalleerd.



Als je naar install packages gaat, zie je dit:



Misschien ben je gewend om met Apache te werken, maar Nginx is eigenlijk een verbeterde versie daarvan. Deze gaan we dan ook als eerst installeren. Bij de installatie kan je de basis en advanced mogelijkheden gewoon zo laten, al zie je wel dat je zelf kan instellen op welke poorten de omgeving moet komen.

### Nginx 1.19.7 (x86) Installation

The screenshot shows the 'Basic' tab selected in the top navigation bar. On the left, there's a 'Label' section with the text 'nginx 1.19.7 (x86)' in a box. On the right, there are fields for 'Interface' (set to '127.0.0.1 (Local Only - Recommended)'), 'HTTP Port' (set to '80'), and 'HTTPS Port' (set to '443'). At the bottom, there are 'Continue' and 'Cancel' buttons.

Als je op Continue drukt, wordt het geïnstalleerd en zie je dit:

### Status

The screenshot shows a table with columns: NAME, INSTALLED, INTERFACES, PIDS, and SERVICE. One row is visible: 'nginx 1.19.7 (x86)' with '2021-03-12 10:42:46' in the INSTALLED column, '-' in both INTERFACES and PIDS, and a checkbox icon in the SERVICE column. At the bottom right are 'Start', 'Browse', and a dropdown menu.

NAME	INSTALLED	INTERFACES	PIDS	SERVICE
nginx 1.19.7 (x86)	2021-03-12 10:42:46	-	-	<input type="checkbox"/> <span>▶ Start</span> <span>Browse</span> ▾

De webserver is geïnstalleerd. Let op, deze staat nog niet aan!

Voor de database heb je Mysql. Ook daar heb je een verbeterde versie van, namelijk MariaDB. Deze gaan we dan ook installeren. Ook hierbij laten we de standaard staan. (lokaal is niet zozeer beveiliging nodig met bij een root password)

### Mariadb 10.5.9 (x64) Installation

The screenshot shows the 'Basic' tab selected in the top navigation bar. On the left, there's a 'Label' section with the text 'mariadb 10.5.9 (x64)' in a box. Below it are sections for 'Data Location' (with a note to leave blank for automatic assignment) and 'Root Password'. On the right, there are fields for 'Interface' (set to '127.0.0.1 (Local Only - Recommended)'), 'TCP Port' (set to '3306'), 'key\_buffer\_size' (set to '32'), and 'max\_allowed\_packet' (set to '128'). At the bottom, there are 'Continue' and 'Cancel' buttons.

Bij de installatie zie je, net als bij Nginx iets als dit:

### Mariadb 10.5.9 (x64) Installation



Uiteindelijk staat mariadb onder de nginx. De webserver en database zijn geregeld.

### Status

[Install Packages](#)

Success Installation Complete!					
NAME	INSTALLED	INTERFACES	PIDS	SERVICE	
nginx 1.19.7 (x86)	2021-03-12 10:42:46	-	-	<input type="checkbox"/>	<a href="#">▶ Start</a> <a href="#">Browse</a> ▾
mariadb 10.5.9 (x64)	2021-03-12 10:46:43	-	-	<input type="checkbox"/>	<a href="#">▶ Start</a> <a href="#">Browse</a> ▾

Voor Laravel hebben we natuurlijk ook PHP nodig. De laatste versie op dit moment is php 8.0.3. Deze gaan we dan ook installeren. De tijdzone heb ik nog wel even aangepast, maar verder de basis instellingen behouden.

### PHP 8.0.3 (x64) Installation

General   FCGI Daemon   Resources   Modules

**Label**  
php 8.0.3 (x64)

**Time Zone**  
Europe/Amsterdam

[Continue](#) [Cancel](#)

Zoals je kan zien is ook php geinstalleerd.

### Status

[Install Packages](#)

Success Installation Complete!					
NAME	INSTALLED	INTERFACES	PIDS	SERVICE	
nginx 1.19.7 (x86)	2021-03-12 10:42:46	-	-	<input type="checkbox"/>	<a href="#">▶ Start</a> <a href="#">Browse</a> ▾
mariadb 10.5.9 (x64)	2021-03-12 10:46:43	-	-	<input type="checkbox"/>	<a href="#">▶ Start</a> <a href="#">Browse</a> ▾
php 8.0.3 (x64)	2021-03-12 10:50:22	-	-	<input type="checkbox"/>	<a href="#">▶ Start</a> <a href="#">Browse</a> ▾

Voordat we nu verder kunnen, zullen we een site aan moeten maken. Je moet zien, dat een site een project is waar je mee bezig bent. Links in het menu kan je de sites vinden. Je kan hier de domain name invullen. Dit wordt de naam van de url in je lokale omgeving. Ik noem het nu laravelv1. Daarbij selecteer ik de webserver en php versie die we net hebben geïnstalleerd.

## Create Site

**Domain Name**  
laravelv1

**Domain Aliases**  
e.g. www or www, api or \*

**Document Root**  
Leave blank for default

**Web Server**  
nginx 1.19.7 (x86)

**PHP Version**  
php 8.0.3 (x64)

Als je deze dan aanmaakt zie je dit:

## Sites

Success Site successfully created.					
DOMAIN NAME	ALIASES	BROWSE	DOCUMENT ROOT	WEB SERVER	PHP VERSION
laravelv1	-	http   https	D:\wamp\sites\laravelv1	nginx 1.19.7 (x86)	php 8.0.3 (x64)

Door het aanmaken van de site, zie je ook bij status dat de webserver gestart is.

NAME	INSTALLED	INTERFACES	PIDS	SERVICE
nginx 1.19.7 (x86)	2021-03-12 10:42:46	TCP 127.0.0.1 80   TCP 127.0.0.1 443	10048   11584	<input type="checkbox"/> <input type="button" value="Stop"/> <input type="button" value="Restart"/> <input type="button" value="Browse"/>

We kunnen de rest ook gaan starten:

NAME	INSTALLED	INTERFACES	PIDS	SERVICE
nginx 1.19.7 (x86)	2021-03-12 10:42:46	TCP 127.0.0.1 80   TCP 127.0.0.1 443	10048   11584	<input type="checkbox"/> <input type="button" value="Stop"/> <input type="button" value="Restart"/> <input type="button" value="Browse"/>
mariadb 10.5.9 (x64)	2021-03-12 10:46:43	TCP 127.0.0.1 3306	13588	<input type="checkbox"/> <input type="button" value="Stop"/> <input type="button" value="Restart"/> <input type="button" value="Browse"/>
php 8.0.3 (x64)	2021-03-12 10:50:22	TCP 127.0.0.1 803	12552	<input type="checkbox"/> <input type="button" value="Stop"/> <input type="button" value="Restart"/> <input type="button" value="Browse"/>

De volgende package zal Laravel zijn. Bij het tabblad site kiezen we voor de site die we net hebben aangemaakt.

#### Laravel 8.31.0 (x86) Installation

This screenshot shows the 'Site' configuration step of the Laravel installation wizard. It has three tabs at the top: 'Site' (selected), 'Database', and 'Laravel'. The 'Site' section contains a dropdown menu set to 'laravelv1'. The 'Path' section contains a text input field with the placeholder 'e.g. /dir/blog (Optional - leave blank to install in root of site)'. At the bottom are 'Continue' and 'Cancel' buttons.

Bij het tabblad database kies je voor de database server MariaDB. Verder kan je de database naam invullen, waarbij ik dezelfde naar als de site gebruik.

#### Laravel 8.31.0 (x86) Installation

This screenshot shows the 'Database' configuration step of the Laravel installation wizard. It has three tabs at the top: 'Site' (selected), 'Database' (selected), and 'Laravel'. Under 'Database Server', the 'mariadb 10.5.9 (x64)' option is selected. Under 'New Database Name', the value 'laravelv1' is entered. At the bottom are 'Continue' and 'Cancel' buttons.

Verder kan je de naam van de applicatie aanpassen en de environment. Voor ons local, dus dat kunnen we laten staan. De naam kan je wel veranderen, waarbij ik weer dezelfde naam als de site gebruik.

#### Laravel 8.31.0 (x86) Installation

This screenshot shows the 'Application' configuration step of the Laravel installation wizard. It has three tabs at the top: 'Site' (selected), 'Database', and 'Laravel'. The 'Application Name' is set to 'Laravelv1'. The 'Application Environment' is set to 'local'. A checkbox for 'Application Debug' is checked. At the bottom are 'Continue' and 'Cancel' buttons.

Als je dan op Continue klikt, zal Laravel geïnstalleerd worden.

## Laravel 8.31.0 (x86) Installation



Met als resultaat:

Sites Create Site

DOMAIN NAME	ALIASES	BROWSE	DOCUMENT ROOT	WEB SERVER	PHP VERSION	
laravelv1	-	http   https	D:\wamp\sites\laravelv1	nginx 1.19.7 (x86)	php 8.0.3 (x64)	<span style="color: red;">Delete</span> <span style="color: blue;">Edit</span> <span style="color: cyan;">Preview</span>

Nu kan je meteen naar de https gaan om de site te zien, alleen krijg je dan dit te zien:

The browser address bar shows: `< → C 🔒 laravelv1/`

The main content of the page is: **403 Forbidden**

Below the content, it says: `nginx/1.19.7`

Dit komt omdat de root map voor Laravel de public map is. Als we de site aanpassen (wielje) kunnen we de document root wijzigen naar de public map.

Bij Document root zetten we dan ook de public map erbij.

## Edit Site

**Domain Name**  
laravelv1

**Domain Aliases**  
e.g www or www, api or \*

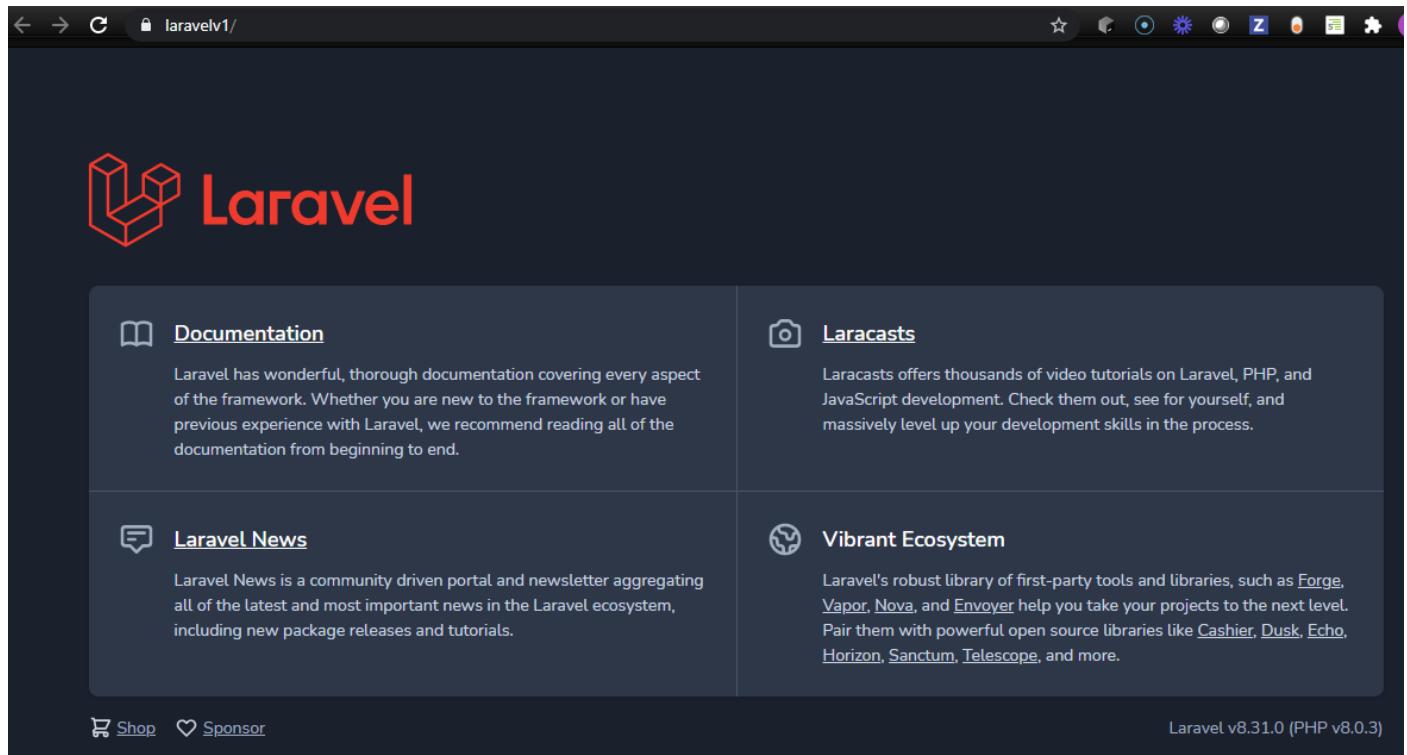
**Document Root**  
D:\wamp\sites\laravelv1\public [Browse](#)

**Web Server**  
nginx 1.19.7 (x86)

**PHP Version**  
php 8.0.3 (x64)

[Update](#) [Cancel](#)

Als je nu naar de https link gaat, zie je de startpagina van Laravel.



The screenshot shows a web browser window with the URL 'laravelv1/' in the address bar. The page itself is the Laravel welcome screen. It features the Laravel logo (a red 'L' composed of three cubes) and the word 'Laravel' in a large red font. Below this, there are four main sections: 'Documentation' (with a book icon), 'Laracasts' (with a camera icon), 'Laravel News' (with a speech bubble icon), and 'Vibrant Ecosystem' (with a gear icon). Each section contains a brief description and links to further resources. At the bottom of the page, there are links for 'Shop' and 'Sponsor', and a note indicating the page is running on 'Laravel v8.31.0 (PHP v8.0.3)'.

We missen nu nog 1 ding, en dat is de phpMyAdmin.

Bij de installatie kunnen van phpMyAdmin package kunnen we nu de site en path invullen. Vergeet niet dat we voor phpMyAdmin, wat zichtbaar moet zijn, ook in de public map te zetten. Door middel van de path doe je dit.

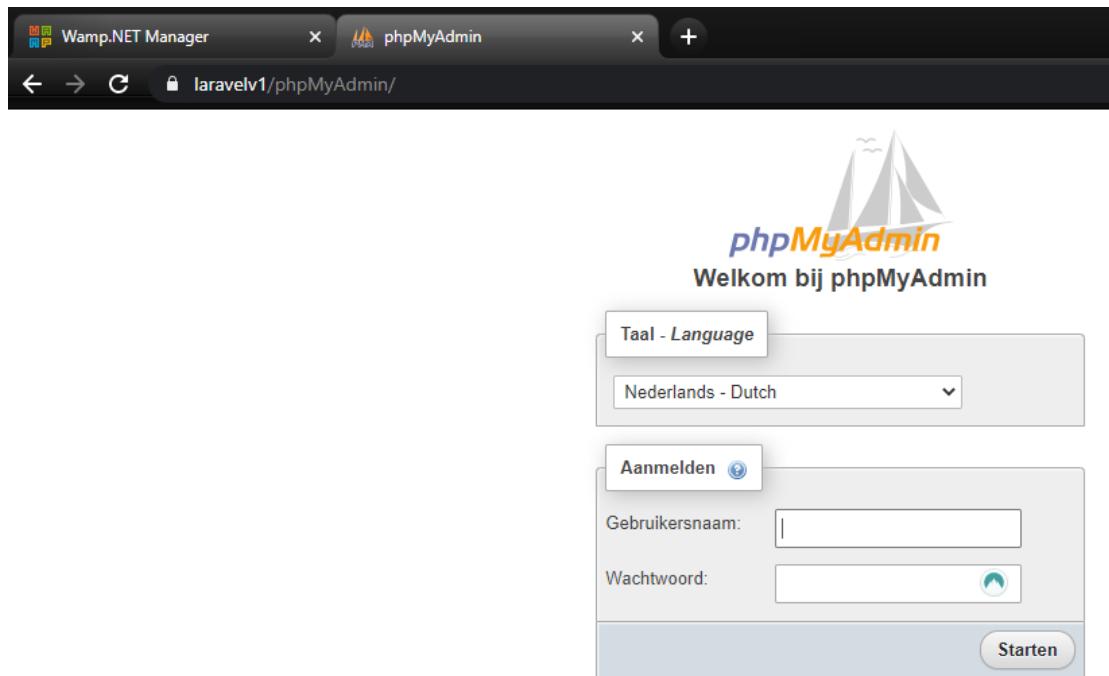
## Phpmyadmin 5.1.0 (x86) Installation

Site  
laravelv1

Path  
/public/phpMyAdmin

**Continue** **Cancel**

Als de installatie klaar is kan je in de browser naar je phpMyAdmin. Deze staat gewoon in je site in de map phpMyAdmin.



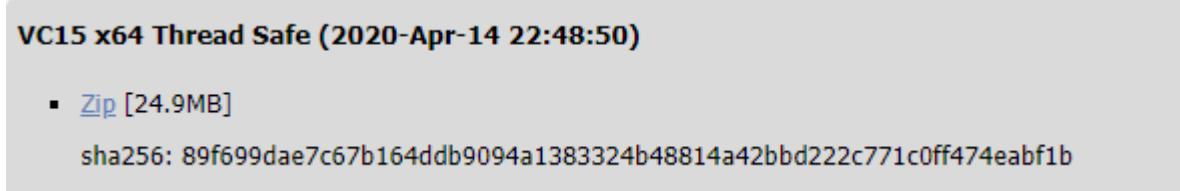
Als je inlogt met root, kan je je database zien (laravelv1). Nu is de omgeving geheel klaar.

## 9.2. Laragon gebruiksklaar maken

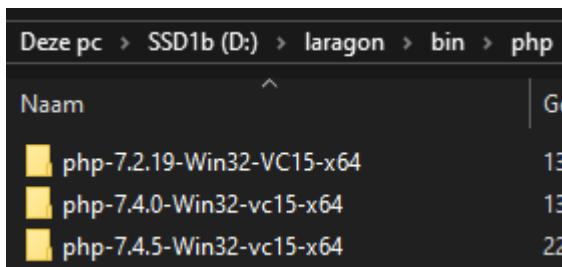
Als eerst gaan we aan de slag met de installatie van de omgeving. Ik neem aan dat de installatie van Laragon (<https://laragon.org/>) gewoon goed gaat. Op de website van Laravel staan de vereisten. De omgeving van Laragon op dit moment voldoet nog niet aan de eisen van Laravel. Dit maakt niet uit, want we kunnen het gewoon updaten. (<https://laravel.com/docs/8.x>)

Om ervoor te zorgen dat we straks alles kunnen doen, gaan we eerst Laragon updaten. In de huidige versie zit php versie 7.2.19. Laravel heeft minimaal 7.3 nodig, dus php heeft een update nodig. Dit is erg simpel.

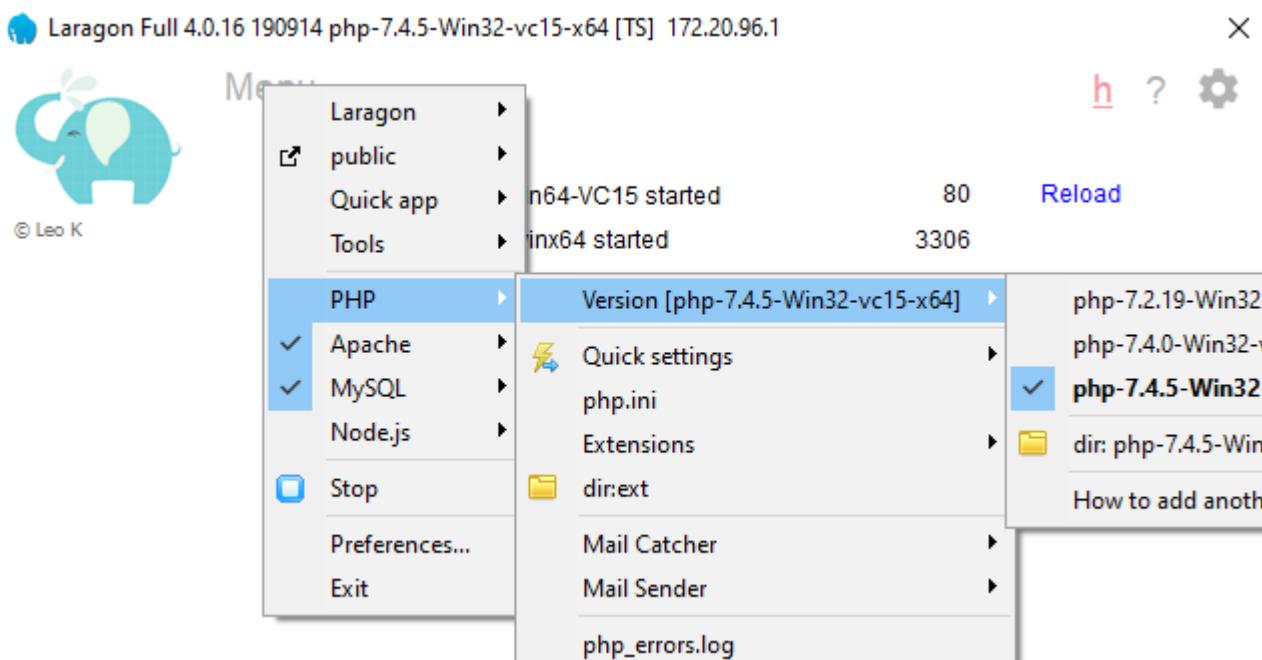
Download eerst de nieuwe php versie. Voor mij nu versie 7.4.5 van <https://windows.php.net/download#php-7.4>. Ik pak hier de VC15 x64 Thread Safe versie. ( Ondertussen is php 8 al beschikbaar )



Deze pak ik uit in Laragon\bin\php

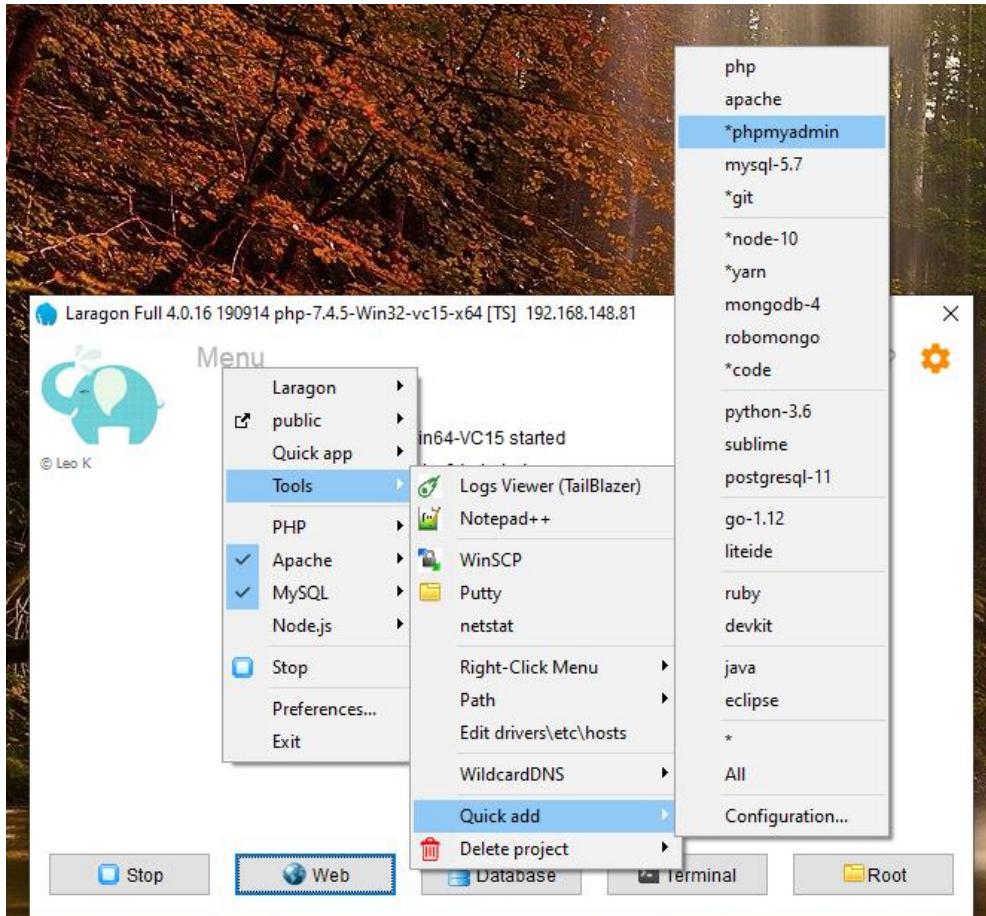


PHP is nu up to date. Je moet nog wel de php versie kiezen in Laragon



Nu gebruik ik standaard phpMyAdmin als beheer programma voor de MySQL database. Dit kan gewoon met Laragon, maar is niet de standaard omgeving.

We kunnen deze toevoegen door in het menu te gaan naar: Tools->Quick add->\*phpmyadmin



PhpMyAdmin heeft alweer nieuwe versies, die ik zelf prettiger vind. (Bij sommige werkt de phpMyAdmin versie niet met de nieuwe php versie). Hierdoor ga ik naar de website van phpMyAdmin en download degene die voor mij zal werken (versie 5.0.2 niet nemen)

<https://www.phpmyadmin.net/downloads/>

## phpMyAdmin 4.9.5

Released 2020-03-21, see [release notes](#) for details.

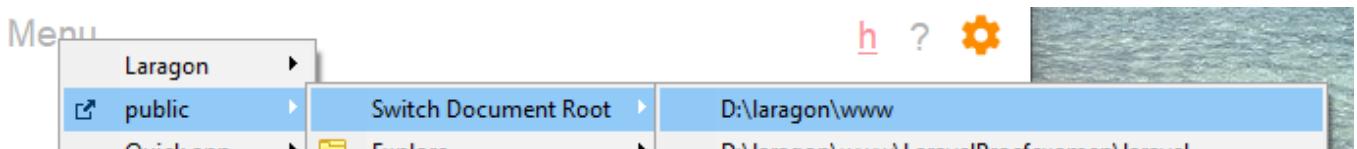
Older version compatible with PHP 5.5 to 7.4 and MySQL 5.5 and newer. Currently supported for security fixes only.

File	Size	Verification
<a href="#">phpMyAdmin-4.9.5-all-languages.zip</a>	10.6 MB	[PGP] [SHA256]
<a href="#">phpMyAdmin-4.9.5-all-languages.tar.gz</a>	9.7 MB	[PGP] [SHA256]

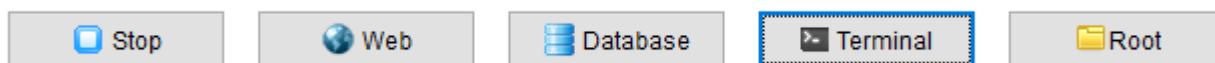
Ik verwijder alle bestanden in Laragon/etc/aps/phpMyAdmin, en zet daar de nieuwe bestanden van versie 4.9.5 in. Hiermee heb je dat phpMyAdmin geüpdate. De omgeving voor Laravel is dan klaar.

### 9.3. Installatie met laravel installer binnen laragon

Voordat we verder gaan met de uitgebreide installatie zal je dit even moeten checken. Als je al eerder met Laragon hebt gewerkt, check even of de webroot wel op Laragon/www staat. Heb je net Laragon geïnstalleerd, staat hij hier standaard op.



De installatie van Laravel 8 gaan we met de Terminal doen. Deze zit ingebouwd in Laragon.



Daarna zorgen we dat de laravel installer door composer wordt binnengehaald

```
D:\laragon\www
λ composer global require laravel/installer
Changed current directory to C:/Users/mkoni/AppData/Roaming/Composer
Using version ^4.0 for laravel/installer
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
 - Installing laravel/installer (v4.0.5): Loading from cache
Writing lock file
Generating autoload files
```

Zelf heb ik al Laravel geïnstalleerd gehad, waardoor er geen dependencies geïnstalleerd hoeven te worden. Bij een eerste installatie van Laravel zal dit wel het geval zijn.

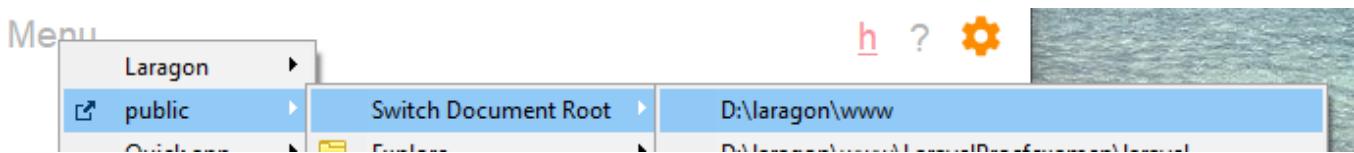
Mocht het zijn dat je een eerdere versie van Laravel heb gebruikt, dan kan je proberen met *composer global update* de installer te updaten. Lukt dit niet en je hebt nog installer versie 3, verwijder dan de global met: *composer global remove laravel/installer*. Daarna kan je de geupdate installer ophalen met composer

De eerste stap die we nu doen is Laravel 8 installeren. Nu is het mogelijk om met Laravel 8 bijvoorbeeld Laravel Sail te gebruiken, waarbij je aan de slag gaat met Docker. Dit gaan we niet doen, omdat je later toch dan de omgeving in moet duiken wat op dit moment te veel tijd kost. Het doel is nu om Laravel te leren in een simpele omgeving.

Naast Sail kan je bijvoorbeeld ook Jetstream bij Laravel installeren, wat je kan helpen om voor authenticatie en zelfs rollen. Nu Laravel 8 al even uit is en heb rond gevraagd bij bedrijven wat ze gebruiken, wordt Jetstream niet vaak gebruikt.

Door dit allemaal kiezen we voor de basis installatie van Laravel, met daarbij een package van Spatie om de permissies te regelen. Dit is voor nu de simpelste en meest gebruikte manier. Er zijn 2 opties om te beginnen, namelijk de quick app van laragon gebruiken, of composer. Beide geven hetzelfde resultaat.

Voordat je dit doet. Als je al eerder met Laragon hebt gewerkt, check even of de webroot wel op Laragon/www staat. Heb je net Laragon geïnstalleerd, staat hij hier standaard op.



Met composer gaan we dan laravel installeren met behulp van de laravel installer:

```
D:\laragon\www
$ laravel new laravel85

[laravel85] | [laravel85] | [laravel85]

Creating a "laravel/laravel" project at "./laravel85"
Installing laravel/laravel (v8.5.9)
- Installing laravel/laravel (v8.5.9): Extracting archive
Created project in D:\laragon\www\laravel85
```

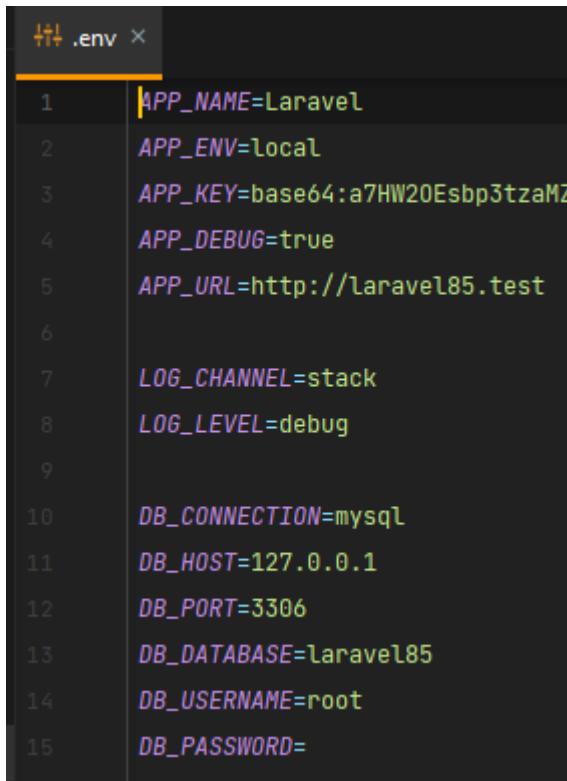
Via composer zal je zelf een database aan moeten maken. Zelf houd ik van de controle, dus hierdoor heb ik zelf voor de composer installatie gekozen. Hierdoor maak ik nog een database aan in phpMyAdmin voor de website:

## Databases

A screenshot of the phpMyAdmin interface. At the top, there's a header with the title 'Databases'. Below the header, there's a search bar containing the text 'laravel85'. To the right of the search bar is a blue 'Create' button with a plus sign icon. The main area is currently empty, showing a light gray background.

Om ervoor te zorgen dat we deze gaan gebruiken binnen ons project zullen we de .env file moeten controleren of alles goed staat.

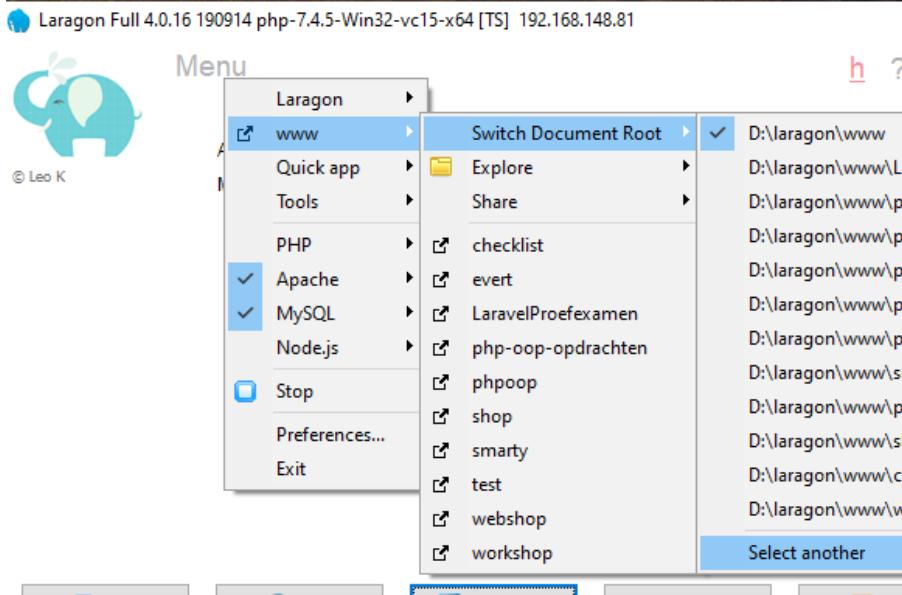
In de root directory van ons project vinden we een .env bestand. Dit wordt gebruikt als een environment bestand, waarin alle instellingen staan. Hier checken we de DB\_DATABASE, die op laravel85 staat, zoals ik ook de database in phpMyAdmin heb genoemd. Dit zodat de database gebruikt gaat worden.



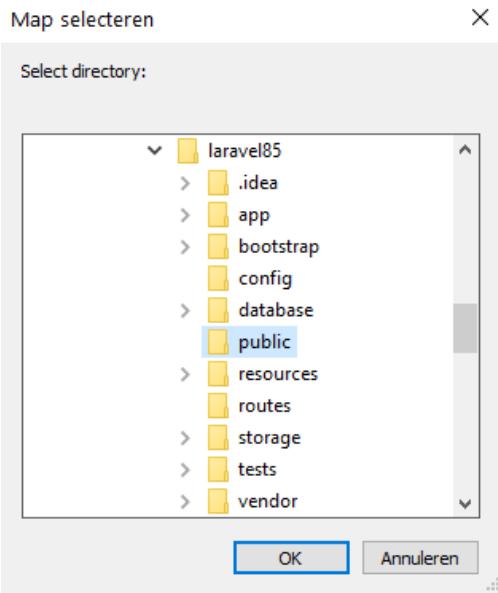
```
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:a7HW20Esbp3tzaMZ
4 APP_DEBUG=true
5 APP_URL=http://laravel85.test
6
7 LOG_CHANNEL=stack
8 LOG_LEVEL=debug
9
10 DB_CONNECTION=mysql
11 DB_HOST=127.0.0.1
12 DB_PORT=3306
13 DB_DATABASE=laravel85
14 DB_USERNAME=root
15 DB_PASSWORD=
```

Je hebt ook een APP\_KEY erin staan. Deze is uniek en zorgt voor een stuk beveiliging. Deel deze APP\_KEY nooit. Het .env bestand staat ook in de .gitignore, zodat je niet dit bestand op git gaat zetten.

Om ervoor te zorgen dat we de Laravel website nu gaan zien, gaan we de document root van Laragon veranderen. Hiervoor kiezen we ervoor dat we een andere map willen met menu->www->Switch Document Root->Select another.



Hier kies ik in de workshop map de map public. Dit is namelijk de map die te zien is voor publiek, waar ook de index.php in zit



Je zal een pop-up krijgen die je moet accepteren. Er wordt namelijk op de achtergrond een virtual host aangemaakt voor je.



Als je dan in Laragon op Web klikt, zal de browser openen met je localhost, waarin je de beginpagina van Laravel ziet. (Wel aangenomen dat je de server al wel hebt gestart)

The screenshot shows the Laravel welcome page with the following content:

- Documentation**: Laravel has wonderful, thorough documentation covering every aspect of the framework. Whether you are new to the framework or have previous experience with Laravel, we recommend reading all of the documentation from beginning to end.
- Laracasts**: Laracasts offers thousands of video tutorials on Laravel, PHP, and JavaScript development. Check them out, see for yourself, and massively level up your development skills in the process.
- Laravel News**: Laravel News is a community driven portal and newsletter aggregating all of the latest and most important news in the Laravel ecosystem, including new package releases and tutorials.
- Vibrant Ecosystem**: Laravel's robust library of first-party tools and libraries, such as [Forge](#), [Vapor](#), [Nova](#), and [Envoyer](#) help you take your projects to the next level. Pair them with powerful open source libraries like [Cashier](#), [Dusk](#), [Echo](#), [Horizon](#), [Sanctum](#), [Telescope](#), and more.

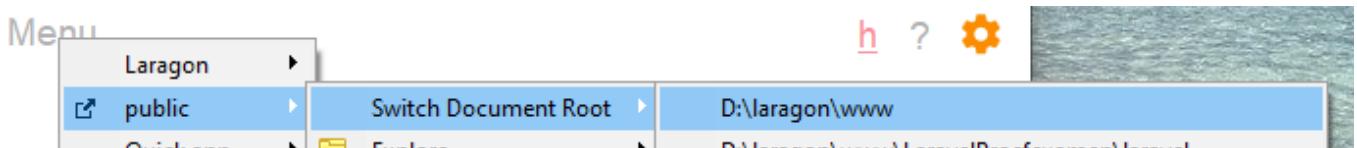
At the bottom of the page, there are links for [Shop](#) and [Sponsor](#). The footer indicates "Laravel v8.25.0 (PHP v7.4.5)".

Onze Laravel installatie is tot zover nu klaar.

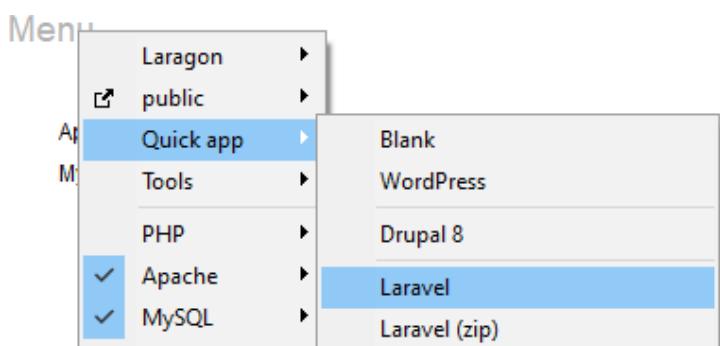
## 9.4. Installatie met quickapp binnen laragon

In Laragon zit een mooie optie om een Laravel project aan te maken. Deze gaan we dan ook gebruiken. Let op, dat je niet de allerlaatste versie van Laravel gaat krijgen met deze optie.

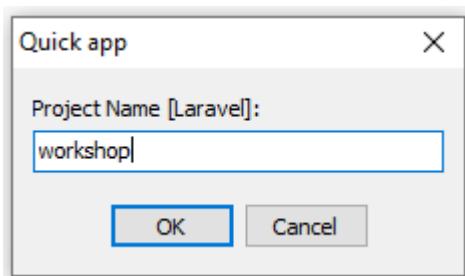
Voordat je dit doet. Als je al eerder met Laragon hebt gewerkt, check even of de webroot wel op Laragon/www staat. Heb je net Laragon geïnstalleerd, staat hij hier standaard op.



Bij het menu kunnen we dit vinden in: Quick app->Laravel



Je krijgt dan een nieuw scherm waarbij je de projectnaam kan invullen. Ik ga het project 'workshop' noemen.



Een command prompt komt op, waarin de installatie gebeurt. Omdat ik dit natuurlijk al is gedaan heb wordt bij mij alles vanuit een cache geladen. Heb je dit nooit gedaan gaat hij alles van internet downloaden.

```
C:\WINDOWS\SYSTEM32\cmd.exe - php
**** Database:
reated database: [workshop]

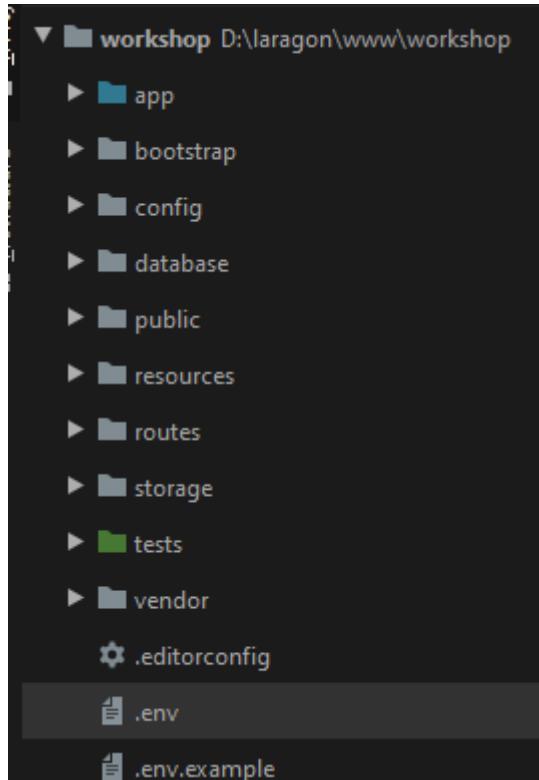
**** Hint: In Terminal, you can type:
-----
d D:\laragon\www
omposer create-project laravel/laravel workshop --prefer-dist
-----

unning.....
nstalling laravel/laravel (v7.6.0)
- Installing laravel/laravel (v7.6.0): Loading from cache
reated project in workshop
@php -r "file_exists('.env') || copy('.env.example', '.env');"
oading composer repositories with package information
pdating dependencies (including require-dev)
ackage operations: 92 installs, 0 updates, 0 removals
- Installing voku/portable-ascii (1.4.10): Loading from cache
- Installing symfony/polyfill-ctype (v1.15.0): Loading from cache
- Installing phoption/phoption (1.7.3): Loading from cache
```

Aan het einde staat dan deze melding

```
Package manifest generated successfully.  
> @php artisan key:generate --ansi  
Application key set successfully.  
  
***** NOTE: Now, you can use pretty url for your awesome project :) *****  
-----  
(Laragon) Project path: D:/laragon/www/workshop  
(Laragon) Pretty url: http://workshop.test  
-----
```

Het project staat netjes in de map www/workshop/



Er is een database voor ons aangemaakt met dezelfde naam: workshop

Om ervoor te zorgen dat we deze gaan gebruiken binnen ons project zullen we dit moeten veranderen bij de instellingen van Laravel.

In de root directory van ons project vinden we een .env bestand. Dit wordt gebruikt als een environment bestand, waarin alle instellingen staan. Hier veranderen we de DB\_DATABASE van Laravel naar workshop. Dit zodat de workshop database gebruikt gaat worden.

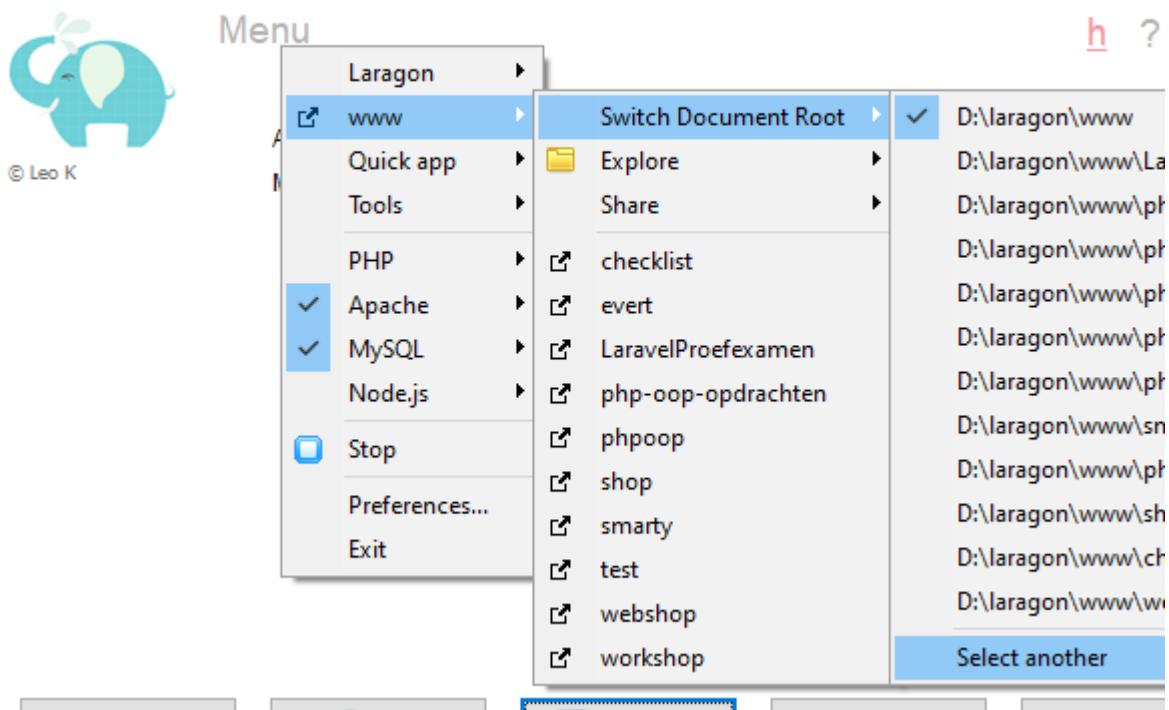
```
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8
9 DB_CONNECTION=mysql
10 DB_HOST=127.0.0.1
11 DB_PORT=3306
12 DB_DATABASE=workshop
13 DB_USERNAME=root
14 DB_PASSWORD=
```

Je hebt ook een APP\_KEY erin staan. Deze is uniek en zorgt voor een stuk beveiliging. Deel deze APP\_KEY nooit. Het .env bestand staat ook in de .gitignore, zodat je niet dit bestand op git gaat zetten.

Om ervoor te zorgen dat we de Laravel website nu gaan zien, gaan we de document root van Laragon veranderen. Hiervoor kiezen we ervoor dat we een andere map willen met menu->www->Switch Document Root->Select another.

---

Laragon Full 4.0.16 190914 php-7.4.5-Win32-vc15-x64 [TS] 192.168.148.81



Hier kies ik in de workshop map de map public. Dit is namelijk de map die te zien is voor publiek, waar ook de index.php in zit

Map selecteren

X

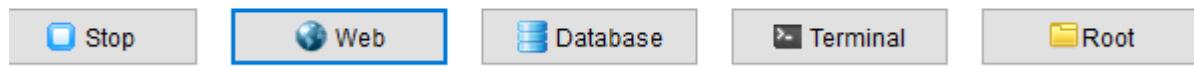
Select directory:

- workshop
  - .idea
  - app
  - bootstrap
  - config
  - database
  - public**
  - resources
  - routes
  - storage
  - tests
  - vendor

OK

Annuleren

Je zal een pop-up krijgen die je moet accepteren. Er wordt namelijk op de achtergrond een virtual host aangemaakt voor je.



Als je dan in Laragon op Web klikt, zal de browser openen met je localhost, waarin je de beginpagina van Laravel ziet. (Wel aangenomen dat je de server al wel hebt gestart)

The screenshot shows the Laravel welcome page. It includes the following sections:

- Documentation**: A link to the official Laravel documentation.
- Laracasts**: A link to Laracasts video tutorials.
- Laravel News**: A link to the Laravel News community portal.
- Vibrant Ecosystem**: A link to the Laravel ecosystem, mentioning tools like Forge, Vapor, Nova, and Envoyer.

At the bottom, there are links for 'Shop' and 'Sponsor', and the text 'Laravel v8.25.0 (PHP v7.4.5)'.

Onze Laravel installatie is tot zover nu klaar.

# 10. Tips & Tricks

## 10.1. Gebruik van phpstorms terminal met laragon

Laragon heeft een eigen terminal, die ervoor zorgt dat alles wat je met Laragon configureert ook gebruikt wordt. De terminal van PhpStorm geeft ook heel wat anders aan. Als voorbeeld, laten we de php versie checken.

Met php -v kan je bekijken welke versie van draait. Binnen de terminal van Laragon zie ik dit.

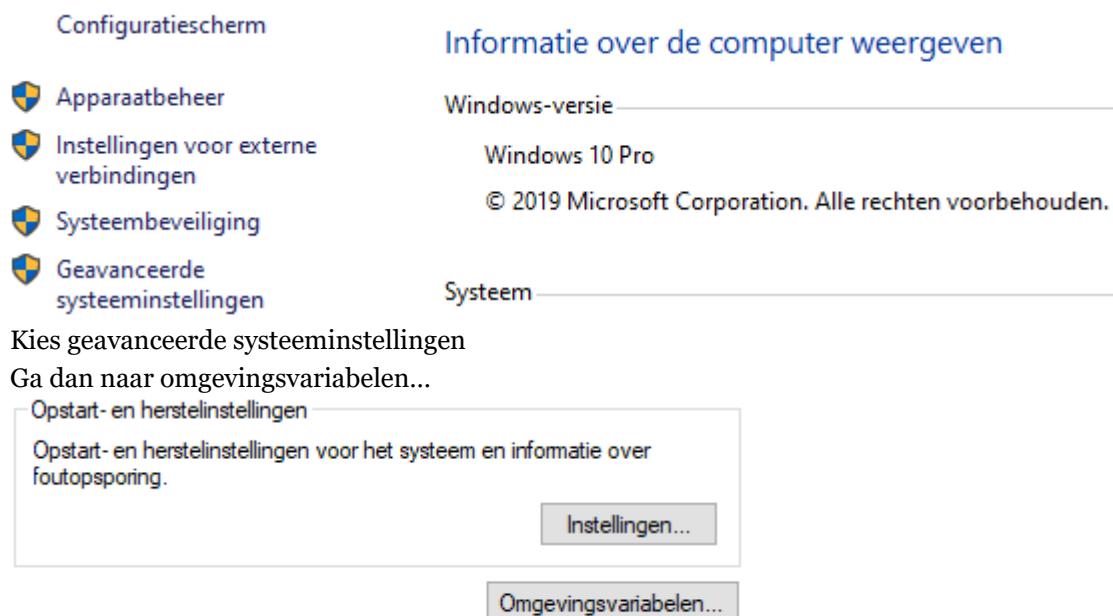
```
D:\laragon\www\workshop (master)
λ php -v
PHP 7.4.5 (cli) (built: Apr 14 2020 16:17:34) ( ZTS Visual C++ 2017 x64 )
Copyright (c) The PHP Group
Zend Engine v3.4.0, Copyright (c) Zend Technologies
```

Maar als ik precies hetzelfde doe binnen PhpStorm, zie ik wat anders

```
D:\laragon\www\workshop>php -v
PHP 7.2.2 (cli) (built: Jan 31 2018 19:51:55) ( ZTS MSVC15 (Visual C++ 2017) x86 )
Copyright (c) 1997-2018 The PHP Group
Zend Engine v3.2.0, Copyright (c) 1998-2018 Zend Technologies
```

In PhpStorm wordt php 7.2.2 gezien, wat niet genoeg is voor bijvoorbeeld Laravel of Pest. De vraag is, waarom komt dit. PhpStorm kijkt wat er in het geheugen “path” staat in Windows. Je kan dit controleren door de volgende stappen.

- Ga naar de verkenner.
- Rechtermuis klik op ‘ deze pc’ en kies eigenschappen. Je ziet dan het volgende:

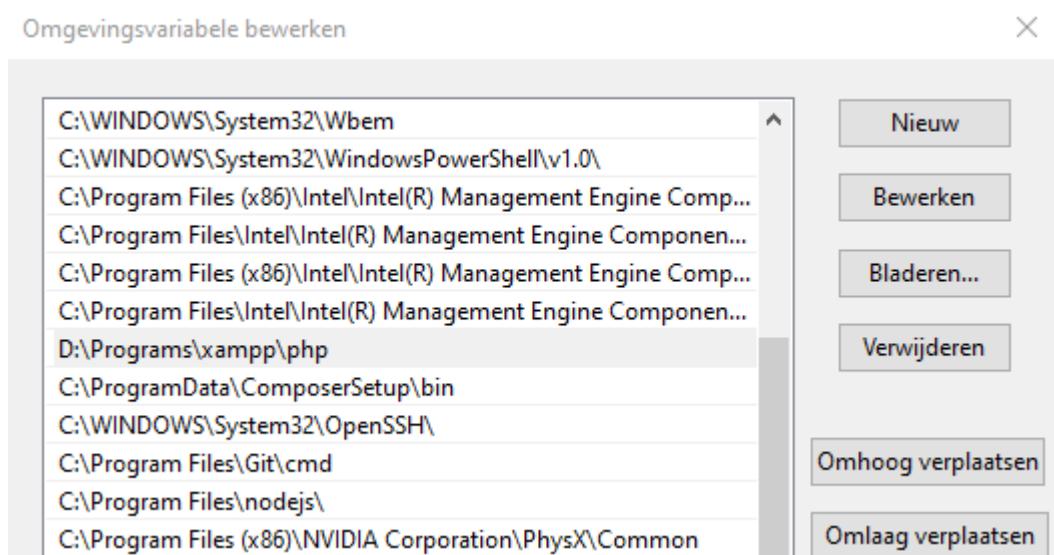


- Let op, hier moet je voorzichtig zijn. Als je het verkeerd doet kan je je Windows om zeep helpen.

- Zoek bij systeemvariabelen: Path

Systeemvariabelen	
Variabele	Waarde
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	12
OS	Windows_NT
Path	C:\ProgramData\DockerDesktop\version-bin;C:\Program Files\Doc...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64

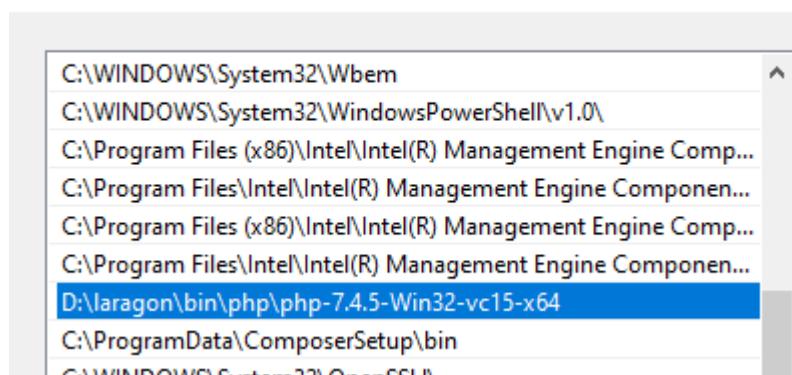
- Dubbelklik op Path
- Je ziet nu: omgevingsvariabele bewerken



Nu zie je bij mij, dat ik ooit Xampp heb gehad. Deze php versie gebruikt Windows op dit moment. Als je Xampp niet meer gebruikt, kan je deze wijzigen

Ik verander deze naar php 7.4.5

Omgevingsvariabele bewerken



Je kan nu OK klikken. Daarna de rest van de schermen ook op OK klikken, anders kan het misschien verkeerd gaan.

Nu ben ik geen Windows expert, dus misschien kan je een path makkelijker opnieuw inladen. De manier die ik doe, is gewoon een reboot.

Na de reboot, kan ik opnieuw controleren in PhpStorm wat de versie van php is.

```
D:\laragon\www\workshop>php -v
PHP 7.4.5 (cli) (built: Apr 14 2020 16:17:34) ( ZTS Visual C++ 2017 x64 )
Copyright (c) The PHP Group
Zend Engine v3.4.0, Copyright (c) Zend Technologies
```

Het is nu netjes 7.4.5

## 10.2. Composer update

Om de nieuwste versie van Laravel te gebruiken heb je minimaal composer versie 2 nodig. Mocht het zijn dat je de server omgeving of composer al een stuk eerder heb geïnstalleerd, kan het zijn dat dit nog een oude versie is.

```
D:\laragon\www
\ composer

[Progress Bar]

Composer version 1.8.6 2019-06-11 15:03:05

Usage:
  command [options] [arguments]
```

Als je dan naar versie 2 wilt gaan, zou je zeggen dat een self-update zou werken. Dit kan je uitvoeren met composer self-update

```
D:\laragon\www
\ composer self-update
Updating to version 2.0.9 (stable channel).
  Downloading (100%)
Use composer self-update --rollback to return to version 1.8.6
```

Als er opnieuw wordt gekeken naar de versie zie je dat versie 2.0.9 meteen werkt.

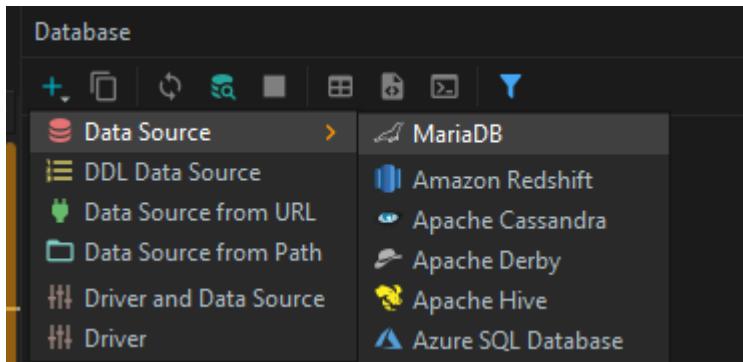
```
D:\laragon\www
\ composer

[Progress Bar]

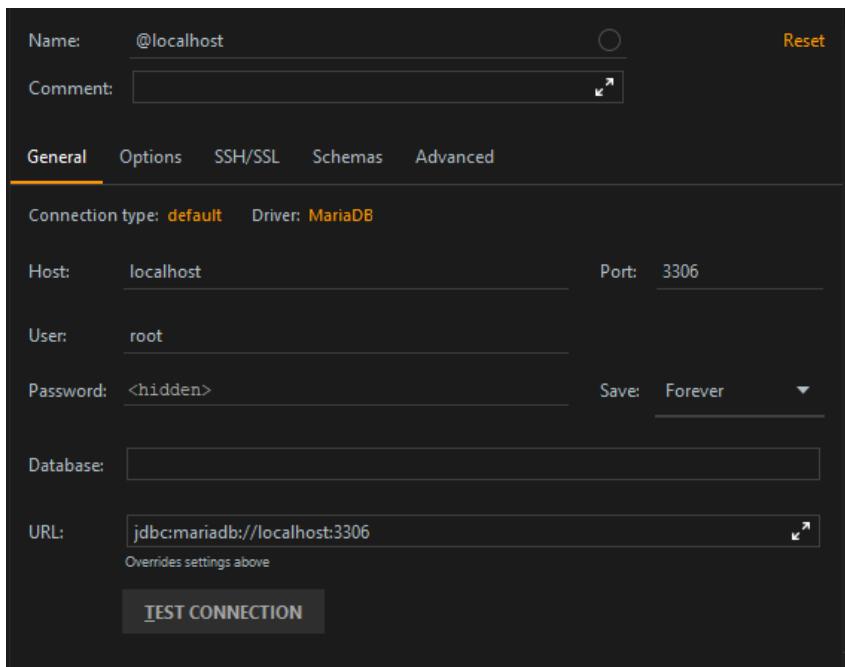
Composer version 2.0.9 2021-01-27 16:09:27
```

## 10.3. Database in PhpStorm

Binnen PhpStorm is een plugin om de database te kunnen zien. Als je de plugin hebt geïnstalleerd krijg je aan de rechterkant de optie voor database. We hebben dan een data source nodig en die van ons is van MariaDB.

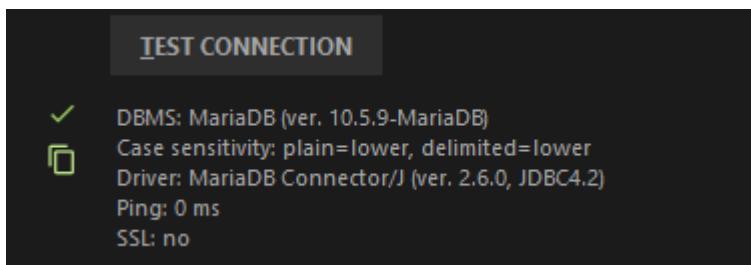


Dan krijg je een scherm om alles in te stellen



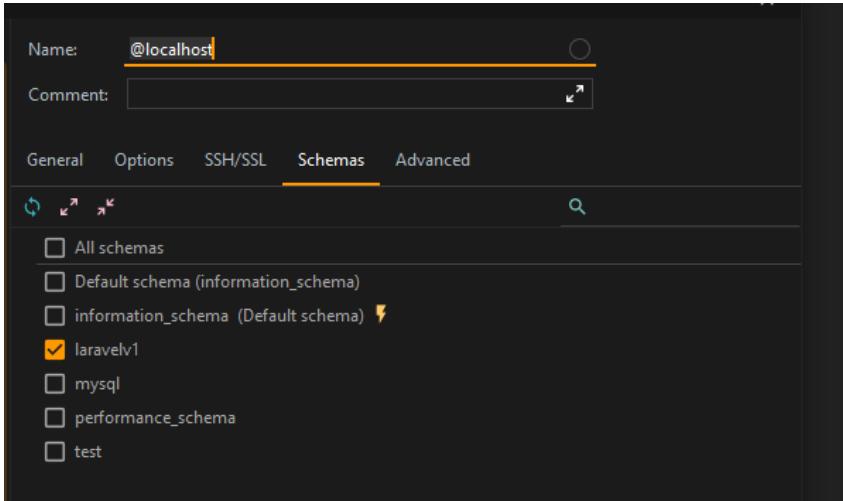
Let op, onder ‘Test Connection’ kan het zijn dat er een melding staat dat je een driver mist. Door op de link te klikken download phpstorm de juiste driver voor je.

Vul bij User de database gebruiker in “root”. We hebben op dit moment geen wachtwoord. Je kan nog even de verbinding testen door op test connection te klikken.



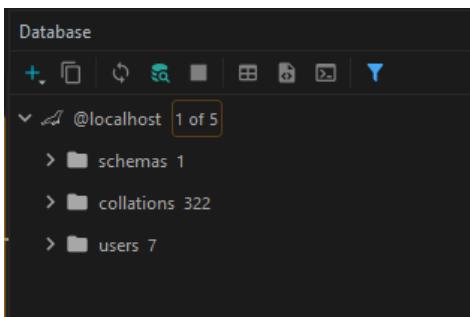
Hierboven zie je dat het gelukt is.

Zorg dan dat je naar het tabblad Schemas gaat:

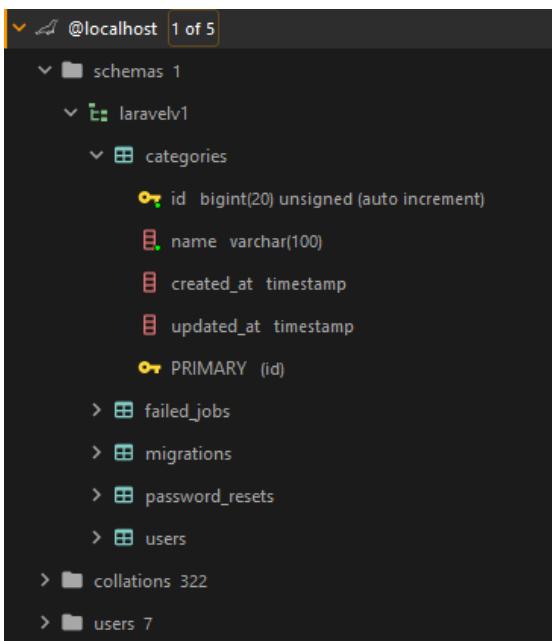


Kies hier dan voor de database die je gemaakt hebt. Voor nu laravelv1.

Als je op Apply klikt, zie je in de rechterkolom dit verschijnen:



Je kan nu alles uitvouwen van het schema en je ziet netjes een categories tabel.



## 10.4. NPM errors met Laragon

Het kan zijn, dat als je laragon gebruikt je errors krijgt met npm.

```
D:\laragon\www\laravel85
\ npm run dev

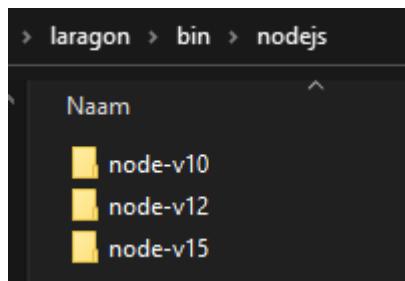
> @ dev D:\laragon\www\laravel85
> npm run development

> @ development D:\laragon\www\laravel85
> mix

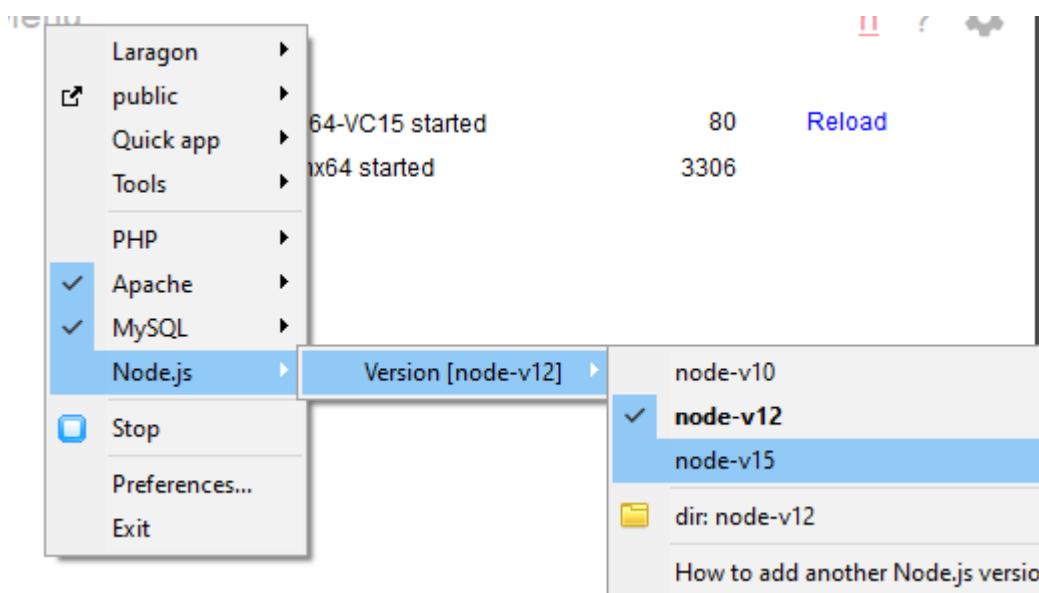
Error: You are using an unsupported version of Node. Please update to at least Node v12.14
  at assertSupportedNodeVersion (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
    at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
      at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
        at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
          at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
            at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
              at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                  at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                    at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                      at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                        at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                          at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                            at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                              at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                  at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                    at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                      at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                        at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                          at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                            at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                              at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                  at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                    at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                      at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                        at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                          at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                            at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                              at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                  at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                    at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                      at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                        at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                          at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                            at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                              at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                                at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                                  at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                                    at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                                      at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                                        at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                                          at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                                            at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                                              at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                                                at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                                                  at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                                                    at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                                                      at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                                                        at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                                                          at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                                                            at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                                                              at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                                                                at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                                                                  at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
                                                                                                                    at Object. (D:\laragon\www\laravel85\node_modules\laravel-mix\laravel-mix\index.js:11:11)
................................................................
npm ERR! code ELIFECYCLE
```

Er komen nu allerlei errors dat de npm versie geupdate moet worden.

Indien je deze melding krijgt, ga naar <https://nodejs.org/en/> en download de laatste versie van npm. Installeer deze in de map laragon/bin/nodejs



Zorg dan dat je in het Laragon menu de nieuwste versie van node kiest. Nu is dat v15.



Herstart je computer na de instelling, anders gaat het nog niet werken. Dit komt omdat windows een PATH inlaad bij het starten van windows.

Als je dit allemaal hebt gedaan, kan je opnieuw doen: npm run dev

```
D:\laragon\www\laravel85
λ npm run dev

> dev
> npm run development

> development
> mix

10% building import loader ./node_modules/css-loader/dist/cjs.js|
```

Als resultaat komt er dit:

```
Laravel Mix v6.0.11

✓ Compiled Successfully in 3856ms
File           Size
/js/app.js      671 KiB
css/app.css    3.75 MiB

D:\laragon\www\laravel85
```