

Laravel Project



Laravel

Marcel Koningstein

Techniek College Rotterdam locatie Spijkenisse



Inhoud

| | |
|---|-----|
| Inleiding | 5 |
| Versiebeheer | 6 |
| Omgeving | 7 |
| Installatie | 9 |
| Casus beschrijving | 13 |
| Categories | 14 |
| Model | 14 |
| Migration | 16 |
| Factory | 20 |
| Seeder | 22 |
| Lay-out Masterpage | 26 |
| Resource Controller | 33 |
| Aanmaken van de resource controller | 33 |
| Routing | 36 |
| Index in controller | 38 |
| Index in View | 39 |
| Van url naar scherm | 42 |
| Create | 45 |
| Store | 50 |
| Validatie in de store | 53 |
| Show | 59 |
| Edit | 64 |
| Update | 68 |
| Delete | 71 |
| Destroy | 75 |
| Rollen en Permissies | 76 |
| Role and Permission Seeder | 79 |
| User Seeder | 81 |
| Middleware gebruiken in Routes | 84 |
| Auth binnen onze eigen layout | 87 |
| Permissies in de Controller | 91 |
| Product & Price | 96 |
| Models | 97 |
| Migrations | 100 |
| Factory | 103 |
| Seed | 104 |

| | |
|--------------------------------------|-----|
| Product Controller | 110 |
| Index | 110 |
| Create | 116 |
| Store | 119 |
| Show | 125 |
| Edit | 129 |
| Update | 131 |
| Delete | 134 |
| Destroy | 137 |
| Category Test..... | 139 |
| Unit & Feature testing | 142 |
| Omgeving | 142 |
| Gebruik van Pest / Phpunit | 146 |
| Response Status | 148 |
| Schrijven van een test | 149 |
| Feature test | 152 |
| Dataset | 155 |
| Gebruik van een database | 157 |
| Naamgeving en Setup | 160 |
| Product Tests | 162 |
| Product Index | 163 |
| Extra Testing Configuratie..... | 164 |
| Verder met Product Index Test..... | 168 |
| Product Create | 172 |
| Product Store | 174 |
| ProductStoreCheck met phpunit | 177 |
| Onderzoek naar status fout..... | 183 |
| Product edit..... | 190 |
| Product update..... | 192 |
| ProductUpdateCheck met phpunit | 195 |
| Product Delete | 200 |
| Product Destroy | 202 |
| Unit Testing | 204 |
| Wijzigingen en testen | 206 |
| TDD met orders..... | 210 |
| Index | 210 |
| Order Unit Test | 223 |

| | |
|--|-----|
| Verder met TDD van Index..... | 224 |
| Create..... | 229 |
| Store..... | 234 |
| Show..... | 240 |
| Edit..... | 241 |
| Update..... | 249 |
| Delete | 252 |
| Destroy..... | 258 |
| Checklist | 261 |
| Omgeving..... | 261 |
| Installatie | 261 |
| Permissions..... | 261 |
| Data (voorbeeld met een basis webshop) | 262 |
| Categorie | 262 |
| Product..... | 262 |
| Price..... | 263 |
| Order | 263 |
| Orderrow | 264 |
| Reviews..... | 264 |
| Tips & Tricks..... | 265 |
| Gebruik van phpstorms terminal met laragon | 265 |
| Uitbreidingen | 268 |
| Per validatie een aparte message | 268 |
| Prefix voor resource controller | 268 |

Inleiding

Dit is een lessenserie om stap voor stap de basis van Laravel te leren. Bij het schrijven van deze lessenserie is de huidige versie van Laravel 7. In de lessenserie leer je hoe je de omgeving kan opzetten, de basisvaardigheden van Laravel en hoe je een werkelijk project kan maken.

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/>

Ik verwacht dat de volgende onderdelen al geïnstalleerd zijn:

- Laragon: <https://laragon.org/>
- PhpStorm: <https://www.jetbrains.com/phpstorm/>
- Git: <https://git-sem.com/>
- Composer: <https://getcomposer.org/>

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)

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 unittesting |
| 2.1 | Uitbreiding | 29-5-2020 | Onderdeel toegevoegd met unit & feature testing |
| 2.2 | Uitbreiding | 22-6-2020 | Onderdeel toegevoegd met test driven development |

Omgeving

Als eerst gaan we aan de slag met de installatie van Laravel. 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. Wel is voor Laragon gekozen omdat het uiteindelijk erg makkelijk is.

(<https://laravel.com/docs/7.x>)

Ik installeer het op een Windows 10 laptop, omdat de meeste studenten dit gebruiken. Heb je Mac, ga naar Valet kijken (<https://laravel.com/docs/7.x/valet>)

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 7.2.5 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

VC15 x64 Thread Safe (2020-Apr-14 22:48:50)

- [Zip \[24.9MB\]](#)

sha256: 89f699dae7c67b164ddb9094a1383324b48814a42bbd222c771c0ff474eabf1b

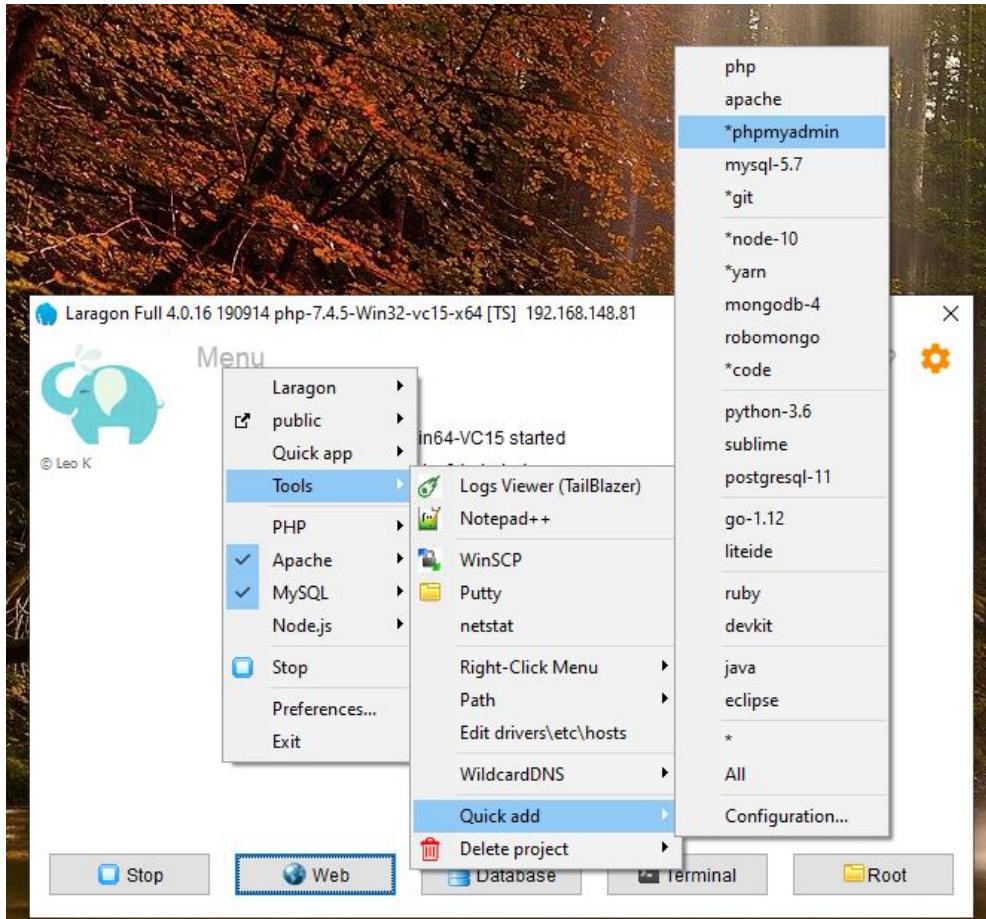
Deze pak ik uit in Laragon\bin\php

| Deze pc → SSD1b (D:) → laragon → bin → php | | |
|--|---------|------------|
| Naam | Grootte | Aanpassing |
| php-7.2.19-Win32-VC15-x64 | 13 MB | |
| php-7.4.0-Win32-vc15-x64 | 13 MB | |
| php-7.4.5-Win32-vc15-x64 | 22 MB | |

Php is nu up to date.

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 nieuwere 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.

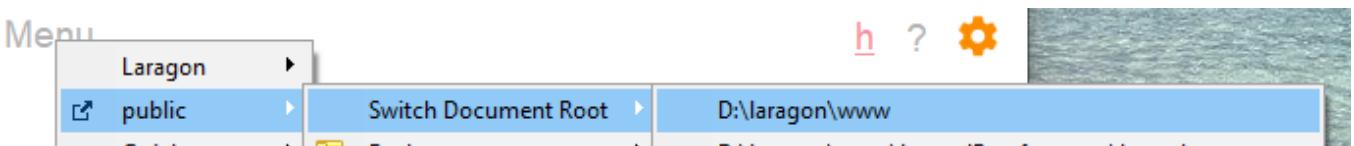
| File | Size | Verification |
|---|---------|----------------|
| phpMyAdmin-4.9.5-all-languages.zip | 10.6 MB | [PGP] [SHA256] |
| phpMyAdmin-4.9.5-all-languages.tar.gz | 9.7 MB | [PGP] [SHA256] |

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.

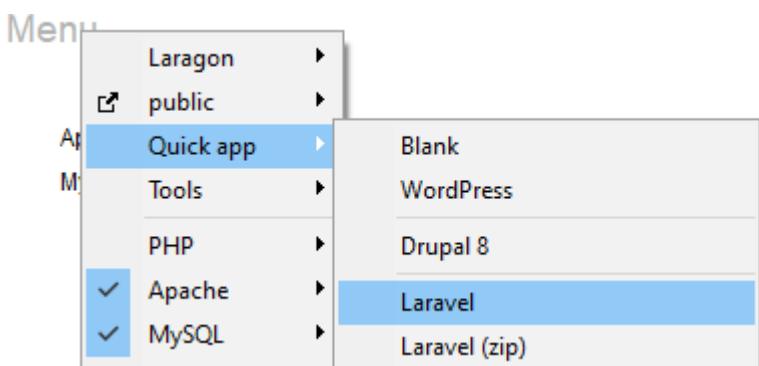
Installatie

In Laragon zit een mooie optie om een Laravel project aan te maken. Deze gaan we dan ook gebruiken.

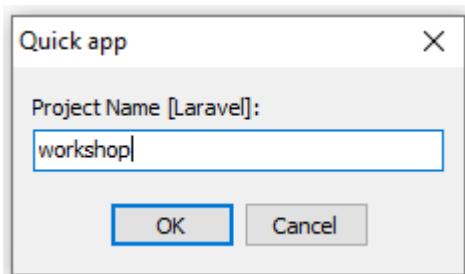
Voordat je dit doet. Als je al eerder met Laragon hebt gewerkt, check even of de webroot wel op Laragon/www staat. Heb je net Laragon geïnstalleerd, staat hij hier standaard op.



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]

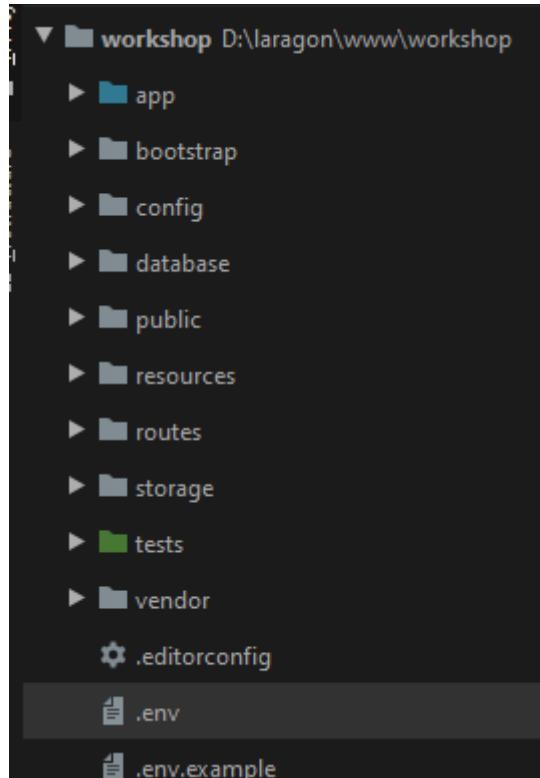
**** Hint: In Terminal, you can type:
-----
d D:\laragon\www
omposer create-project laravel/laravel workshop --prefer-dist
-----

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 phopoption/phopoption (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  
-----
```

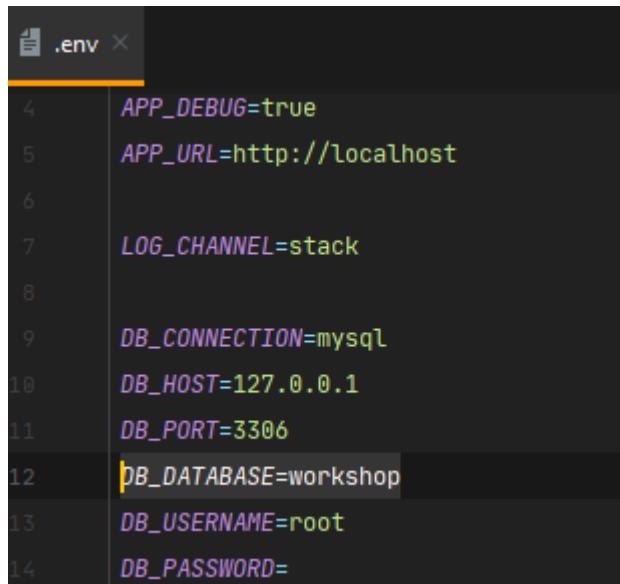
Het project staat netjes in de map www/workshop/



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.

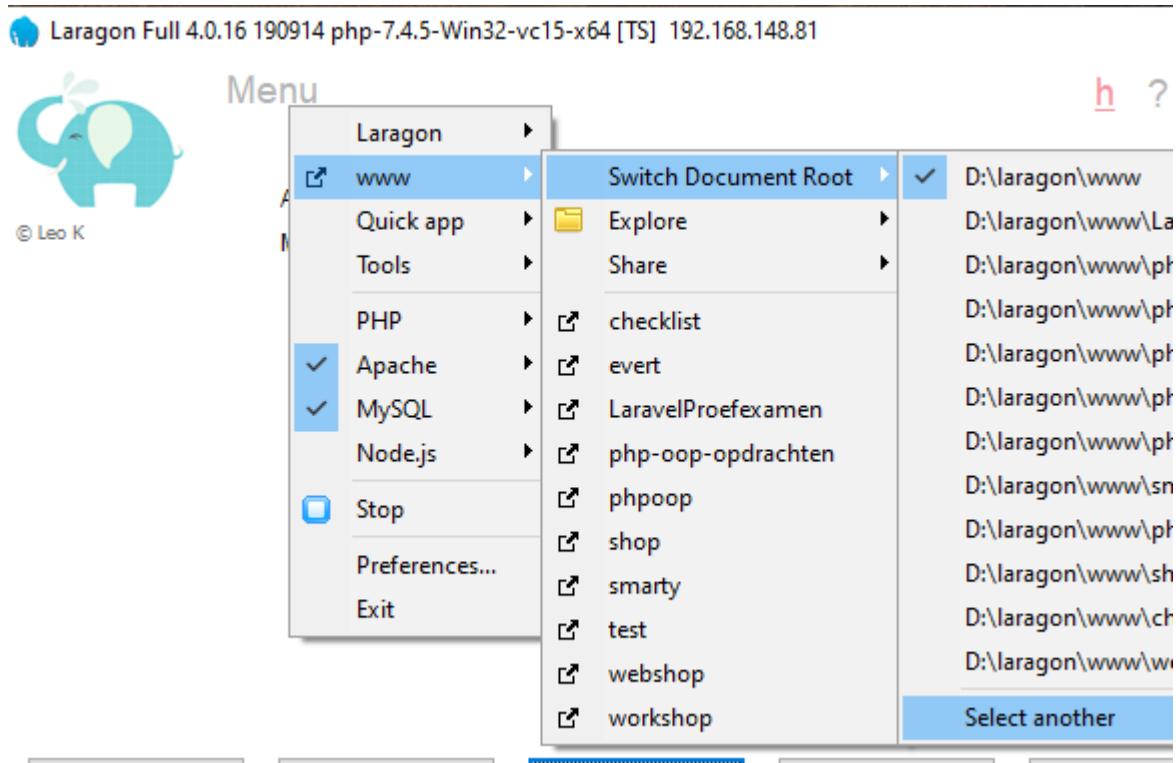
In de root directory van ons project vinden we een .env bestand. Dit wordt gebruikt als een environment bestand, waarin alle instellingen staan. Hier veranderen we de DB_DATABASE van Laravel naar workshop. Dit zodat de workshop database gebruikt gaat worden.



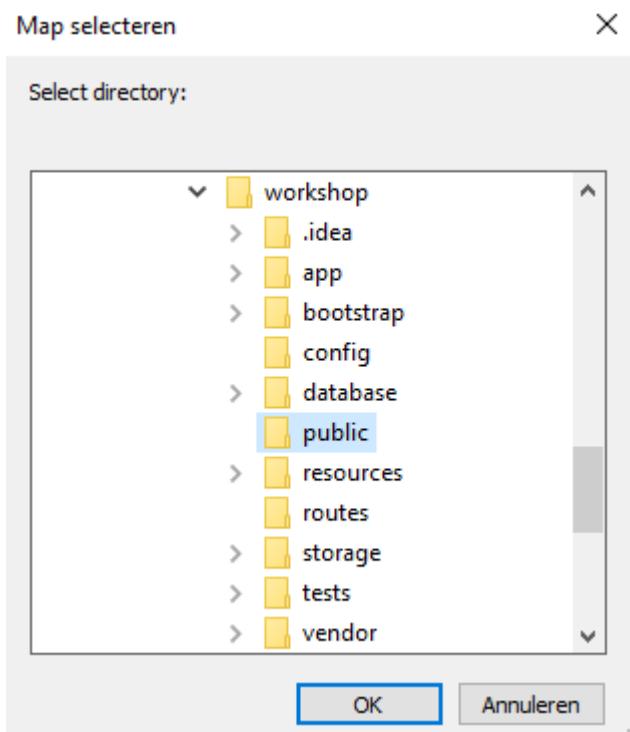
```
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8
9 DB_CONNECTION=mysql
10 DB_HOST=127.0.0.1
11 DB_PORT=3306
12 DB_DATABASE=workshop
13 DB_USERNAME=root
14 DB_PASSWORD=
```

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.

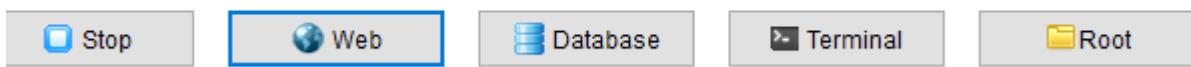
Om ervoor te zorgen dat we de Laravel website nu gaan zien, gaan we de document root van Laragon veranderen. Hiervoor kiezen we ervoor dat we een andere map willen met menu->www->Switch Document Root->Select another.



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)

DOCS LARACASTS NEWS BLOG NOVA FORGE VAPOR GITHUB

Onze Laravel installatie is tot zover nu klaar.

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.

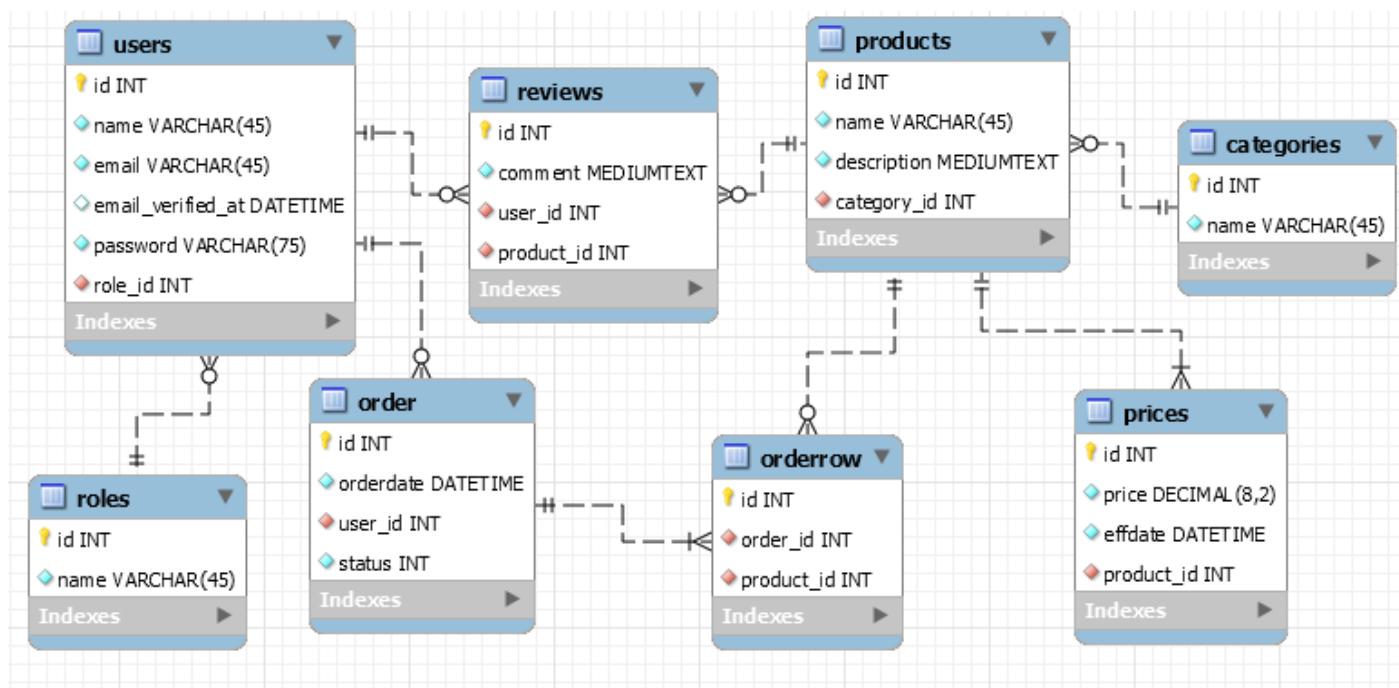
We gaan een simpele webshop maken voor spellen. Hierbij kunnen gebruikers:

- 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.



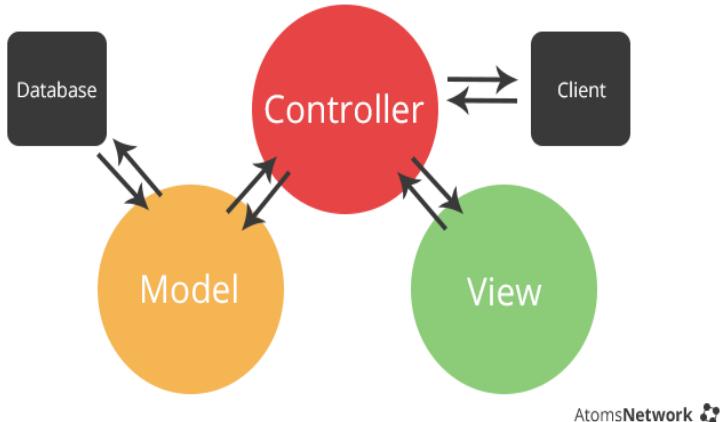
Om ervoor te zorgen dat je alle principes geleerd krijgt gaan we de website stap voor stap maken. De users worden door Laravel al voor een gedeelte geregeld. Daarnaast gaan we gebruik maken van een package die de rollen en permissies gaat regelen

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.

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:\laragon\www\workshop
λ php artisan help make:model
Description: Create a new Eloquent model class
    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 gaan regelen

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 this help message
  -q, --quiet     Do not output any message
  -V, --version    Display this application version
  --ansi          Force ANSI output
  --no-ansi       Disable ANSI output
  -n, --no-interaction Do not ask any interactive question
  --env[=ENV]      The environment the command should run under
  -v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
```

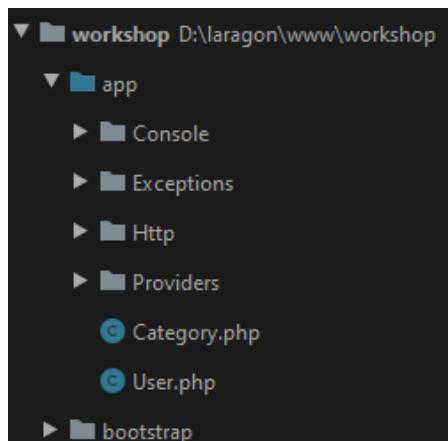
Nu kan je de makkelijkste optie gebruiken, namelijk de `--all` optie erbij zetten. Deze zorgt dat je alles krijgt wat je nodig hebt voor een CRUD. Nu in het begin gebruiken we deze nog even niet om juist even alle onderdelen los te leren, maar later in de lessenserie, bij de andere tabellen, gaan we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

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

```
D:\laragon\www\workshop
λ php artisan make:model Category
Model created successfully.

D:\laragon\www\workshop
```

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



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/7.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:\laragon\www\workshop
λ 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
  -h, --help         Display this help message
  -q, --quiet        Do not output any message
  -V, --version      Display this application version
  --ansi            Force ANSI output
  --no-ansi          Disable ANSI output
  -n, --no-interaction  Do not ask any interactive question
  --env[=ENV]        The environment the command should run under
  -v|vv|vvv, --verbose  Increase the verbosity of messages: 1 for normal output, 2 for more verbose
  tput and 3 for debug
```

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:

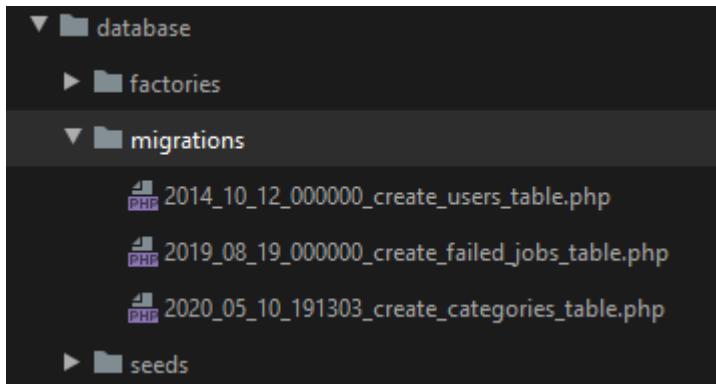
`Php artisan make:migration --create=categories create_categories_table`

```
D:\laragon\www\workshop
λ php artisan make:migration --create=categories create_categories_table
Created Migration: 2020_05_10_191303_create_categories_table
```

Bij dit commando maken we dus een migration aan, waar de tabel `products` 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 meervoudig, dus de benaming van de migration is ook meervoudig bij categories)

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 2 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 is kijken naar de categories migration.

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

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:\laragon\www\workshop
λ php artisan migrate
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (0.02 seconds)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (0.01 seconds)
Migrating: 2020_05_10_191303_create_categories_table
Migrated: 2020_05_10_191303_create_categories_table (0.01 seconds)
```

Er worden 3 tabellen aangemaakt, users, 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 |
|--------------------------|-------------------|--|-------|-----------------|
| <input type="checkbox"/> | categories | Verkennen Structuur Zoeken Invoegen Legen Verwijderen | 0 | InnoDB |
| <input type="checkbox"/> | failed_jobs | Verkennen Structuur Zoeken Invoegen Legen Verwijderen | 0 | InnoDB |
| <input type="checkbox"/> | migrations | Verkennen Structuur Zoeken Invoegen Legen Verwijderen | 3 | InnoDB |
| <input type="checkbox"/> | users | Verkennen Structuur Zoeken Invoegen Legen Verwijderen | 0 | InnoDB |
| | 4 tabellen | Som | | 3 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.

| | Wijzigen | Kopiëren | Verwijderen | id | migration | batch |
|--------------------------|----------|----------|-------------|----|--|-------|
| <input type="checkbox"/> | | | | 1 | 2014_10_12_000000_create_users_table | 1 |
| <input type="checkbox"/> | | | | 2 | 2019_08_19_000000_create_failed_jobs_table | 1 |
| <input type="checkbox"/> | | | | 3 | 2020_05_10_191303_create_categories_table | 1 |

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:\laragon\www\workshop
└ 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 this help message
-q, --quiet              Do not output any message
-V, --version             Display this application version
--ansi                  Force ANSI output
--no-ansi                Disable ANSI output
-n, --no-interaction     Do not ask any interactive question
--env[=ENV]               The environment the command should run under
-v|vv|vvv, --verbose     Increase the verbosity of messages: 1 for normal output,
2 for more verbose output and 3 for debug
```

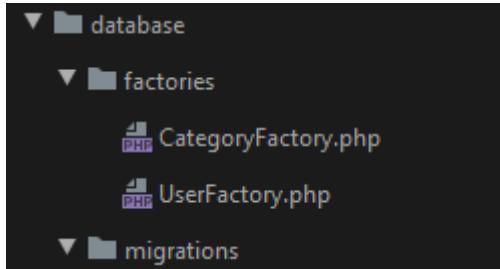
Als we de model meegeven bij het aanmaken van de factory zal er meteen een stuk extra code voor je gemaakt worden. Handig! Dit gaan we dan ook doen.

```
D:\laragon\www\workshop
└ 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.

```
1  k?php
2
3  /** @var \Illuminate\Database\Eloquent\Factory $factory */
4
5  use App\Category;
6  use Faker\Generator as Faker;
7
8  $factory->define( Category::class, function (Faker $faker) {
9      return [
10          //
11      ];
12  });

```

Zoals je in de code al ziet staat er automatisch al bij dat we de class Faker gebruiken.

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.

```
$factory->define( class: Category::class, function (Faker $faker) {
    return [
        // ...
        'name' => $faker->name
    ];
});
```

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.

```
$factory->define( class: Category::class, function (Faker $faker) {
    return [
        // ...
        'name' => $faker->name,
        'created_at' => $faker->dateTimeThisDecade( max: 'now', timezone: 'Europe/Amsterdam'),
        'updated_at' => $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.

Seeder

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

```
D:\laragon\www\workshop
λ 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 this help message
-q, --quiet     Do not output any message
-V, --version   Display this application version
--ansi         Force ANSI output
--no-ansi      Disable ANSI output
-n, --no-interaction  Do not ask any interactive question
--env[=ENV]     The environment the command should run under
-v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output,
2 for more verbose output and 3 for debug
```

We kunnen nu dus niet echt wat extra's doen bij de seeder.

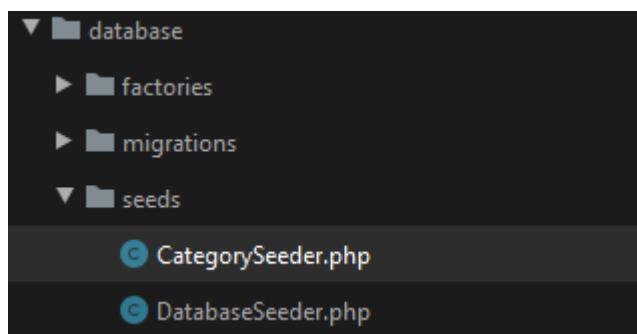
We gaan dus de seeder voor Categorie nu maken.

```
D:\laragon\www\workshop
└ 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/seeds 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 use Illuminate\Database\Seeder;
4
5 class CategorySeeder extends Seeder
6 {
7     /**
8      * Run the database seeds. ...
9     */
10    public function run()
11    {
12        //
13    }
14}
15
16}
```

Om ervoor te zorgen dat we nu vulling krijgen in de database moeten we regel 16 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 4 is ook deze toegevoegd zodat de class CategorySeeder weet dat we deze gebruiken.

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

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 15.

```
1  <?php
2
3  use Illuminate\Database\Seeder;
4
5  class DatabaseSeeder extends Seeder
6  {
7      /**
8       * Seed the application's database.
9       *
10      * @return void
11     */
12
13     public function run()
14     {
15         // $this->call(UserSeeder::class);
16         $this->call( CategorySeeder::class);
17     }
18 }
```

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

```
D:\laragon\www\workshop
λ php artisan db:seed
Seeding: CategorySeeder
Seeded: CategorySeeder (0.05 seconds)
Database seeding completed successfully.
```

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

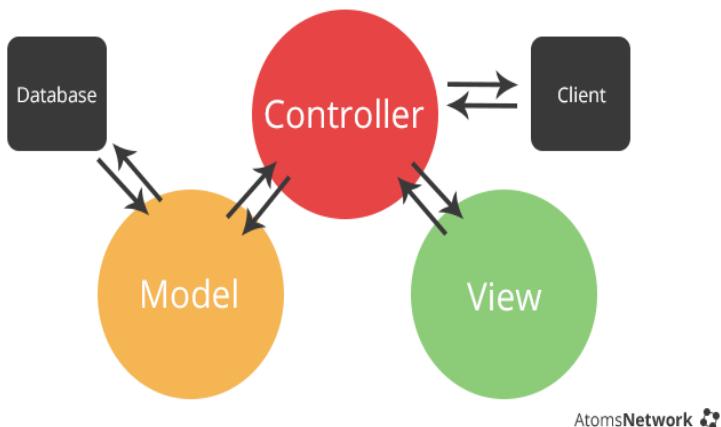
| | | | id | name | created_at | updated_at | | | | |
|--------------------------|--|--------------------------|-----------|--------------------------|-------------------|-----------------------------|----|----------------------|---------------------|---------------------|
| <input type="checkbox"/> | | Wijzigen | | Kopiëren | | Verwijderen | 1 | Dr. Bruce Effertz | 2018-06-30 07:16:16 | 2011-05-02 20:31:46 |
| <input type="checkbox"/> | | Wijzigen | | Kopiëren | | Verwijderen | 2 | Jimmy Nolan | 2012-12-21 23:56:31 | 2010-09-08 10:38:22 |
| <input type="checkbox"/> | | Wijzigen | | Kopiëren | | Verwijderen | 3 | Cora Macejkovic | 2016-02-19 20:38:01 | 2011-05-01 08:41:22 |
| <input type="checkbox"/> | | Wijzigen | | Kopiëren | | Verwijderen | 4 | Jessica Murray I | 2013-07-13 17:55:03 | 2016-05-03 22:13:19 |
| <input type="checkbox"/> | | Wijzigen | | Kopiëren | | Verwijderen | 5 | Sophie Wunsch DDS | 2019-11-18 04:30:00 | 2016-05-02 08:05:51 |
| <input type="checkbox"/> | | Wijzigen | | Kopiëren | | Verwijderen | 6 | Prof. Kylee Bayer IV | 2020-05-08 01:57:29 | 2018-05-10 10:58:04 |
| <input type="checkbox"/> | | Wijzigen | | Kopiëren | | Verwijderen | 7 | Wilton Wehner | 2011-02-23 06:18:28 | 2015-05-08 09:39:42 |
| <input type="checkbox"/> | | Wijzigen | | Kopiëren | | Verwijderen | 8 | Kaden Kuphal I | 2016-11-02 05:46:44 | 2012-03-25 16:34:54 |
| <input type="checkbox"/> | | Wijzigen | | Kopiëren | | Verwijderen | 9 | Lucius Johnson | 2013-09-13 08:42:45 | 2016-12-03 03:27:25 |
| <input type="checkbox"/> | | Wijzigen | | Kopiëren | | Verwijderen | 10 | Laura Howell | 2013-05-30 12:38:49 | 2011-05-23 22:05:32 |

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.

Lay-out Masterpage

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.

Om nu niet de focus op de html te leggen, gaan we simpel beginnen met een lay-out met Bootstrap.
(<https://getbootstrap.com/>)

We gaan de lay-out niet zo downloaden en in een mapje zetten, maar doen dit op de manier zoals we met andere onderdelen gaan doen, namelijk via composer. Je kan dit vinden op <https://laravel.com/docs/7.x/frontend>

Als eerst gaan we ervoor zorgen dat we laravel/ui binnenhalen met composer.

```
D:\laragon\www\workshop
λ composer require laravel/ui
Using version ^2.0 for laravel/ui
./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/ui (v2.0.3): Loading from cache
Writing lock file
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
Discovered Package: facade/ignition
Discovered Package: fideloper/proxy
Discovered Package: fruitcake/laravel-cors
Discovered Package: laravel/tinker
Discovered Package: laravel/ui
Discovered Package: nesbot/carbon
Discovered Package: nunomaduro/collision
Discovered Package: spatie/laravel-permission
Package manifest generated successfully.
```

Omdat we met Bootstrap willen werken gaan we dit aangeven in Laravel UI. We beginnen dan ook met basic scaffolding. (Ja, zo heet het gewoon)

De standaard scaffolding kan je doen met `php artisan ui bootstrap`

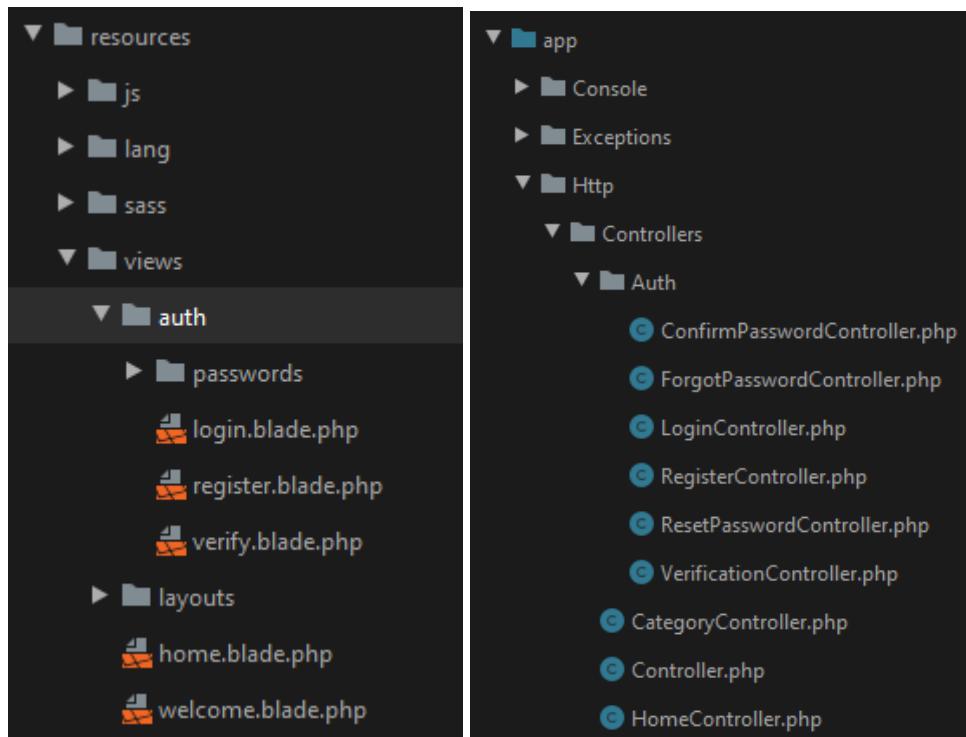
Alleen kan je nog meer doen dan de standaard. Je kan namelijk ook meteen het hele login systeem laten maken. Dit gaan we binnen ons project nodig hebben, dus gaan we het in een keer goed doen.

We gebruiken dan `php artisan ui bootstrap --auth`

```
D:\laragon\www\workshop
λ php artisan ui bootstrap --auth
Bootstrap scaffolding installed successfully.
Please run "npm install && npm run dev" to compile your fresh scaffolding.
Authentication scaffolding generated successfully.
```

Je ziet dat door deze stap allerlei bestanden er al zijn bijgekomen. In de views map zijn al schermen gemaakt voor allerlei onderdelen die te maken hebben met de login, zoals login, registreren, forgot password etc.

Verder zie je ook in de map controllers, waar we later zelf ook in gaan werken voor de functionaliteit, allerlei bestanden staan.



Wel zie je in de terminal dat je nog npm install en npm run dev moet doen. Dit gaan we dan ook doen.

Als eerst `npm install`

```
D:\laragon\www\workshop
λ npm install
npm [WARN] deprecated popper.js@1.16.1: You can find the new Popper v2 at @popperjs/core, this package is dedicated to the legacy v1
npm [WARN] deprecated chokidar@2.1.8: Chokidar 2 will break on node v14+. Upgrade to chokidar 3 with 15x less dependencies.
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 [notice] created a lockfile as package-lock.json. You should commit this file.
npm [WARN] optional SKIPPING OPTIONAL DEPENDENCY: fsevents@^1.2.7 (node_modules\chokidar\node_modules\fsevents):
npm [WARN] notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.13: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm [WARN] sass-loader@8.0.2 requires a peer of node-sass@^4.0.0 but none is installed. You must install peer dependencies yourself.
npm [WARN] sass-loader@8.0.2 requires a peer of fibers@>= 3.1.0 but none is installed. You must install peer dependencies yourself.

added 1098 packages from 494 contributors in 19.279s
```

Je ziet dat er van alles aan het project wordt toegevoegd. Er is ook een nieuwe map aangemaakt, namelijk `node_modules`.

```
▶ └── database
▶ └── node_modules library root
▶ └── public
```

Daarna doen we `npm run dev`

```
D:\laragon\www\workshop
λ npm run dev

> @ dev D:\laragon\www\workshop
> npm run development

> @ development D:\laragon\www\workshop
> cross-env NODE_ENV=development node_modules/webpack/bin/webpack.js --progress --hide-modules --config=node_modules/laravel-mix/setup/webpack.config.js
```

Uiteindelijk worden 2 bestanden gecompileerd, namelijk de `app.css` en `app.js`

Je kan deze bestanden terugvinden in de `css` en `js` map, binnen de `public` map.

```
DONE Compiled successfully in 4117ms

      Asset      Size  Chunks             Chunk Names
/css/app.css   177 KiB /js/app  [emitted]  /js/app
  /js/app.js   1.07 MiB /js/app  [emitted]  /js/app

D:\laragon\www\workshop
```

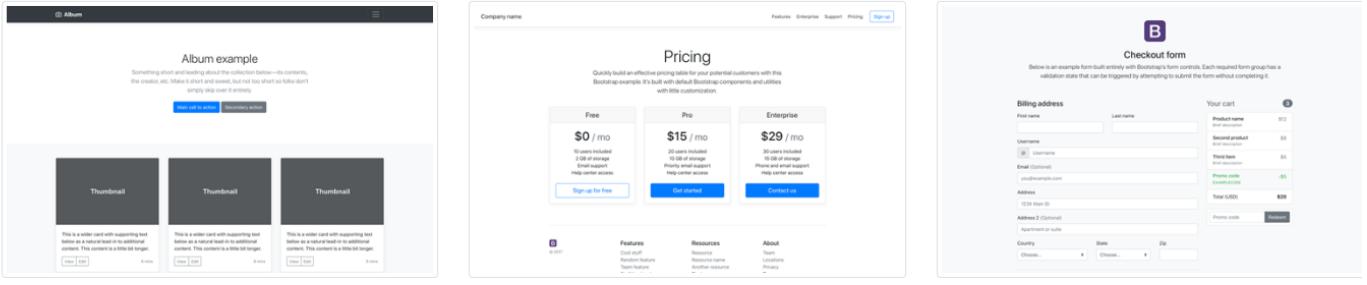
Nu wordt het tijd om onze lay-out te kiezen. Dit doen we op de website van bootstrap (<https://getbootstrap.com/docs/4.5/examples/>)



The screenshot shows the Bootstrap Examples page with a purple header containing a logo and navigation links for Home, Documentation, Examples, Icons, Themes, Expo, and Blog. Below the header, there's a section titled 'Examples' with a sub-section titled 'Album'. It features a grid of three cards, each labeled 'Thumbnail'. A 'Download source code' button is visible at the bottom left of this section.

Custom components

Brand new components and templates to help folks quickly get started with Bootstrap and demonstrate best practices for adding onto the framework.



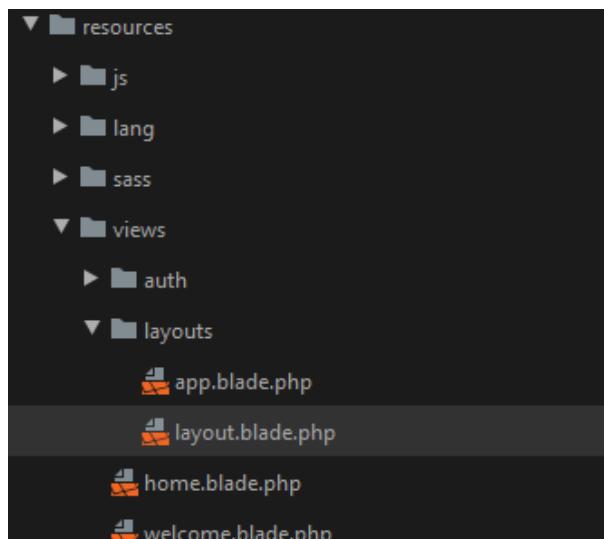
The screenshot displays three examples of custom Bootstrap components:

- Album**: A one-page template for photo galleries, portfolios, and more. It features a grid of three cards, each labeled 'Thumbnail'.
- Pricing**: An example pricing page built with Cards and featuring a custom header and footer.
- Checkout**: A custom checkout form showing form components and their validation features.

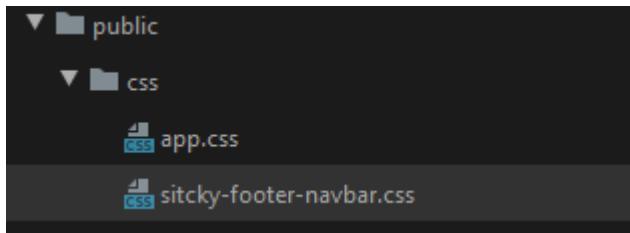
Mijn keuze is gevallen op de sticky footer lay-out, omdat deze al weinig onderdelen in de content heeft. (<https://getbootstrap.com/docs/4.4/examples/sticky-footer-navbar/>)

Als je dan naar de broncode gaat en dat allemaal kopieert naar een nieuw bestand in het project: `resources/views/layouts/layout.blade.php`

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 `layout.blade.php` te noemen.



In de broncode staat ook nog een sticky-footer-navbar.css, deze code zal ook gekopieerd moeten worden en in een nieuw bestand in `public/css/sticky-footer-navbar.css` gezet moeten worden.



Daarin staat dan de css zoals bij het voorbeeld op de bootstrap pagina.

```
/* Custom page CSS
   ----- */
/* Not required for template or sticky footer method. */

main > .container {
    padding: 60px 15px 0;
}

.footer {
    background-color: #f5f5f5;
}

.footer > .container {
    padding-right: 15px;
    padding-left: 15px;
}

code {
    font-size: 80%;
}
```

Nu gaan we ervoor zorgen dat de lay-out geheel gaat werken. Hiervoor moeten we naar elk path in de broncode gaan kijken. Op dit moment zie je in de <head> dit staan:

```
<link rel="canonical" href="https://getbootstrap.com/docs/4.4/examples/sticky-footer-navbar/">

<!-- Bootstrap core CSS -->
<link href="/docs/4.4/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-Vkoo8x4CGs03+Hhxv" type="text/css">

<!-- Favicons -->
<link rel="apple-touch-icon" href="/docs/4.4/assets/img/favicons/apple-touch-icon.png" sizes="180x180" type="image/png">
<link rel="icon" href="/docs/4.4/assets/img/favicons/favicon-32x32.png" sizes="32x32" type="image/png">
<link rel="icon" href="/docs/4.4/assets/img/favicons/favicon-16x16.png" sizes="16x16" type="image/png">
<link rel="manifest" href="/docs/4.4/assets/img/favicons/manifest.json">
<link rel="mask-icon" href="/docs/4.4/assets/img/favicons/safari-pinned-tab.svg" color="#563d7c">
<link rel="icon" href="/docs/4.4/assets/img/favicons/favicon.ico">
<meta name="msapplication-config" content="/docs/4.4/assets/img/favicons/browserconfig.xml">
<meta name="theme-color" content="#563d7c">
```

De Favicons kunnen weg. Het CSS-bestand gaan we anders linken. We veranderen dit in:

```
9      <title>Sticky Footer Navbar Template · Bootstrap</title>
10
11      <!-- Bootstrap core CSS -->
12      <link href="{{asset('css/app.css')}}" rel="stylesheet">
13
14      <style>
15          .bd-placeholder-img {
16              font-size: 1.125rem;
17              text-anchor: middle;
18              -webkit-user-select: none;
19              -moz-user-select: none;
20              -ms-user-select: none;
21              user-select: none;
22
23
24          @media (min-width: 768px) {
25              .bd-placeholder-img-lg {
26                  font-size: 3.5rem;
27              }
28          }
29      </style>
30
31      <!-- Custom styles for this template -->
32      <link href="{{asset('css/sticky-footer-navbar.css')}}" rel="stylesheet">
33
34  </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

Alle Javascript onderdelen onder in de lay-out haal ik standaard weg als het binnen school wordt gebruikt.

```
74     </footer>
75     <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa4849blE2+poT4WnyKhv5vI5Vd9eC5e0Hfcrv6+Xn&gt;</script>
76     <script>window.jQuery || document.write('<script src="/docs/4.4/assets/js/vendor/jquery.slim.min.js"></script>');
77   </html>
```

We gaan hier alleen de standaard bootstrap javascript inladen.

```
74     </footer>
75     <script src="{{asset('js/app.js')}}"></script>
76   </body>
```

(Kleine hint, let op dat je het eind van de body niet weghaalt! Deze staat namelijk achter het javascript geplakt)

De masterpage is bijna klaar. Alleen de plek waar onze content komt moet nog bepaald worden. Dit doen we natuurlijk in de <main>. De main haal ik geheel leeg en zet er alleen neer:

```
61     <!-- Begin page content -->
62     <main role="main" class="flex-shrink-0">
63       <div class="container">
64         |   @yield('content')
65       </div>
66     </main>
```

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

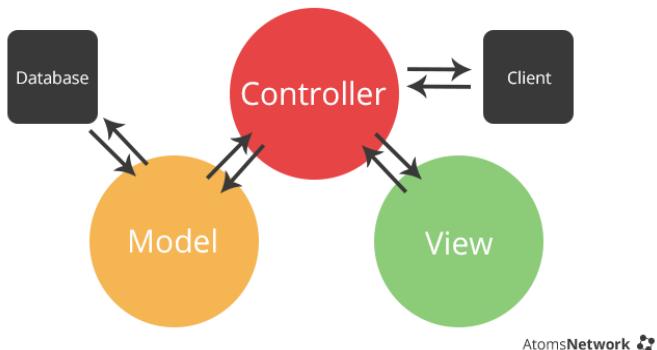
We zijn nu klaar met de masterpage. Zodra we de sub pagina's gaan maken, die we nodig hebben bij de controller, zal je precies zien hoe we dit gaan gebruiken.

Resource Controller

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.



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:\laragon\www\workshop
λ 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 this help message
  -q, --quiet     Do not output any message
  -V, --version   Display this application version
      --ansi       Force ANSI output
      --no-ansi    Disable ANSI output
  -n, --no-interaction  Do not ask any interactive question
      --env[=ENV]   The environment the command should run under
  -v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output,
                           2 for more verbose output and 3 for debug
```

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.

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

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

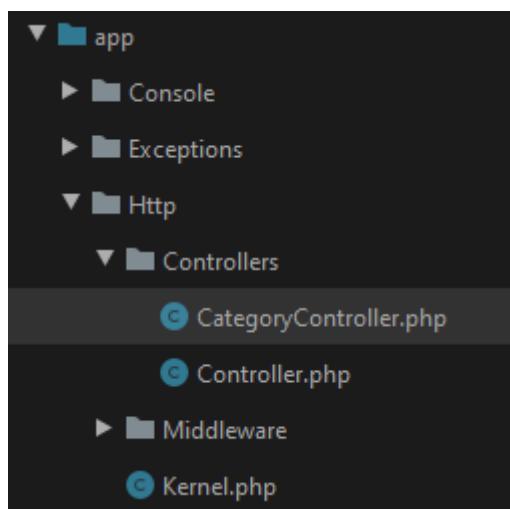
Format van de naam van de controller is ModelnaamController. Dit voegen we dan ook toe aan de regels:

- 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:\laragon\www\workshop
λ php artisan make:controller --model=Category CategoryController
Controller created successfully.
```

Binnen app/Http/Controllers 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 |

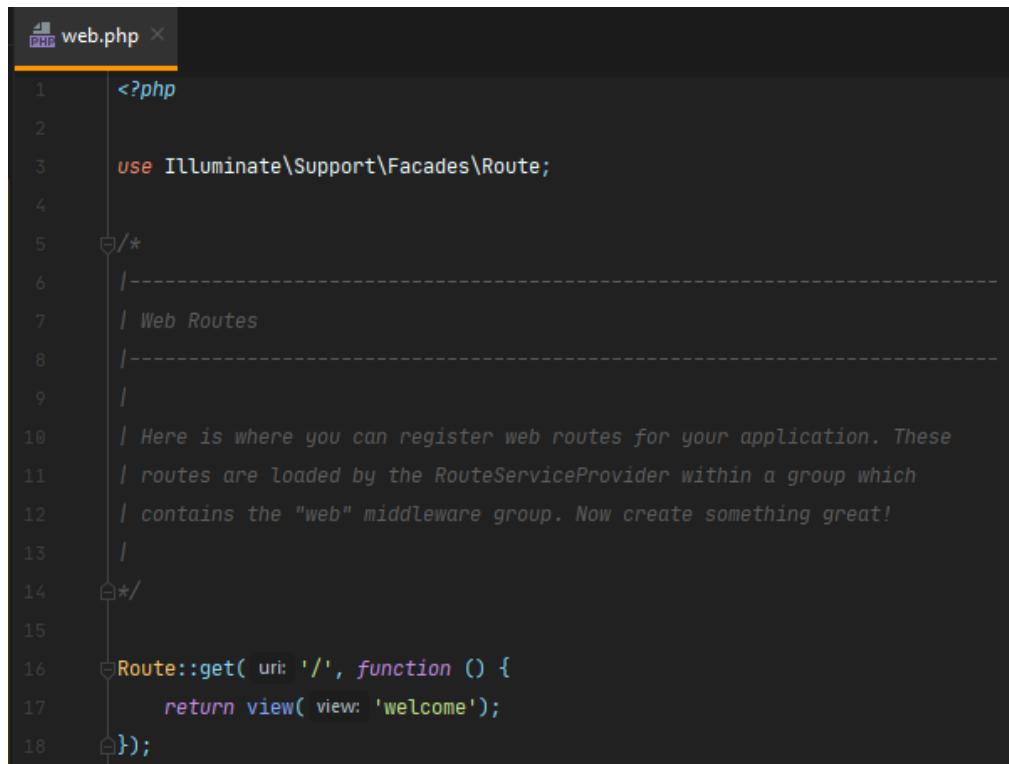
Met de resource controller hebben we dus alles wat we nodig hebben voor de crud.

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

```
5  use App\Category;
6  use Illuminate\Http\Request;
7
8  class CategoryController extends Controller
9  {
10     /**
11      * Display a listing of the resource. ...
12      */
13     public function index(){...}
14
15     /**
16      * Show the form for creating a new resource. ...
17      */
18     public function create(){...}
19
20     /**
21      * Store a newly created resource in storage. ...
22      */
23     public function store(Request $request){...}
24
25     /**
26      * Display the specified resource.
27      */
28     public function show(Category $category){...}
29
30     /**
31      * Show the form for editing the specified resource. ...
32      */
33     public function edit(Category $category){...}
34
35     /**
36      * Update the specified resource in storage. ...
37      */
38     public function update(Request $request, Category $category){...}
39
40     /**
41      * Remove the specified resource from storage. ...
42      */
43     public function destroy(Category $category){...}
44 }
45 }
```

Routing

Om ervoor te zorgen dat de resource controller straks gaat 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.



```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 /*
6 -----
7 | Web Routes
8 -----
9 |
10 | Here is where you can register web routes for your application. These
11 | routes are loaded by the RouteServiceProvider within a group which
12 | contains the "web" middleware group. Now create something great!
13 |
14 */
15
16 Route::get('/', function () {
17     return view('welcome');
18 });


```

Wat hier 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/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.

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 doen we op de manier die op regel 20 staat. Als we dus in de browser naar onze website gaan, zullen we iets van de resource controller zien.



```
16 Route::get('/', function () {
17     return view('welcome');
18 });
19
20 Route::resource('categories', 'CategoryController');
```

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 /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.

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 |

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

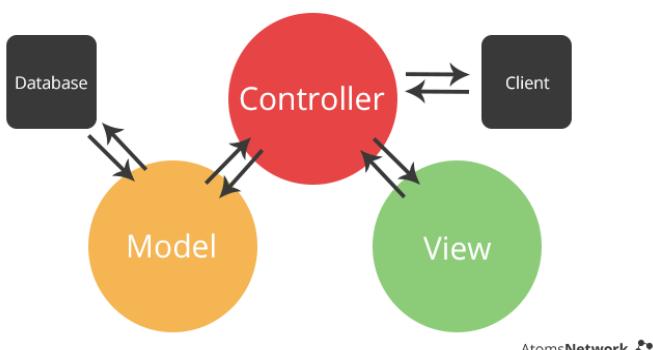
The Eloquent ORM included with Laravel provides a beautiful, simple ActiveRecord implementation for working with your database. Each database table has a corresponding "Model" which is used to interact with that table.

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.

```
3  namespace App\Http\Controllers;  
4  
5  use App\Category;  
6  use App\Http\Requests\CategoryStoreRequest;  
7  use Illuminate\Http\Request;  
8  
9  class CategoryController extends Controller  
10 {  
11     /** Display a listing of the resource. .... */  
12     public function index()  
13     {  
14         // ophalen van categories  
15         $categories = Category::all();  
16     }  
17 }
```

Let even op dat we Category op regel 18 kunnen gebruiken, omdat we op regel 5 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()
{
    // ophalen van categories
    $categories = Category::all();

    // een view returnen en de variabel $categories meestellen
    return view( view: 'admin.categories.index', compact('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`

Bij het aangeven van welke view we willen zien gebruiken we 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.

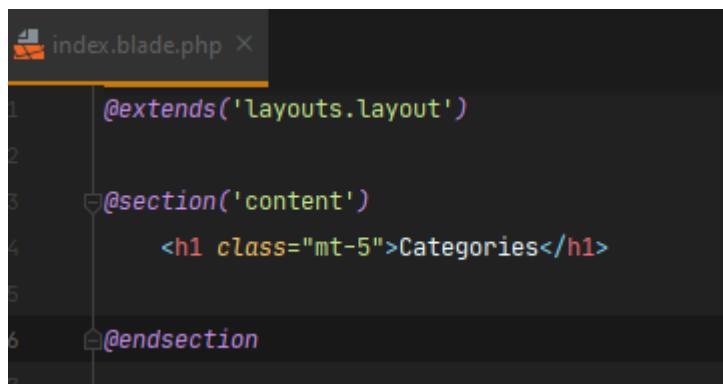
Met compact kan je variabelen meegeven richting de view. Je ziet geen \$ er staan, maar \$categories wordt op deze manier meegegeven. Als je wil kan je meerdere variabelen meegeven. Op dit moment nog niet nodig, maar dit gaan we later een keer gebruiken.

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.

Index in View

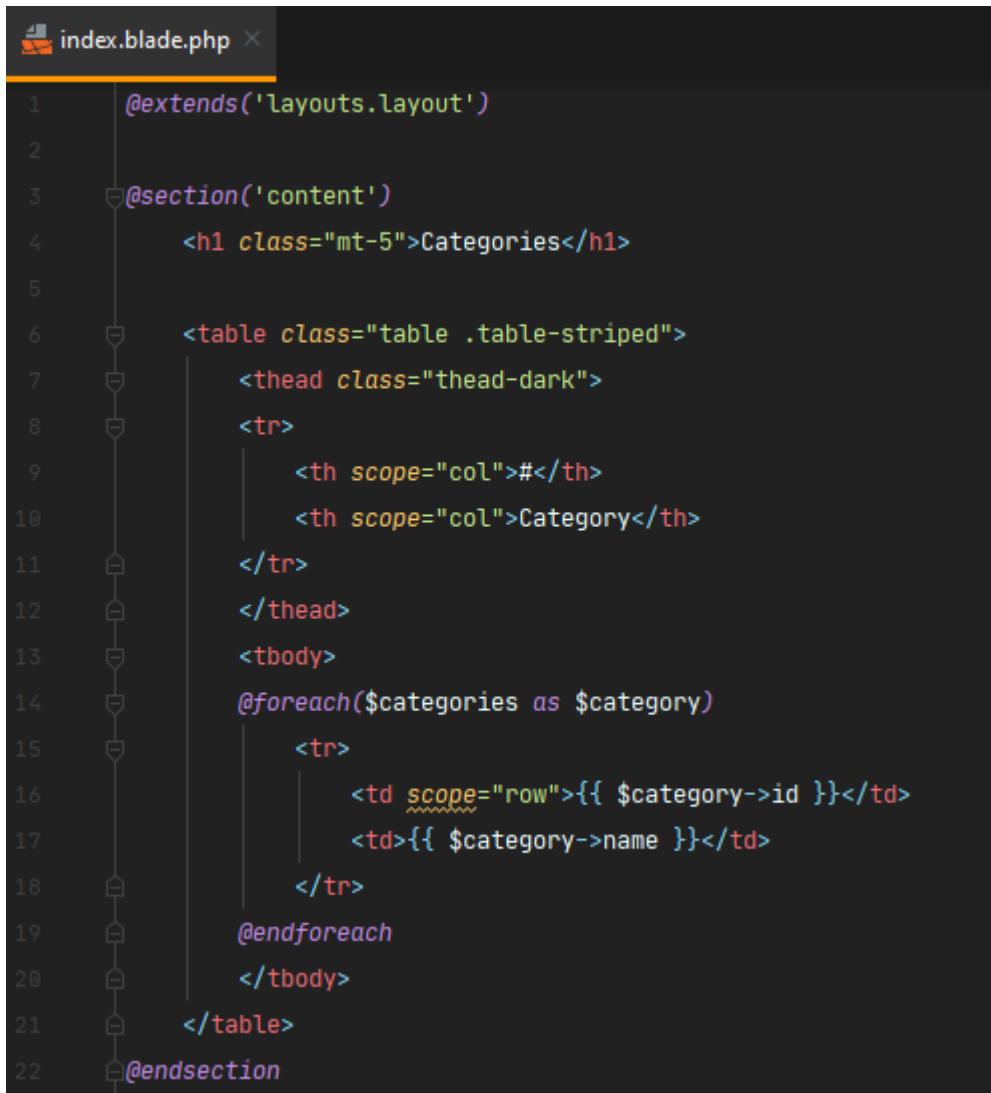
In de index.blade.php gaan we ervoor zorgen dat we een klein stukje html hebben, wat straks op de plek zal komen in de masterpage bij yield('content'). Zorgen dat de masterpage gebruikt wordt doen we met @extends



```
index.blade.php ×
1  @extends('layouts.layout')
2
3  @section('content')
4      <h1 class="mt-5">Categories</h1>
5
6  @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:



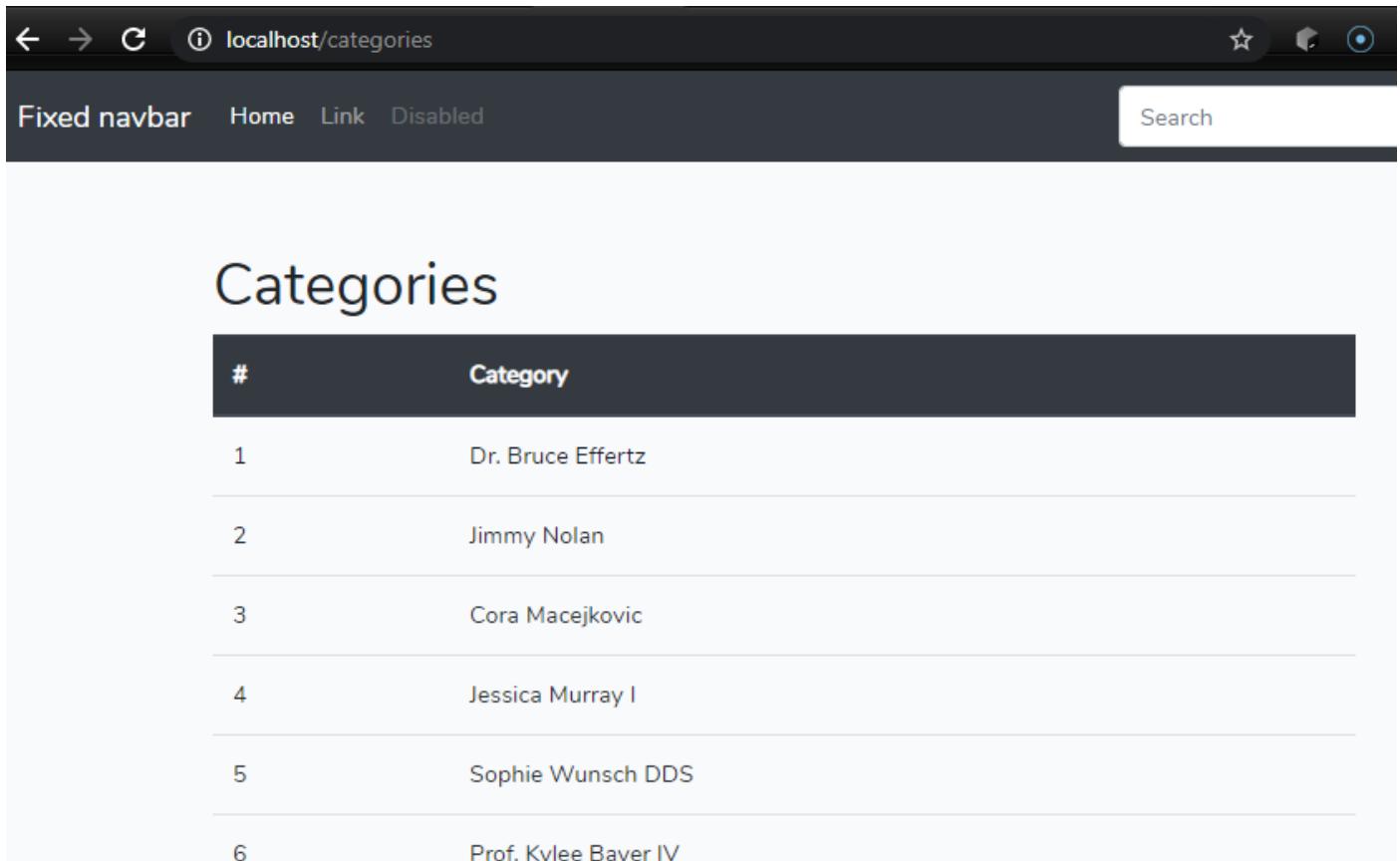
```
index.blade.php
1  @extends('layouts.layout')
2
3  @section('content')
4      <h1 class="mt-5">Categories</h1>
5
6      <table class="table .table-striped">
7          <thead class="thead-dark">
8              <tr>
9                  <th scope="col">#</th>
10                 <th scope="col">Category</th>
11             </tr>
12         </thead>
13         <tbody>
14             @foreach($categories as $category)
15                 <tr>
16                     <td scope="row">{{ $category->id }}</td>
17                     <td>{{ $category->name }}</td>
18                 </tr>
19             @endforeach
20         </tbody>
21     </table>
22     @endsection
```

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/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 `localhost/categories` in the address bar. The page has a dark header with a "Fixed navbar" link, a "Home" link, a "Link" link, and a "Disabled" link. There is also a search bar. The main content area displays a table with the following data:

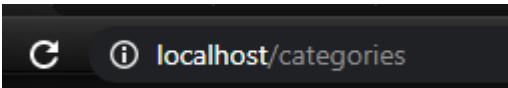
| # | Category |
|---|----------------------|
| 1 | Dr. Bruce Effertz |
| 2 | Jimmy Nolan |
| 3 | Cora Macejkovic |
| 4 | Jessica Murray I |
| 5 | Sophie Wunsch DDS |
| 6 | Prof. Kylee Bayer IV |

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.

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.

1. Je gaat naar de url localhost/categories



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

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

3. 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

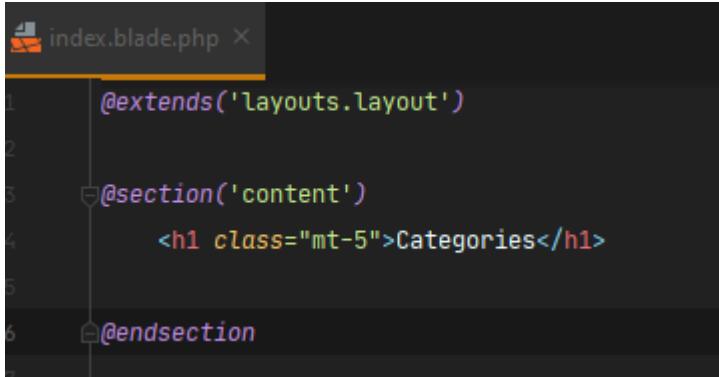
| Verb | URI | Action | Route Name |
|-----------|----------------------|---------|----------------|
| GET | /photos | index | photos.index |
| GET | /photos/create | create | photos.create |
| POST | /photos | store | photos.store |
| GET | /photos/{photo} | show | photos.show |
| GET | /photos/{photo}/edit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

4. 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.

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

    // een view returnen en de variabel $categories meestellen
    return view('admin.categories.index', compact('categories'));
}
```

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



```

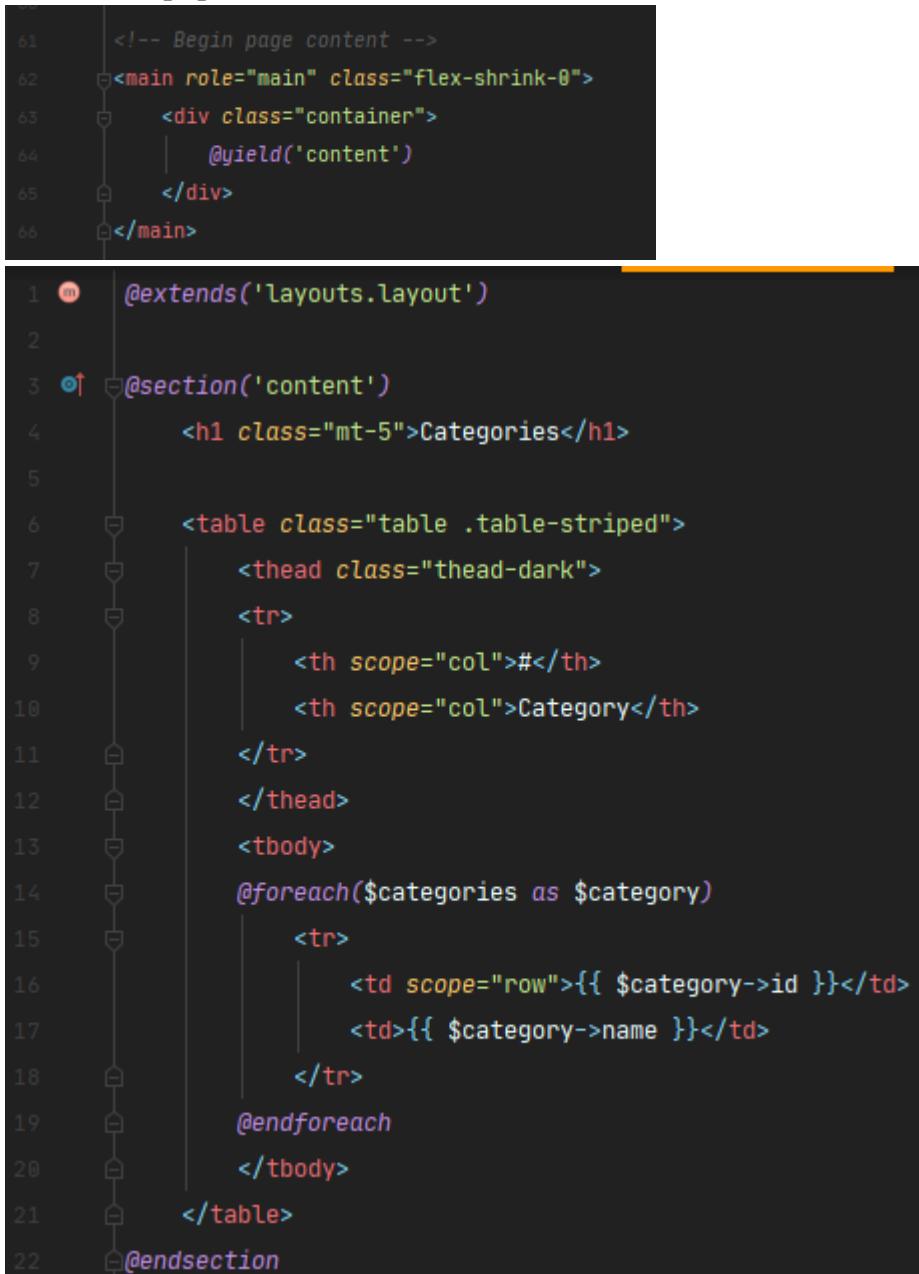
1 @extends('layouts.layout')

2

3 @section('content')
4     <h1 class="mt-5">Categories</h1>
5
6 @endsection

```

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



```

61     <!-- Begin page content -->
62     <main role="main" class="flex-shrink-0">
63         <div class="container">
64             @yield('content')
65         </div>
66     </main>

1 @extends('layouts.layout')

2

3 @section('content')
4     <h1 class="mt-5">Categories</h1>
5
6     <table class="table .table-striped">
7         <thead class="thead-dark">
8             <tr>
9                 <th scope="col">#</th>
10                <th scope="col">Category</th>
11            </tr>
12        </thead>
13        <tbody>
14            @foreach($categories as $category)
15                <tr>
16                    <td scope="row">{{ $category->id }}</td>
17                    <td>{{ $category->name }}</td>
18                </tr>
19            @endforeach
20        </tbody>
21    </table>
22 @endsection

```

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

The screenshot shows a web browser window with the URL 'localhost/categories' in the address bar. The page has a dark grey header with a 'Fixed navbar' logo, 'Home', 'Link', and 'Disabled' buttons, and a search input field. The main content area has a light grey background and features a large title 'Categories' at the top. Below it is a table with a dark grey header row containing the columns '# Category'. Six rows of data follow, each with a number from 1 to 6 and a corresponding name.

| # | Category |
|---|----------------------|
| 1 | Dr. Bruce Effertz |
| 2 | Jimmy Nolan |
| 3 | Cora Macejkovic |
| 4 | Jessica Murray I |
| 5 | Sophie Wunsch DDS |
| 6 | Prof. Kylee Bayer IV |

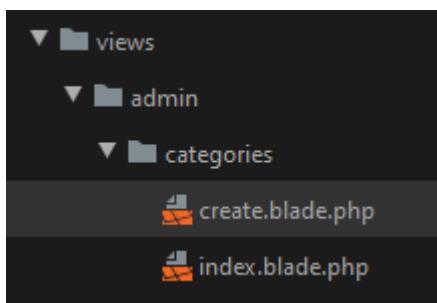
Create

Nu de index 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 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.

A screenshot of a code editor window titled 'create.blade.php'. The code is as follows:

```
1  @extends('layouts.layout')
2
3  @section('content')
4      <h1 class="mt-5">Categories</h1>
5
6
7
8  @endsection
```

The code uses Blade syntax to extend a layout and define a section with an H1 heading.

Nu gaan we met html een formulier maken. Omdat we bootstrap gebruiken kunnen we gebruik maken van de voorbeeld forms op hun website. (<https://getbootstrap.com/docs/4.5/components/forms/>)



```
1 @extends('layouts.layout')
2
3 @section('content')
4     <h1 class="mt-5">Categories</h1>
5
6     <form method="POST" action="{{ route('categories.store') }}>
7         @csrf
8
9         <div class="form-group">
10             <label for="name">Category name</label>
11             <input type="text" name="name" class="form-control" id="name"
12                 aria-describedby="categorienameHelp" placeholder="Enter Category name">
13         </div>
14
15         <button type="Submit" class="btn btn-primary">Submit</button>
16
17     </form>
18
19     @endsection
```

Als we een POST stuurt naar /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 method="POST" action="{{ route('categories.store') }}>
| @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 method="POST" action="http://localhost/categories">
    <input type="hidden" name="_token" value="cmqQU3F0rX4BMQlJzeA7xjA5WcZXZaKwlzOkjHjf">
    <div class="form-group">
        <label for="name">Category name</label>
        <input type="text" name="name" class="form-control" id="name"
               aria-describedby="categorienameHelp" placeholder="Enter Category name">
    </div>
    <button type="Submit" class="btn btn-primary">Submit</button>
</form>
```

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

The screenshot shows a web browser window with the URL `localhost/categories/create` in the address bar. The page has a dark header with a back arrow, forward arrow, and refresh icon. Below the header is a fixed navbar with links for 'Fixed navbar', 'Home', 'Link', and 'Disabled'. The main content area has a large title 'Categories'. Below the title is a label 'Category name' followed by a text input field containing 'Enter Category name'. At the bottom is a blue 'Submit' button.

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. Hiervoor ga ik in de masterpage een link maken naar de Categories. Hiervoor gaan we dus de layout.blade.php bewerken.

In ons project zullen meerdere admins komen. Het is dan misschien makkelijker om in het menu een dropdown te maken waar we alle admin pagina's gaan zetten. Hiervoor hebben we dus een dropdown nodig.

Op de website van bootstrap vinden we die hier: <https://getbootstrap.com/docs/4.5/components/navbar/>

In de masterpage voeg ik dus de dropdown in bij het menu.

```
<div class="collapse navbar-collapse" id="navbarCollapse">
  <ul class="navbar-nav mr-auto">
    <li class="nav-item active">
      <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Link</a>
    </li>
    <li class="nav-item">
      <a class="nav-link disabled" href="#" tabindex="-1" aria-disabled="true">Disabled</a>
    </li>
    <li class="nav-item dropdown">
      <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="dropdown"
         aria-haspopup="true" aria-expanded="false">Admin
      </a>
      <div class="dropdown-menu" aria-labelledby="navbarDropdown">
        <a class="dropdown-item" href="{{ route('categories.index') }}>Category Admin</a>
        <a class="dropdown-item" href="#">Product Admin</a>
        <div class="dropdown-divider"></div>
        <a class="dropdown-item" href="#">Something else here</a>
      </div>
    </li>
  </ul>
```

Ook hier maar ik weer gebruik van de named routes binnen Laravel.

Verder vind ik het handig om dan vanuit de categories alles te kunnen bereiken. Hier maak ik dan een extra navigatie aan. Dit doe ik nu even in elk bestand van de categorie views. Hiervoor gebruik ik een stukje navigatie met tabs: <https://getbootstrap.com/docs/4.5/components/navs/#tabs>

In de *index.blade.php* en de *create.blade.php* komt dus dit stuk code erbij. (Let even op dat bij de linkjes een active class zit. Deze hoort wel goed te staan)

```
<h1 class="mt-5">Categories</h1>

<nav class="nav">
  <ul class="nav nav-tabs">
    <li class="nav-item">
      <a class="nav-link active" href="{{ route('categories.index') }}">Index</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="{{ route('categories.create') }}">Create</a>
    </li>
  </ul>
</nav>

<table class="table .table-striped">
```

Nu kunnen we vanuit de browser makkelijk van de index naar de create en weer terug.

Categories

| # | Category |
|---|-------------------|
| 1 | Dr. Bruce Effertz |
| 2 | Jimmy Nolan |

Categories

Index Create

Category name

Submit

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 staan 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:

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

In de \$request wordt alle Postdata bewaart.

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

`Illuminate\Database\Eloquent\MassAssignmentException`

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

<http://localhost/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/Category`

```
class Category extends Model
{
    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

Facade\Ignition\Exceptions\ViewException

Undefined variable: categories (View:

D:\laragon\www\workshop\resources\views\admin\categories\index.blade.php)

<http://localhost/categories>

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.

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.

Hiervoor gebruiken we `php artisan make:request CategoryStoreRequest`

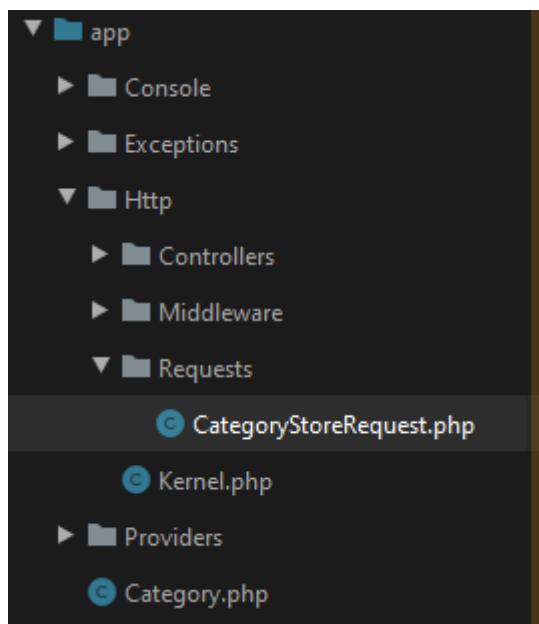
```
D:\laragon\www\workshop
└ php artisan make:request CategoryStoreRequest
Request created successfully.
```

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 nu een map voor Requests gekomen, 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      /** Determine if the user is authorized to make this request. ...*/  
14     public function authorize()  
15     {  
16         return false;  
17     }  
18  
19     /** Get the validation rules that apply to the request. ...*/  
24     public function rules()  
25     {  
26         return [  
27             //  
28         ];  
29     }  
30 }
```

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/7.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.

```
public function rules()
{
    return [
        'name' => 'required|string|unique:categories|max:100'
    ];
}
```

- 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 mag maximaal 100 characters bevatten. We gebruiken 100 characters omdat ons veld in de database die grootte heeft.

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;
4
5  use App\Category;
6  use App\Http\Requests\CategoryStoreRequest;
7  use Illuminate\Http\Request;
8
9  class CategoryController extends Controller
10 {
```

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.

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 <h1>. Ik gebruik hiervoor de standaard soort van Laravel. Je kan deze hier vinden: <https://laravel.com/docs/7.x/validation#quick-displaying-the-validation-errors>

```
3   @section('content')
4       <h1 class="mt-5">Categories</h1>
5
6       @if ($errors->any())
7           <div class="alert alert-danger">
8               <ul>
9                   @foreach ($errors->all() as $error)
10                      <li>{{ $error }}</li>
11                  @endforeach
12              </ul>
13          </div>
14      @endif
15
16      <nav class="nav">
```

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.

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/7.x/responses#redirecting-with-flashed-session-data>

Ook deze zet ik onder de <h1>.

```
3     @section('content')
4         <h1 class="mt-5">Categories</h1>
5
6         @if (session('message'))
7             <div class="alert alert-success">
8                 {{ session('message') }}
9             </div>
10        @endif
11
12    <nav class="nav">
```

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('message', 'Categorie aangemaakt');
}
```

Als je nu gaat testen merk je, dat je netjes de errors ziet:

The screenshot shows a web page titled "Categories". Below the title, there is a red callout box containing the error message: "The name has already been taken." At the bottom of the page, there is a form with a "Create" button highlighted in blue. The form fields include "Category name" and "Enter Category name". A "Submit" button is also present.

En 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.

Categories

Categorie aangemaakt

[Index](#)

[Create](#)

| # | Category |
|---|-------------------|
| 1 | Dr. Bruce Effertz |
| 2 | Jimmy Nolan |

De validatie voor het opslaan van categorie is nu klaar.

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:

`/categories/{category}`

Als we dan kijken naar de show methode zoals die nu is, gaan we dus straks een id mee krijgen vanuit de URL. Hiermee wordt automatisch door Laravel wat mee gedaan. 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('category'));
}
```

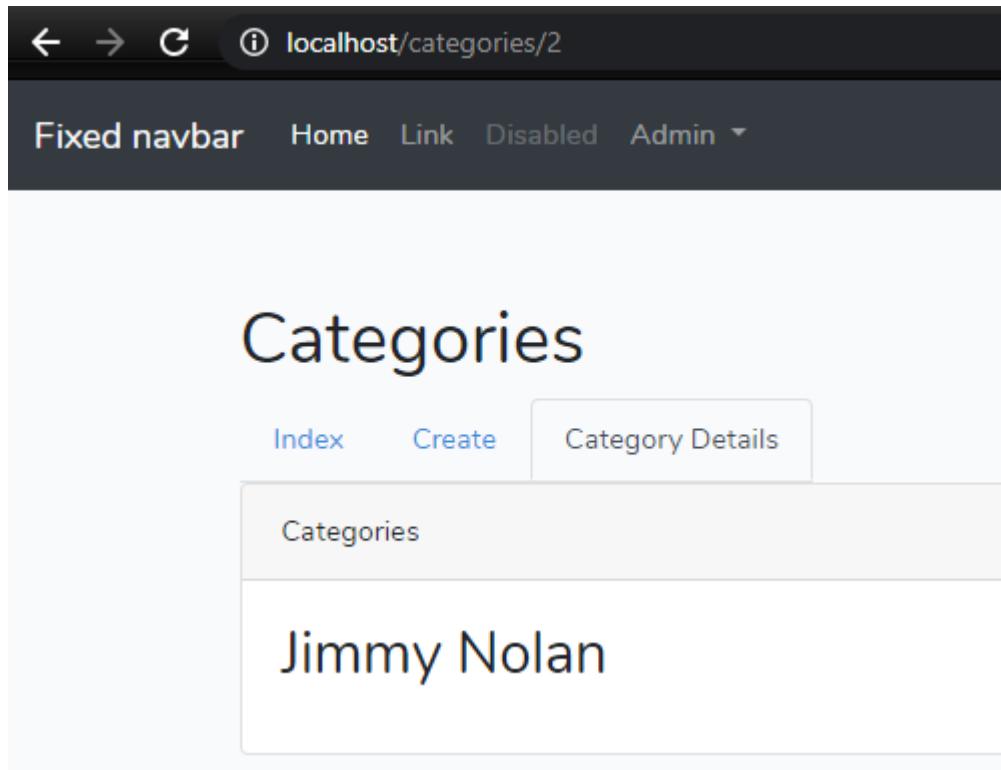
Voor de view maken we een show.blade.php aan in de map categories (mag ondertussen verwachten dat je weet waar dit is). Om ervoor te zorgen dat het laten zien van 1 categorie een beetje netjes is, gebruik ik een card van bootstrap (<https://getbootstrap.com/docs/4.5/components/card/>)

Verder zorg ik ervoor dat de masterpage wordt ingeladen en dat de section content heet. Het is nog steeds handig om een navigatie binnen de categories te hebben, zoals we al in de index en create hebben. Omdat we in de url straks een category id willen krijgen, bijvoorbeeld /categories/3 om naar de show met category 3 te gaan, zullen we de id moeten meegeven met de named route in onze navigatie.

Je ziet op regel 16+17 hoe je dit kan doen.

```
1  @extends('layouts.layout')
2
3  @section('content')
4      <h1 class="mt-5">Categories</h1>
5
6      <nav class="nav">
7          <ul class="nav nav-tabs">
8              <li class="nav-item">
9                  <a class="nav-link" href="{{ route('categories.index') }}">Index</a>
10             </li>
11             <li class="nav-item">
12                 <a class="nav-link" href="{{ route('categories.create') }}">Create</a>
13             </li>
14             <li class="nav-item">
15                 <a class="nav-link active" href="{{ route('categories.show', [
16                     'category' => $category->id]) }}>Category Details</a>
17             </li>
18         </ul>
19     </nav>
20
21
22     <div class="card">
23         <div class="card-header">
24             Categories
25         </div>
26         <div class="card-body">
27             <h2 class="card-title">{{ $category->name }}</h2>
28         </div>
29     </div>
30
31     @endsection
```

Het resultaat ziet er ook strak genoeg uit. Natuurlijk hebben we bij een categorie alleen maar een naam. Als we dit bij andere Cruds gaan doen zal je zien dat het goed te doen is.



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.

```
<table class="table .table-striped">
    <thead class="thead-dark">
        <tr>
            <th scope="col">#</th>
            <th scope="col">Category</th>
            <th scope="col">Category details</th>
        </tr>
    </thead>
    <tbody>
        @foreach($categories as $category)
            <tr>
                <td scope="row">{{ $category->id }}</td>
                <td>{{ $category->name }}</td>
                <td><a href="{{ route('categories.show', ['category' => $category->id]) }}">Details</a></td>
            </tr>
        @endforeach
    </tbody>
</table>
```

Het resultaat van de aangepaste index.blade ziet er goed uit. Later gaan we deze index verder uitbreiden.

Categories

Index

Create

| # | Category | Category details |
|---|----------------------|-------------------------|
| 1 | Dr. Bruce Effertz | Details |
| 2 | Jimmy Nolan | Details |
| 3 | Cora Macejkovic | Details |
| 4 | Jessica Murray I | Details |
| 5 | Sophie Wunsch DDS | Details |
| 6 | Prof. Kylee Bayer IV | Details |
| 7 | Wilton Wehner | Details |

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( 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 de create niet de actieve tab is. De navigatie wordt dan zo:

```
<nav class="nav">
    <ul class="nav nav-tabs">
        <li class="nav-item">
            <a class="nav-link" href="{{ route('categories.index') }}>Index</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="{{ route('categories.create') }}>Create</a>
        </li>
        <li class="nav-item">
            <a class="nav-link active" href="{{ route('categories.edit',
                ['category' => $category->id]) }}>Edit Category</a>
        </li>
    </ul>
</nav>
```

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 method="POST" action="{{ route('categories.update', ['category' => $category->id]) }}>
    @method('PUT')
    @csrf

    <div class="form-group">
        <label for="name">Category name</label>
        <input type="text" name="name" class="form-control" id="name" aria-describedby="categorienameHelp"
               value="{{ $category->name }}>
    </div>
    <button type="submit" class="btn btn-primary">Update</button>
</form>
```

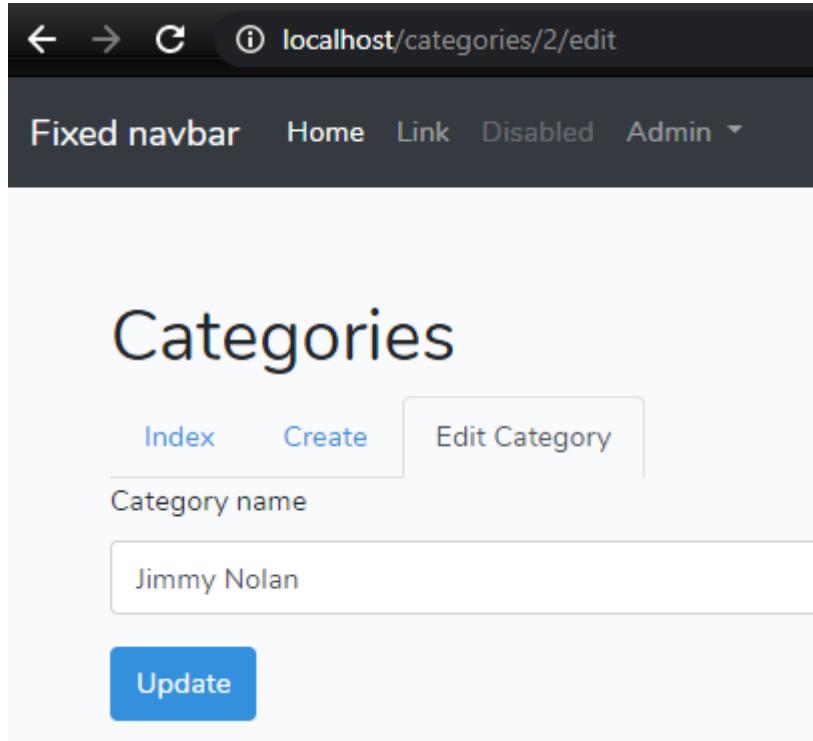
Als je goed kijkt sturen we het formulier nu met de method PUT. Deze methode heb je nog nooit gezien, maar wordt door Laravel gebruikt om de verschillen duidelijk te maken. Als je naar de bron van het formulier gaat kijken in je browser zal je dit zien.

```
92     <form method="POST" action="http://localhost/categories/2">
93         <input type="hidden" name="_method" value="PUT">           <input type="hidden" name="_token"
value="Wgt61j2S8abf16YebG9ULutIgu81vsAzNPqJDYtu">
94         <div class="form-group">
95             <label for="name">Category name</label>
96             <input type="text" name="name" class="form-control" id="name" aria-
describedby="categorienameHelp"
97                   value="Jimmy Nolan">
98         </div>
99         <button type="submit" class="btn btn-primary">Update</button>
100    </form>
```

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 lege quotes bij de inputs hebben we ook niet meer. Daar staat nu de value, wat dus met het object \$category gevuld wordt.

We hebben nu een netjes ingevuld formulier gekregen



localhost/categories/2/edit

Fixed navbar Home Link Disabled Admin ▾

Categories

Index Create Edit Category

Category name

Jimmy Nolan

Update

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. De *index.blade* wordt dan zo:

```
<table class="table .table-striped">
  <thead class="thead-dark">
    <tr>
      <th scope="col">#</th>
      <th scope="col">Category</th>
      <th scope="col">Category details</th>
      <th scope="col">Edit</th>
    </tr>
  </thead>
  <tbody>
    @foreach($categories as $category)
      <tr>
        <td scope="row">{{ $category->id }}</td>
        <td>{{ $category->name }}</td>
        <td><a href="{{ route('categories.show', ['category' => $category->id]) }}>Details</a></td>
        <td><a href="{{ route('categories.edit', ['category' => $category->id]) }}>Edit</a></td>
      </tr>
    @endforeach
  </tbody>
</table>
```

Ons overzicht ziet er dan zo uit:

Categories

| Index | Create | Category | Category details | Edit |
|-------|--------|-------------------|-------------------------|----------------------|
| 1 | | Dr. Bruce Effertz | Details | Edit |
| 2 | | Jimmy Nolan | Details | Edit |
| 3 | | Cora Macejkovic | Details | Edit |
| 4 | | Jessica Murray I | Details | Edit |
| 5 | | Sophie Wunsch DDS | Details | Edit |

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.

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)
{
    //
}
```

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. Als eerst gaan we dan ook de update uitvoeren.

```
/** Update the specified resource in storage. ....*/
public function update(Request $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:

```
public function rules()
{
    return [
        'name' => 'required|string|unique:categories|max:100'
    ];
}
```

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:\laragon\www\workshop
λ 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.

```
/** 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|max:100|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 en dan we een redirect krijgen richting de index.

```
/** 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('message', 'Categorie geupdate');
}
```

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

Categories

- The name has already been taken.

[Index](#) [Create](#)

[Edit Category](#)

Category name

Dr. Bruce Effertz

Update

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

Als we dan een naam gebruiken die wel kan:

Categories

Categorie geupdate

[Index](#) [Create](#)

| # | Category | Category details | Edit |
|---|-------------------|-------------------------|----------------------|
| 1 | Dr. Bruce Effertz | Details | Edit |
| 2 | Jimmy Nolan | Details | Edit |
| 3 | Cora Macejkovic | Details | Edit |

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

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.

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)

```
<table class="table .table-striped">
    <thead class="thead-dark">
        <tr>
            <th scope="col">#</th>
            <th scope="col">Category</th>
            <th scope="col">Category details</th>
            <th scope="col">Edit</th>
            <th scope="col">Delete</th>
        </tr>
    </thead>
    <tbody>
        @foreach($categories as $category)
            <tr>
                <td scope="row">{{ $category->id }}</td>
                <td>{{ $category->name }}</td>
                <td><a href="{{ route('categories.show', ['category' => $category->id]) }}>Details</a></td>
                <td><a href="{{ route('categories.edit', ['category' => $category->id]) }}>Edit</a></td>
                <td><a href="{{ route('categories.delete', ['category' => $category->id]) }}>Delete</a></td>
            </tr>
        @endforeach
    </tbody>
</table>
```

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

De routes vinden we in de map routes/web.php

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

```
20     Route::get( uri: 'categories/{category}/delete',   action: 'CategoryController@delete')
21         ->name( name: 'categories.delete');
22     Route::resource( name: '/categories',   controller: 'CategoryController');
```

Nu bestaat de route om ervoor te zorgen dat we voor de CategoryController we naar de delete method kunnen. We geven deze ook meteen een naam omdat we al de hele tijd met named routes werken. Bij Resource gaat dat automatisch, bij de get niet.

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.

```
88     /** Show the form for deleting the specified resource. ...*/
94     public function delete(Category $category)
95     {
96
97     }
98
99     /** Remove the specified resource from storage. ...*/
105    public function destroy(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.

```
88     /** Show the form for deleting the specified resource. ...*/
94     public function delete(Category $category)
95     {
96         return view( view: 'admin.categories.delete', compact('category'));
97     }
```

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.

In de navigatie hiervan kunnen we meteen wijzigen dat we op de delete zijn

```
<nav class="nav">
  <ul class="nav nav-tabs">
    <li class="nav-item">
      <a class="nav-link" href="{{ route('categories.index') }}">Index</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="{{ route('categories.create') }}">Create</a>
    </li>
    <li class="nav-item">
      <a class="nav-link active" href="{{ route('categories.delete', [
        'category' => $category->id]) }}">Delete Category</a>
    </li>
  </ul>
</nav>
```

Bij het formulier wijzigen we de method naar DELETE. Verder zorgen we ervoor dat de inputs disabled zijn, zodat het niet lijkt alsof we een categorie aan het wijzigen zijn. Verder veranderen we de tekst van de knop in Delete.

```
<form method="POST" action="{{ route('categories.destroy', ['category' => $category->id]) }}>
  @method('DELETE')
  @csrf

  <div class="form-group">
    <label for="name">Category name</label>
    <input type="text" name="name" class="form-control" id="name" aria-describedby="categorienameHelp"
           value="{{ $category->name }}" disabled="disabled">
  </div>
  <button type="submit" class="btn btn-primary">Delete</button>
</form>
```

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.

De links naar Delete staan in het overzicht.

| Categories | | | | |
|------------|-------------------|-------------------------|----------------------|------------------------|
| # | Category | Category details | Edit | Delete |
| 1 | Dr. Bruce Effertz | Details | Edit | Delete |
| 2 | Jimmy Nolan | Details | Edit | Delete |
| 3 | Cora Macejkovic | Details | Edit | Delete |

En we zien netjes het formulier bij delete. De input is ook disabled. Tot zover werkt dus alles.

Categories

Index Create Delete Category

Category name

Dr. Bruce Effertz

Delete

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', 'Category deleted');
}
```

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.

Categories

Category deleted

[Index](#) [Create](#)

| # | Category | Category details | Edit | Delete |
|---|------------------|-------------------------|----------------------|------------------------|
| 2 | Jimmy Nolan | Details | Edit | Delete |
| 3 | Cora Macejkovic | Details | Edit | Delete |
| 4 | Jessica Murray I | Details | Edit | Delete |

Nu is de gehele CRUD voor categories klaar. We hebben een overzicht, kunnen toevoegen, wijzigen en verwijderen.

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.

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://docs.spatie.be/laravel-permission/v3/introduction/>

Associate users with permissions and roles

Sponsor

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/overview>.

packagist v3.11.0 Run Tests passing StyleCI passed downloads 5.6M

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

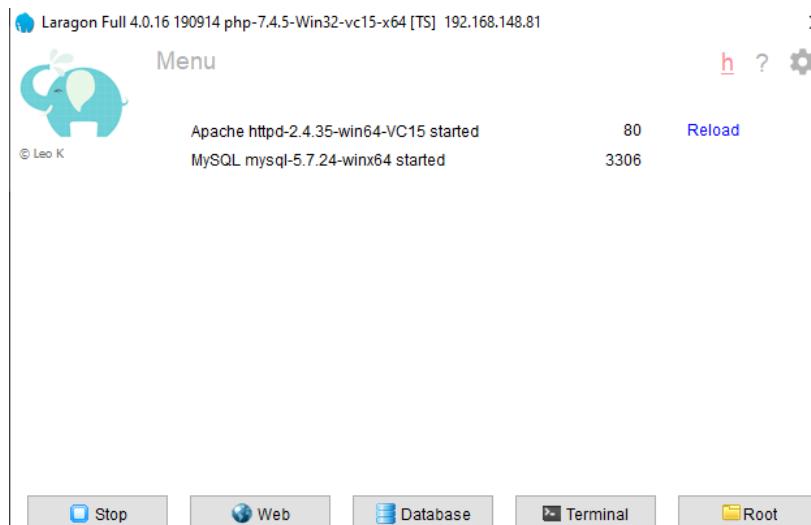
Once installed you can do stuff like this:

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

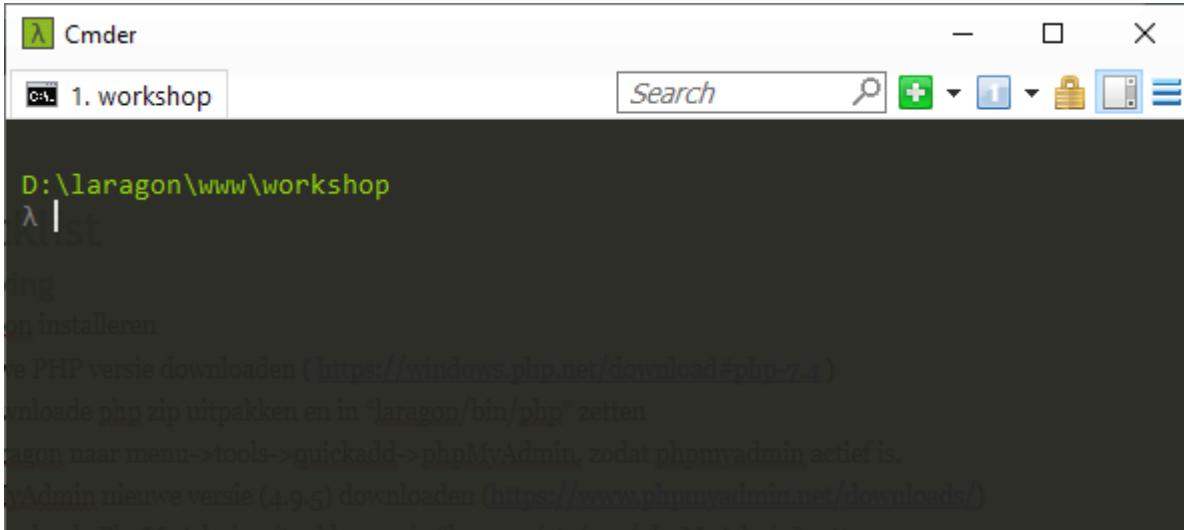
// Adding permissions via a role
$user->assignRole('writer');

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

Om deze package binnen te halen gaan we de command line gebruiken. Let wel op, dat Laragon zelf een terminal heeft, die alle settings van het programma gebruikt. Het kan zijn dat je editor waarin misschien een ingebouwde terminal heeft andere settings gebruikt. Voor nu is het dus veiliger om die van Laragon te gebruiken.



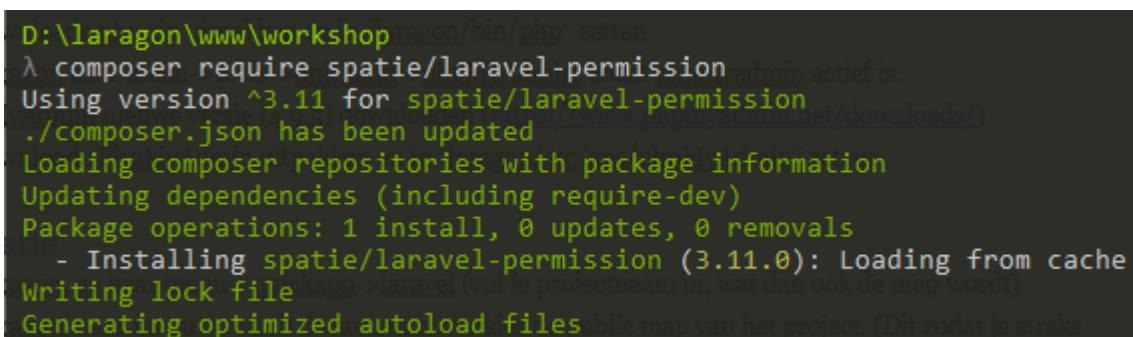
Als je de terminal opent zie je dit:



```
D:\laragon\www\workshop
λ | ng
n installeren
e PHP versie downloaden ( https://windows.php.net/download#php-7.4 )
nlaade php zip uitpakken en in "laragon/bin/php" zetten
ragon naar menu->tools->quickadd->phpMyAdmin, zodat phpmyadmin actief is.
Admin nieuwe versie (4.9.5) downloaden (https://www.phpmyadmin.net/downloads/)
```

Let even op dat je in de juiste map zit, voor mij dus de workshop map. We kunnen dan met composer de package binnen halen met:

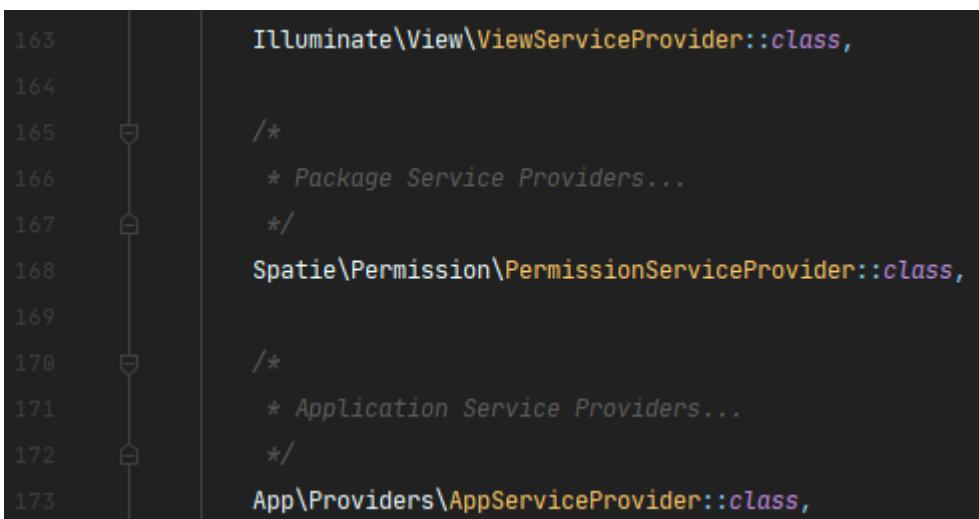
Composer require spatie/laravel-permission



```
D:\laragon\www\workshop
λ composer require spatie/laravel-permission admin actief is.
Using version ^3.11 for spatie/laravel-permission
./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 spatie/laravel-permission (3.11.0): Loading from cache
Writing lock file
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*,



```
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,
```

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:

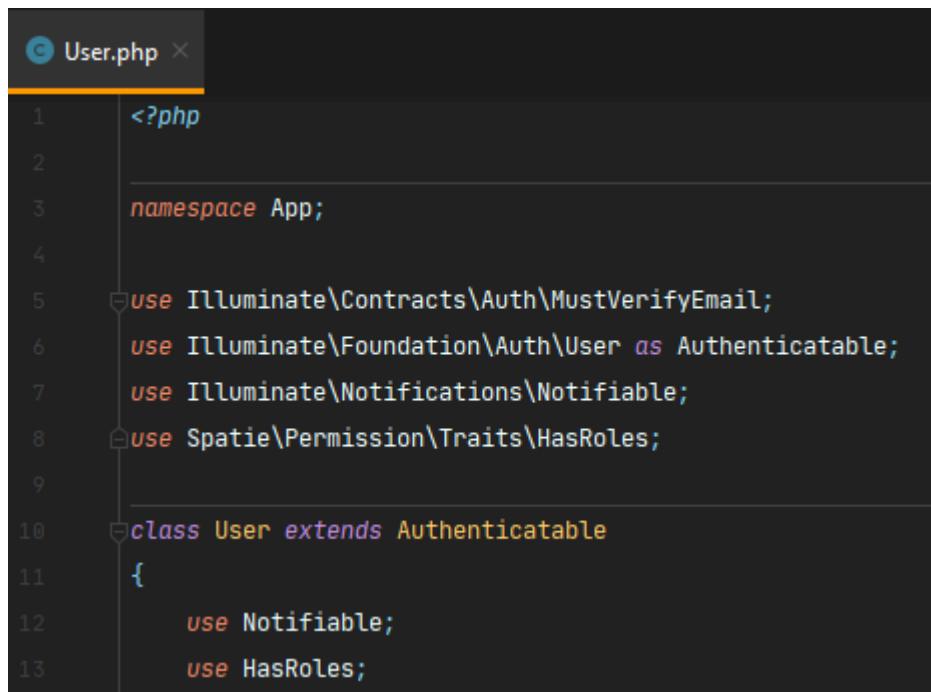
```
php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider"
```

```
D:\laragon\www\workshop
λ 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\2020_05_10_143304_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;*



```
User.php
1 <?php
2
3 namespace App;
4
5 use Illuminate\Contracts\Auth\MustVerifyEmail;
6 use Illuminate\Foundation\Auth\User as Authenticatable;
7 use Illuminate\Notifications\Notifiable;
8 use Spatie\Permission\Traits\HasRoles;
9
10 class User extends Authenticatable
11 {
12     use Notifiable;
13     use HasRoles;
```

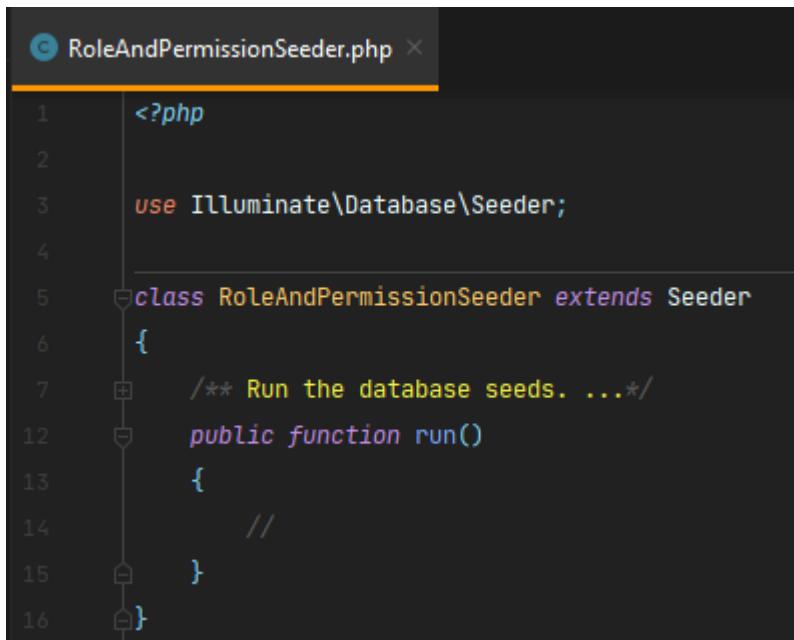
Als het goed is komt regel 8 er vanzelf bij, anders handmatig toevoegen.

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*

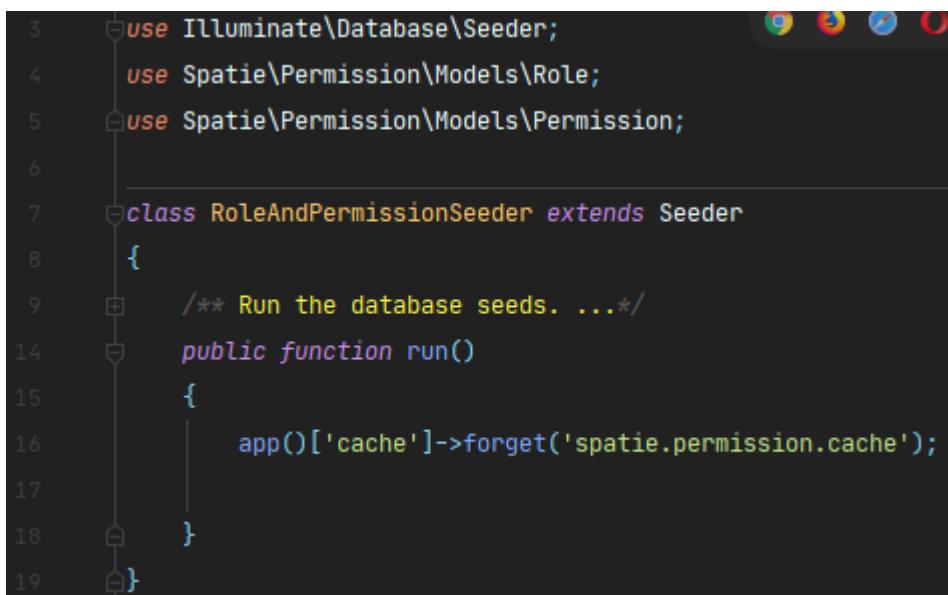
```
D:\laragon\www\workshop
└─ php artisan make:seeder RoleAndPermissionSeeder
    Seeder created successfully.
```

Hierdoor staat de seeder weer klaar.



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

Met de seeder gaan we de database dusdanig vullen dat we een aantal permissies 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. 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.



```
① RoleAndPermissionSeeder.php ×
1
2
3 use Illuminate\Database\Seeder;
4 use Spatie\Permission\Models\Role;
5 use Spatie\Permission\Models\Permission;
6
7 class RoleAndPermissionSeeder extends Seeder
8 {
9     /**
10      * Run the database seeds. ...
11     */
12    public function run()
13    {
14        app()['cache']->forget('spatie.permission.cache');
15    }
16}
17
18
19}
```

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.

```
public function run()
{
    app()['cache']->forget('spatie.permission.cache');

    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.

```
public function run()
{
    app()['cache']->forget('spatie.permission.cache');

    Permission::create(['name' => 'create category']);
    Permission::create(['name' => 'edit category']);
    Permission::create(['name' => 'delete category']);

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

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

    $role = Role::create(['name' => 'admin']);
    $role->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.

De owner heeft alle permissies gekregen.

Sales mag wel een categorie aanmaken en wijzigen, maar niet verwijderen.

De admin mag alles.

User Seeder

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

```
D:\laragon\www\workshop
λ 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\User;* bovenin gebruiken.

```
3   use Illuminate\Database\Seeder;
4
5   use App\User;
6
7   class UserSeeder extends Seeder
8   {
9
10      /** Run the database seeds. ...*/
11
12      public function run()
13      {
14
15          factory( User::class, 1 )->create([
16              'name'=>'Customer',
17              'email'=>'customer@test.com',
18              'password'=> bcrypt( value: 'test1234') ])
19              ->each(function (User $user) {
20                  $user->assignRole( ...roles: 'customer');
21              });
22
23          factory( User::class, 1 )->create([
24              'name'=>'Sales',
25              'email'=>'sales@test.com',
26              'password'=> bcrypt( value: 'test1234') ])
27              ->each(function (User $user) {
28                  $user->assignRole( ...roles: 'sales');
29              });
30
31          factory( User::class, 1 )->create([
32              'name'=>'Admin',
33              'email'=>'admin@test.com',
34              'password'=> bcrypt( value: 'test1234') ])
35              ->each(function (User $user) {
36                  $user->assignRole( ...roles: 'admin');
37              });
38
39          factory( User::class, 50 )->create();
40      }
41  }
```

Wat we hier steeds doen is voor de aangemaakte user de assignRole methode gebruiken om de user een rol te geven. Let wel even op dat je regel 4 niet vergeet, die zorgt ervoor dat we deze methode kunnen gebruiken in onze seeder.

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( class: RolesAndPermissionsSeeder::class );
        $this->call( class: UserSeeder::class );
        $this->call( 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.

```
D:\laragon\www\workshop
λ 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 (0.02 seconds)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (0.02 seconds)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (0.01 seconds)
Migrating: 2020_05_10_143304_create_permission_tables
Migrated: 2020_05_10_143304_create_permission_tables (0.25 seconds)
Migrating: 2020_05_10_191303_create_categories_table
Migrated: 2020_05_10_191303_create_categories_table (0.01 seconds)
Seeding: RoleAndPermissionSeeder
Seeded: RoleAndPermissionSeeder (0.05 seconds)
Seeding: UserSeeder
Seeded: UserSeeder (0.3 seconds)
Seeding: CategorySeeder
Seeded: CategorySeeder (0.02 seconds)
Database seeding completed successfully.
```

In de database hebben we nu wel veel meer tabellen gekregen

| Tabel | |
|--------------------------|-----------------------|
| <input type="checkbox"/> | categories |
| <input type="checkbox"/> | failed_jobs |
| <input type="checkbox"/> | migrations |
| <input type="checkbox"/> | model_has_permissions |
| <input type="checkbox"/> | model_has_roles |
| <input type="checkbox"/> | password_resets |
| <input type="checkbox"/> | permissions |
| <input type="checkbox"/> | roles |
| <input type="checkbox"/> | role_has_permissions |
| <input type="checkbox"/> | users |
| 10 tabellen | |

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.

| <u>id</u> | <u>name</u> | <u>guard_name</u> | |
|-----------|-------------|-------------------|--|
| 1 | customer | web | |
| 2 | sales | web | |
| 3 | admin | web | |

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,
    'bindings' => \Illuminate\Routing\Middleware\SubstituteBindings::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,
];
```

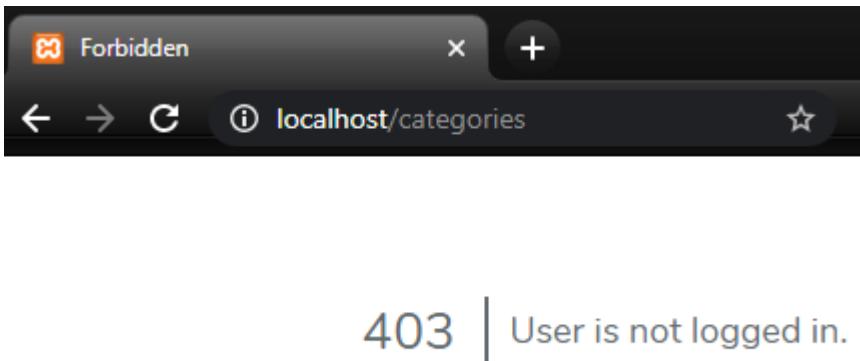
In de routes/web.php gaan we voor de categories routes een middleware gebruiken. Dit doen we zo:

```
24     Route::group(['middleware' => ['role:customer|sales|admin']], function () {
25         Route::get('categories/{category}/delete', 'CategoryController@delete')
26             ->name('categories.delete');
27         Route::resource('categories', 'CategoryController');
28     });
}
```

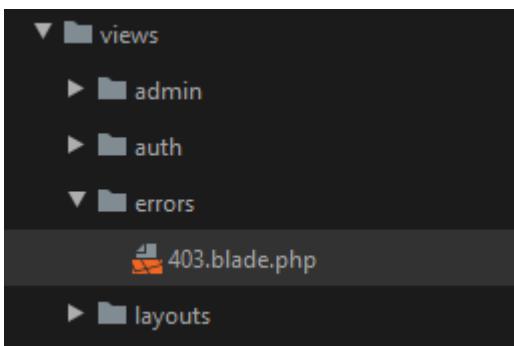
Dit zorgt ervoor dat als we in de browser naar categories gaan, eerst gekeken wordt of je rechten hebt.

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.

```
1 @extends('layouts.layout')
2
3 @section('content')
4     <h1 class="mt-5">Error</h1>
5     <div class="alert alert-danger">
6         {{ $exception->getMessage() }}
7     </div>
8
9 @endsection
```

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

The screenshot shows a browser window with the URL 'localhost/categories'. At the top is a dark blue header bar with navigation icons (back, forward, search) and the URL. Below it is a dark grey fixed navbar containing the text 'Fixed navbar' and links for 'Home', 'Link', 'Disabled', and 'Admin'. The main content area has a light gray background. In the center, the word 'Error' is displayed in large, bold, black font. Below it is a red rectangular box containing the text 'User is not logged in.' in white.

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

Auth binnen onze eigen layout

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 browser window with the URL 'localhost/login'. The page title is 'Laravel'. On the right side, there are links for 'Login' and 'Register'. The main content is a 'Login' form. It contains two input fields: one for 'E-Mail Address' and one for 'Password'. Below these is a checkbox labeled 'Remember Me'. At the bottom are two buttons: a blue 'Login' button and a link 'Forgot Your Password?'. The entire form is enclosed in a light gray box.

Op dit moment heeft alles van auth zijn eigen layout. Dit willen we natuurlijk niet. We willen gewoon onze eigen layout hebben. Hiervoor zullen de views aangepast moeten worden van de auth module.

Als eerst wil ik de mogelijkheid krijgen om in te loggen vanuit de website. Dit wil ik rechts bovenin onze layout neerzetten, waar nu de search balk is.



Dit gaan we bewerken in onze masterpage, dus in *layouts/layout.blade.php*

(en hopelijk heb je de *layouts/app.blade.php* niet weggegooid, want hier staan nu wat dingen in die we in onze eigen layout gaan verwerken)

In *layout.blade.php* gaan we eerst aan de slag met de nav. Het formulier gaan we weggooien, want we hoeven nu even geen search in het menu te hebben. In de plaats hiervan komt een stukje richting login en register form, of we zien dat we ingelogd zijn.

```
</li>
</ul>
<form class="form-inline mt-2 mt-md-0">
  <input class="form-control mr-sm-2" type="text" placeholder="Search" aria-label="Search">
  <button class="btn btn-outline-success my-2 my-sm-0" type="submit">Search</button>
</form>
</div>
</nav>
```

In de layouts/app.blade.php zie je een stuk met right side of navbar.

Kopieer de gehele en zet deze op de plek waar nu onze form stond. (ja, 31 regels)

```
<!-- Right Side Of Navbar -->
<ul class="navbar-nav ml-auto">
    <!-- Authentication Links -->
    @guest
        |
```

Dat gedeelte van de header wordt dus:

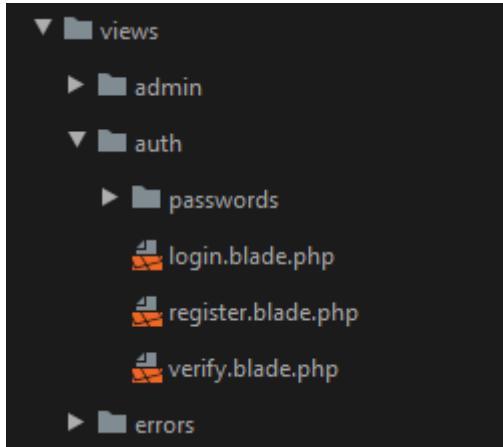
```
61     |         </div>
62     |         </li>
63     |     </ul>
64     |     <!-- Right Side Of Navbar -->
65     |     <ul class="navbar-nav ml-auto">
66         <!-- Authentication Links -->
67         @guest
68             |         <li class="nav-item">
69                 |             <a class="nav-link" href="{{ route('login') }}>{{ __('Login') }}</a>
70             </li>
71             @if (Route::has('register'))
72                 |             <li class="nav-item">
73                     |                 <a class="nav-link" href="{{ route('register') }}>{{ __('Register') }}</a>
74                 </li>
75             @endif
76         @else
77             |         <li class="nav-item dropdown">
78                 |             <a id="navbarDropdown" class="nav-link dropdown-toggle" href="#" role="button"
79                     |                 data-toggle="dropdown" aria-haspopup="true" aria-expanded="false" v-pre>
80                     |                     {{ Auth::user()->name }} <span class="caret"></span>
81                 </a>
82                 |             <div class="dropdown-menu dropdown-menu-right" aria-labelledby="navbarDropdown">
83                     |                 <a class="dropdown-item" href="{{ route('logout') }}"
84                         |                     onclick="event.preventDefault();
85                         |                         document.getElementById('logout-form').submit();">
86                         |                         {{ __('Logout') }}
87                     </a>
88                     |                     <form id="logout-form" action="{{ route('logout') }}" method="POST" style="margin-bottom: 0;">
89                         |                         @csrf
90                     </form>
91                 </div>
92             </li>
93         @endguest
94     </ul>
95     </div>
96     | </nav>
```

Je ziet hier al iets in van @guest (niet ingelogd), waar je links krijgt naar login en register.

Tevens zie je in de else een dropdown waar we de username gaan zien en uit kunnen loggen.

De dropdown heeft de bootstrap app.js nodig. Deze zit al in onze layout dus zou al moeten werken.

In alle bestanden van Auth kunnen we nu de masterpage veranderen.



Op dit moment staat er in alle bestanden binnen de auth map (voorbeeld van *login.blade.php*):

```
@extends('layouts.app')
@section('content')
```

Verander dit in: *@extends('layouts.layout')*

Zelf vind ik het prettig dat alles op dezelfde plek staat. Hierdoor heb ik nog een `<h1 class="mt-5">` ingezet zoals je hieronder ziet. Hier zet ik nog de titel van de pagina in, zodat het bij elke pagina hetzelfde is.

```
@extends('layouts.layout')
@section('content')
<h1 class="mt-5">Login</h1>
<div class="container">
```

Dit moet je dus ook doen in: *register.blade.php* , *verify.blade.php* , *confirm.blade.php* , *email.blade.php* en *reset.blade.php*

Je kan ook nog de *home.blade.php* nog veranderen, waarmee je hetzelfde kan doen.

Vanzelf sprekend zet je dan natuurlijk wel de correcte tekst in de `<h1>`

Nu we zover zijn kan je ook de *layouts/app.blade.php* verwijderen. Wordt alleen maar verwarring als je bijna dezelfde mapnaam hebt.

Binnen het project gaan we eerst aan de slag met allerlei beheer onderdelen. We gaan dus zorgen dat je standaard ingelogd moet zijn, ook vanaf de homepage.

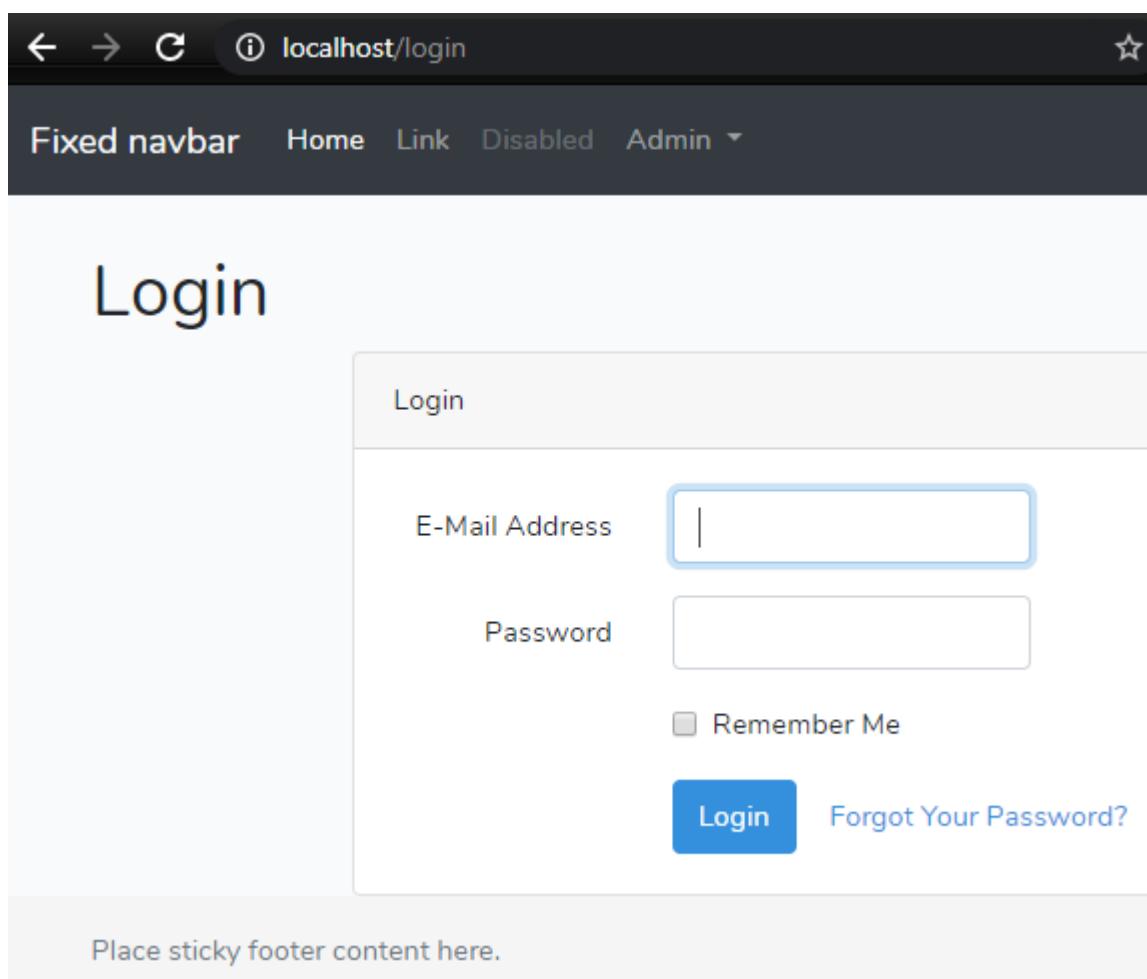
Hier voor roepen we niet meer meteen een view op als we op localhost/ komen, maar de HomeController.

```
16  /*Route::get('/', function () {
17      return view('welcome');
18  });
19
20  Route::get( uri: '/',  action: 'HomeController@index')->name( name: 'start');
```

Omdat in de HomeController een construct staat met middleware('auth'), wordt verwacht dat je ingelogd bent. Ben je niet ingelogd krijg je het loginscherm te zien.

```
public function __construct()
{
    $this->middleware( middleware: 'auth');
}
```

Ga ik nu naar localhost/, dan wordt ik meteen doorgestuurd zoals je hieronder ziet. (en de login is netjes ook met ons eigen layout)



Permissies in de Controller

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

```
public function run()
{
    app()['cache']->forget('spatie.permission.cache');

    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.

```
class CategoryController extends Controller
{
    /**
     * Set permissions on methods ...
     */
    public function __construct()
    {
        $this->middleware( middleware: 'auth');
        $this->middleware( middleware: 'permission:create category', ['only' => ['create', 'store']]);
        $this->middleware( middleware: 'permission:edit category', ['only' => ['edit', 'update']]);
        $this->middleware( middleware: 'permission:delete category', ['only' => ['delete', 'destroy']]);
    }
}
```

Net zoals bij de HomeController zal je zien dat we `middleware('auth')` gaan gebruiken in de construct.

De `__construct` methode is altijd de eerste methode van de class, dus bovenaan.

We zorgen eerst dat bekend is dat we de Auth van Laravel willen gebruiken. Daarna geven we aan de permission:create reviews de koppeling met de create en de store methode.

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.

Nu kunnen we de login gaan uitproberen en kijken of we inderdaad nu bepaalde rechten wel of niet hebben bij de categorie crud. Hiervoor hebben we wel de login nodig van de betreffende gebruiker. Deze staan gewoon in de UserSeeder.

```
factory( class: User::class, amount: 1)->create([
    'name'=>'Customer',
    'email'=>'customer@test.com',
    'password'=> bcrypt( value: 'test1234'))
->each(function (User $user) {
    $user->assignRole( ...roles: 'customer');
});

factory( class: User::class, amount: 1)->create([
    'name'=>'Sales',
    'email'=>'sales@test.com',
    'password'=> bcrypt( value: 'test1234'))
->each(function (User $user) {
    $user->assignRole( ...roles: 'sales');
});

factory( class: User::class, amount: 1)->create([
    'name'=>'Admin',
    'email'=>'admin@test.com',
    'password'=> bcrypt( value: 'test1234'))
->each(function (User $user) {
    $user->assignRole( ...roles: 'admin');
});
```

Als we naar localhost gaan zien we als eerst het loginscherm.

The screenshot shows a web browser window with the URL 'localhost/login' in the address bar. At the top, there's a fixed navigation bar with links for 'Home', 'Link', 'Disabled', and 'Admin'. To the right of the navigation bar are 'Login' and 'Register' buttons. Below the navigation bar is a large 'Login' form. The form has two input fields: 'E-Mail Address' and 'Password'. There is also a 'Remember Me' checkbox and two buttons at the bottom: a blue 'Login' button and a link 'Forgot Your Password?'. The overall layout is clean and modern.

We loggen in met sales@test.com en test1234:

The screenshot shows a dark-themed fixed navbar with links for Home, Link, Disabled, Admin, and Sales. Below the navbar, the word "Welcome" is displayed in a large font. A central box contains the text "Dashboard" at the top and "You are logged in!" below it.

Je ziet ook dat we nu geen Login en Register rechtsboven hebben, maar de naam van de gebruiker. Deze was bij mij *Sales*. Als we naar het overzicht van categories gaan, zien we wel alle categories nu.

Categories

| # | Category | Category details | Edit | Delete |
|---|------------------|-------------------------|----------------------|------------------------|
| 1 | Samara Daugherty | Details | Edit | Delete |
| 2 | Cynthia O'Keefe | Details | Edit | Delete |
| 3 | Lyric Strosin | Details | Edit | Delete |

We mogen als Sales ook een categorie wijzigen

The screenshot shows a browser window with the URL <localhost/categories/1/edit>. The page title is "Categories". It features a navigation bar with links for Index, Create, and Edit Category. Below the navigation is a "Category name" input field containing "Samara Daugherty". At the bottom is a blue "Update" button.

Maar als we een categorie proberen te verwijderen krijgen we de melding dat we de rechten niet hebben.

The screenshot shows a browser window with the address bar containing 'localhost/categories/1/delete'. Below the address bar is a dark navigation bar with links for 'Fixed navbar', 'Home', 'Link', 'Disabled', and 'Admin'. The main content area has a large 'Error' heading. A pink callout box contains the text 'User does not have the right permissions.'

Als we met de customer inloggen, zien we ook de admin voor categories. Dat is eigenlijk niet logisch, want er zou gewoon een publieke pagina moeten zijn. Om ze geen toegang meer te geven zal de routes.php aangepast moeten worden, zodat de customer niet meer erbij staat.

```
Route::group(['middleware' => ['role:sales|admin']], function () {
    Route::get('categories/{category}/delete', 'CategoryController@delete')
        ->name('categories.delete');
    Route::resource('categories', 'CategoryController');
});
```

Nu kunnen we als customer de hele admin niet meer zien.

The screenshot shows a browser window with the address bar containing 'localhost/categories'. Below the address bar is a dark navigation bar with links for 'Fixed navbar', 'Home', 'Link', 'Disabled', and 'Admin'. To the right of the navigation bar is a dropdown menu showing 'Customer'. The main content area has a large 'Error' heading. A pink callout box contains the text 'User does not have the right roles.'

We zijn bijna klaar, maar er is nog 1 ding wat niet klopt. Waarom krijgen we als customer in het menu wel de optie te zien om naar de admin te gaan.

Om dit te wijzigen moeten we naar de master layout, namelijk layouts/layout.blade.php
Hier kunnen we aangeven, dat als je een bepaalde rol hebt, je de dropdown voor admin kan zien.

```
@hasanyrole('sales|admin')
<li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="dropdown"
        aria-haspopup="true" aria-expanded="false">Admin
    </a>
    <div class="dropdown-menu" aria-labelledby="navbarDropdown">
        <a class="dropdown-item" href="{{ route('categories.index') }}>Category Admin</a>
        <a class="dropdown-item" href="#">Product Admin</a>
        <div class="dropdown-divider"></div>
        <a class="dropdown-item" href="#">Something else here</a>
    </div>
</li>
@endhasanyrole
```

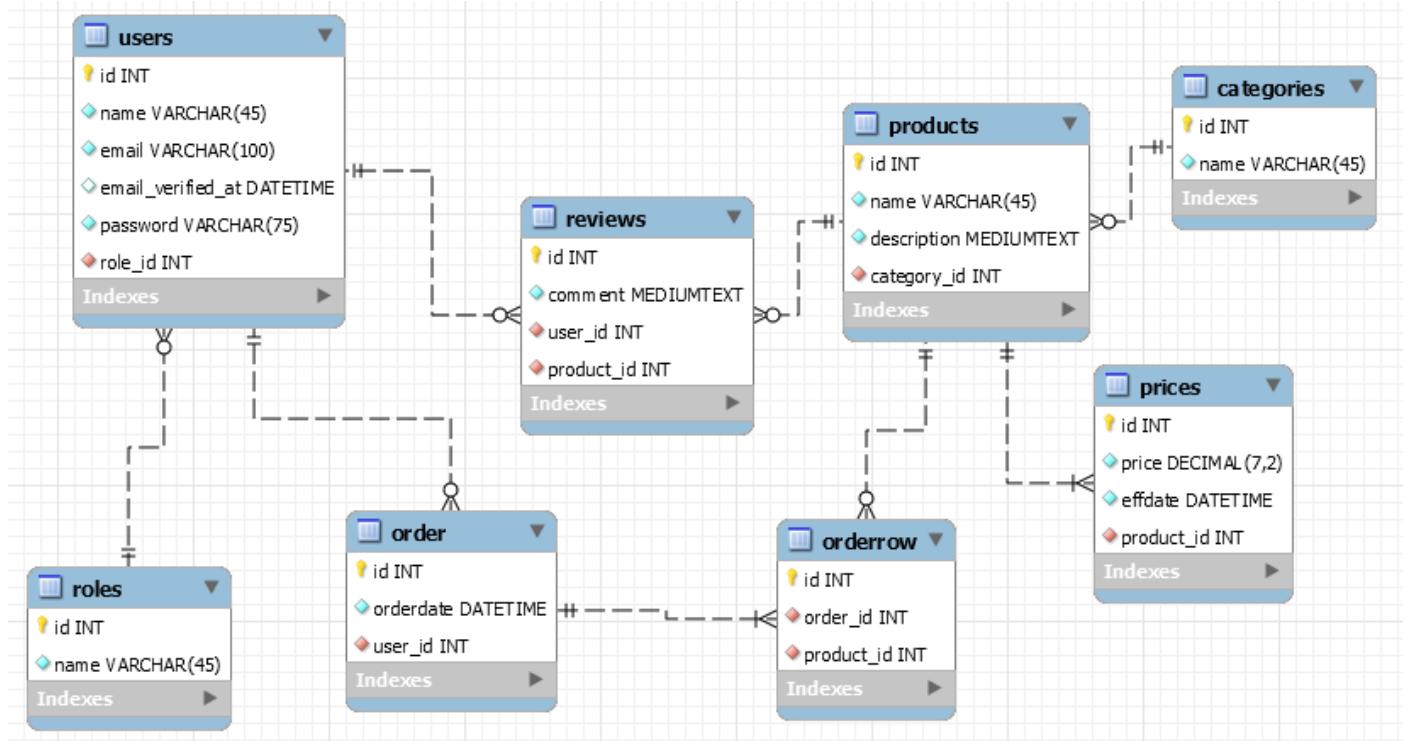
Nu krijgen we als customer niet meer het menu van de admin.

The screenshot shows a web browser window with a dark theme. At the top is a fixed navbar with the following items: 'Fixed navbar', 'Home', 'Link', and 'Disabled'. To the right of 'Link' is a dropdown menu with two items: 'Dashboard' and 'You are logged in!'. The user is identified as 'Customer'. The main content area displays the word 'Welcome'.

Natuurlijk kan het zijn dat straks rollen alleen bij een bepaalde admin mogen komen. Maar dat regelen we dan in het menu. Voor nu zien ze niet waar de admin is en als ze op de pagina willen komen zien ze een foutmelding dat ze geen rechten hebben.

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.

Models

Nu als eerst de models. Voor Product doen we dit met:

```
php artisan make:model --all Product
```

```
D:\laragon\www\workshop (master)
λ php artisan make:model --all Product
Model created successfully.
Factory created successfully.
Created Migration: 2020_05_15_084411_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:\laragon\www\workshop (master)
λ php artisan make:model --all Price
Model created successfully.
Factory created successfully.
Created Migration: 2020_05_15_085736_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.

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
{
    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 moest je in beide richtingen uitschrijven om de relatie te kunnen tekenen. Vanaf de review 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 method is naar welk model we toe verwijzgen. BijhasMany 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
{
    public function category(){
        return $this->belongsTo( related: Category::class);
    }

    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 eenhasMany 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 dehasMany gebruiken.

En dan de Price model.

```
class Price extends Model
{
    //
    public function product()
    {
        return $this->belongsToMany(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.

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.

```
7  class CreateProductsTable extends Migration
8  {
9      /** Run the migrations. ...*/
10     public function up()
11     {
12         Schema::create('products', function (Blueprint $table) {
13             $table->id();
14             $table->string('name', 45);
15             $table->text('description');
16             // foreign key naar category
17             $table->unsignedBigInteger('category_id');
18             $table->foreign('category_id')
19                 ->references('id')->on('categories')
20                 ->onDelete('restrict')
21                 ->onUpdate('restrict');
22             $table->timestamps();
23         });
24     }
25
26     /** Reverse the migrations. ...*/
27     public function down()
28     {
29         Schema::dropIfExists('products');
30     }
31 }
```

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.

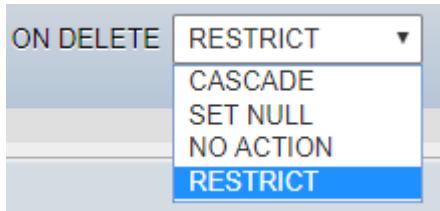
`$table->id();`

Alias of `$table->bigIncrements('id');`

Daarna geven we aan dat het een foreign key is, waar die naar verwijst met ->references en wat er moet gebeuren bij een delete en update met onDelete en OnUpdate. 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.

Dan de Price migration

```
7  class CreatePricesTable extends Migration
8  {
9      /** Run the migrations. ...*/
14     public function up()
15     {
16         Schema::create('prices', function (Blueprint $table) {
17             $table->id();
18             $table->decimal('price', 8, 2);
19             $table->dateTime('effdate');
20             $table->unsignedBigInteger('product_id');
21             $table->foreign('product_id')
22                 ->references('id')->on('products')
23                 ->onDelete('cascade')
24                 ->onUpdate('cascade');
25             $table->timestamps();
26         });
27     }
}
```

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.

Factory

Voor de factory en seed, gaan we nu met relaties 2 mogelijkheden gebruiken. Ik weet van te voren dat het niet de beste optie is om beide manieren nu te gebruiken, maar dit project is om dingen te leren en niet voor een productie omgeving.

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.

```
5  use App\Product;
6
7  use App\Category;
8
9  use Faker\Generator as Faker;
10
11 $factory->define( class: Product::class, function (Faker $faker) {
12
13     return [
14         'name' => $faker->name,
15         'description' => $faker->paragraph( nbSentences: 15),
16         'category_id' => Category::all()->random()->id
17     ];
18 });
19
```

Omdat we op regel 13 de model van Category gebruiken om een random id te krijgen, zullen we op regel 6 de class wel moeten toevoegen. En ja, op regel 13 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.

```
5  use App\Price;
6
7  use App\Product;
8
9  use Faker\Generator as Faker;
10
11 $factory->define( class: Price::class, function (Faker $faker) {
12
13     return [
14         'price' => $faker->randomFloat( nbMaxDecimals: 2, min: 2, max: 6),
15         'effdate' => $faker->dateTime,
16         'product_id' => Product::all()->random()->id
17     ];
18 });
19
```

Je ziet hier nog niet het verschil, omdat ik nu bij de ProductFactory gewoon category ingevuld heb. Eigenlijk is dit onnodig vanwege wat ik zo bij de seed ga doen.

Seed

Bij de seed doen we eerst even Price. Dit lijkt gewoon net zoals de seed van de Category.

```
5  class PriceSeeder extends Seeder
6  {
7      /** Run the database seeds. ...*/
8      public function run()
9      {
10          factory( class: App\Price::class, amount: 500)->create();
11      }
12 }
```

Maar nu de ProductSeed. We kunnen dat op dezelfde manier doen, zodat je random producten aan een categorie koppelt, maar je kan het ook anders doen. Wel hebben we hiervoor de CategorySeed voor nodig.

Op dit moment staat in de CategorySeed dit:

```
6  class CategorySeeder extends Seeder
7  {
8      /** Run the database seeds. ...*/
9      public function run()
10     {
11         //
12         factory( class: Category::class, amount: 10)->create();
13     }
14 }
```

Wat we gaan doen is, dat we per categorie nu een aantal producten gaan maken. Op deze manier zorgen we ervoor dat elke categorie met zekerheid een aantal producten heeft.

```
3  use Illuminate\Database\Seeder;
4  use App\Product;
5  use App\Category;
6
7  class CategorySeeder extends Seeder
8  {
9      /** Run the database seeds. ...*/
10     public function run()
11     {
12         //
13         factory( class: Category::class, amount: 3)->create()
14             ->each(function($category)
15             {
16                 $category->product()->saveMany(factory( class: Product::class, amount: 10)
17                     ->create(['category_id' => $category->id]));
18             });
19     }
20 }
```

Maar waarom zeg ik nu, dat je met deze methode zekerheid hebt?

Je maakt eerst een categorie aan. Per categorie maken we 10 producten aan, die gekoppeld worden met die category_id. Als je de factory gebruikt zoals nu, kan het voorkomen dat een categorie geen product heeft. Hierdoor is de manier in de Seed veiliger om te gebruiken als je in elke categorie een product wilt hebben.

Bij Price is nu eigenlijk hetzelfde. Stel je voor, we hebben 10 categories, 50 producten en 50 prijzen. Zoals nu de factory is, kan het voorkomen dat een product geen prijs heeft. Dan wordt meteen het programmeren van de controller een stuk moeilijker.

Als we de CategorySeeder nog gaan bekijken zien we nog meer.

```
3  use Illuminate\Database\Seeder;
4  use App\Product;
5  use App\Category;
6
7  class CategorySeeder extends Seeder
8  {
9      /** Run the database seeds. ...*/
10     public function run()
11     {
12         /**
13          factory( class: Category::class, amount: 3)->create()
14             ->each(function($category)
15             {
16                 $category->product()->saveMany(factory( class: Product::class, amount: 10)
17                   ->create(['category_id' => $category->id]));
18             });
19         }
20     }
21 }
```

Je ziet op regel 20: \$category->product()

Hebben we opeens een methode product() beschikbaar ?????

Het is niet opeens. Als we naar de model van Category kijken zie je daar een public function product()

```
class Category extends Model
{
    protected $fillable = ['name'];

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

Hier komt de product in de Seed dan ook vandaag. Als we een categorie hebben, kunnen we de relatie die beschreven is in de model gebruiken om ervoor te zorgen dat dit ook gebeurd in de database. Dit werkt natuurlijk ook andersom.

Toch ben ik zelf nog niet tevreden. We kunnen nog steeds een probleem gaan krijgen met de Products, dit omdat het nu nog echt mogelijk is om een product zonder prijs te hebben.

We gaan dit dan ook even nabootsen, zodat je snapt waardoor het is.

In de CategorySeeder geven we aan dat we 3 categories krijgen, met ieder 10 products. Totaal 30 products dus.

Laten we dan ook maar 30 prices genereren. Ik verander dus alleen de hoeveelheid.

```
class PriceSeeder extends Seeder
{
    /**
     * Run the database seeds. ...
     */
    public function run()
    {
        factory( App\Price::class, 30)->create();
    }
}
```

We moeten nog even de PriceSeeder toevoegen aan de DatabaseSeeder

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

Nu kunnen we gaan testen.

We zorgen dat we nieuwe tabellen hebben met verse data.

```
D:\laragon\www\workshop (master)
└ 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 (0.04 seconds)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (0.02 seconds)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (0.01 seconds)
Migrating: 2020_05_10_143304_create_permission_tables
Migrated: 2020_05_10_143304_create_permission_tables (0.26 seconds)
Migrating: 2020_05_10_191303_create_categories_table
Migrated: 2020_05_10_191303_create_categories_table (0.01 seconds)
Migrating: 2020_05_15_084411_create_products_table
Migrated: 2020_05_15_084411_create_products_table (0.05 seconds)
Migrating: 2020_05_15_085736_create_prices_table
Migrated: 2020_05_15_085736_create_prices_table (0.04 seconds)
Seeding: RoleAndPermissionSeeder
Seeded: RoleAndPermissionSeeder (0.05 seconds)
Seeding: UserSeeder
Seeded: UserSeeder (0.28 seconds)
Seeding: CategorySeeder
Seeded: CategorySeeder (0.07 seconds)
Seeding: PriceSeeder
Seeded: PriceSeeder (0.07 seconds)
Database seeding completed successfully.
```

Nu check ik de prices tabel in de database. Hierbij sorteert ik meteen op product_id

| SELECT * FROM `prices` ORDER BY `prices`.`product_id` ASC | | | | | | | | | | | |
|---|---|----------|-------------|--|------------|---------------|---------------------|---------------|---------------------|----------------------|------------|
| | 1 ▾ | > | >> | | Toon alles | Aantal rijen: | 25 ▾ | Filter rijen: | Zoek in deze tabel | Sorteren op sleutel: | Geen |
| · Opties | | | | | | | | | | | |
| | ← T → | | | | | | | | | | |
| | Klik op het pijltje om de zichtbaarheid van de kolom te wijzigen. | | | | | | | | | | |
| <input type="checkbox"/> | Wijzigen | Kopiëren | Verwijderen | | id | price | effdate | product_id | 1 | created_at | updated_at |
| <input type="checkbox"/> | Wijzigen | Kopiëren | Verwijderen | | 1 | 5.13 | 1979-10-07 08:11:26 | 1 | 2020-05-15 19:47:36 | 2020-05-15 19:47:36 | |
| <input type="checkbox"/> | Wijzigen | Kopiëren | Verwijderen | | 8 | 2.16 | 1989-01-29 10:04:45 | 1 | 2020-05-15 19:47:36 | 2020-05-15 19:47:36 | |
| <input type="checkbox"/> | Wijzigen | Kopiëren | Verwijderen | | 12 | 5.98 | 1973-04-04 23:29:56 | 3 | 2020-05-15 19:47:36 | 2020-05-15 19:47:36 | |
| <input type="checkbox"/> | Wijzigen | Kopiëren | Verwijderen | | 14 | 4.20 | 2017-01-09 09:42:59 | 4 | 2020-05-15 19:47:36 | 2020-05-15 19:47:36 | |
| <input type="checkbox"/> | Wijzigen | Kopiëren | Verwijderen | | 17 | 2.68 | 1997-02-09 15:41:44 | 4 | 2020-05-15 19:47:36 | 2020-05-15 19:47:36 | |
| <input type="checkbox"/> | Wijzigen | Kopiëren | Verwijderen | | 13 | 3.45 | 1982-01-09 12:52:01 | 5 | 2020-05-15 19:47:36 | 2020-05-15 19:47:36 | |
| <input type="checkbox"/> | Wijzigen | Kopiëren | Verwijderen | | 27 | 2.02 | 1994-09-25 16:52:50 | 5 | 2020-05-15 19:47:36 | 2020-05-15 19:47:36 | |

Zoals je ziet, hebben we 3 prijzen voor product_id = 1, maar ik zie helemaal geen product_id = 2.

Dit zou je moeilijkheden geven met het programmeren. Je zou in oplossingen moeten denken zoals een left-join. Makkelijker is om de seed aan te passen. Dit gaan we dan ook doen.

Je zal merken dat ik eigenlijk hetzelfde doe met product en price, als met category en product

```
3  use Illuminate\Database\Seeder;
4  use App\Product;
5  use App\Category;
6  use App\Price;
7
8  class CategorySeeder extends Seeder
9  {
10     /** Run the database seeds. ...*/
11     public function run()
12     {
13         /**
14          factory( class: Category::class, amount: 3)->create()
15          ->each(function($category)
16          {
17              $category->product()->saveMany(factory( class: Product::class, amount: 10)
18                  ->create(['category_id' => $category->id])
19                  ->each(function($product)
20                  {
21                      $product->price()->saveMany(factory( class: Price::class, amount: 3))
22                          ->create(['product_id' => $product->id]);
23                  });
24          });
25      });
26  }
27
28  }
29
30 }
```

En ja, het wordt nu wel wat ingewikkelder. Maar zo krijgen we per product 3 prijzen.

In de DatabaseSeeder heb ik de PriceSeeder even als commentaar staan. Zo kan ik testen of ik 90 prijzen nu krijg: $3 * 10 = 30 * 3 = 90$.

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

We voeren opnieuw de migrate en seed uit

```
D:\laragon\www\workshop (master)
λ 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 (0.02 seconds)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (0.02 seconds)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (0.01 seconds)
Migrating: 2020_05_10_143304_create_permission_tables
Migrated: 2020_05_10_143304_create_permission_tables (0.27 seconds)
Migrating: 2020_05_10_191303_create_categories_table
Migrated: 2020_05_10_191303_create_categories_table (0.01 seconds)
Migrating: 2020_05_15_084411_create_products_table
Migrated: 2020_05_15_084411_create_products_table (0.04 seconds)
Migrating: 2020_05_15_085736_create_prices_table
Migrated: 2020_05_15_085736_create_prices_table (0.04 seconds)
Seeding: RoleAndPermissionSeeder
Seeded: RoleAndPermissionSeeder (0.05 seconds)
Seeding: UserSeeder
Seeded: UserSeeder (0.28 seconds)
Seeding: CategorySeeder
Seeded: CategorySeeder (0.28 seconds)
Database seeding completed successfully.
```

En in de prices tabel zien we 90 resultaten. Elk product heeft ook inderdaad nu 3 prijzen.

| Weergave van records 0 - 24 (90 totaal, Query duurde 0.0005 seconden.) [product_id: 1... - 9...] | | | | | | | | | | |
|--|---|---|----|-------|---------------------|------------|---------------------|---------------------|---------------------|--|
| SELECT * FROM `prices` ORDER BY `product_id` ASC | | | | | | | | | | |
| <input type="button" value="1"/> <input type="button" value=">"/> <input type="button" value=">>"/> <input type="checkbox"/> Toon alles Aantal rijen: <input type="button" value="25"/> Filter rijen: <input type="text" value="Zoek in deze tabel"/> Sorteren op sleutel: <input type="text" value="prices_pr"/> | | | | | | | | | | |
| + Opties | ← | → | id | price | effdate | product_id | 1 | created_at | updated_at | |
| <input type="checkbox"/> | | | 1 | 5.25 | 1984-11-05 12:19:12 | 1 | 1 | 2020-05-15 19:57:49 | 2020-05-15 19:57:49 | |
| <input type="checkbox"/> | | | 2 | 4.35 | 2013-02-21 12:49:22 | 1 | 1 | 2020-05-15 19:57:49 | 2020-05-15 19:57:49 | |
| <input type="checkbox"/> | | | 3 | 3.43 | 2011-07-12 17:04:22 | 1 | 1 | 2020-05-15 19:57:49 | 2020-05-15 19:57:49 | |
| <input type="checkbox"/> | | | 4 | 3.87 | 2019-07-17 09:44:17 | 2 | 2 | 2020-05-15 19:57:49 | 2020-05-15 19:57:49 | |
| <input type="checkbox"/> | | | 5 | 4.47 | 2002-07-27 13:51:56 | 2 | 2 | 2020-05-15 19:57:49 | 2020-05-15 19:57:49 | |
| <input type="checkbox"/> | | | 6 | 2.37 | 1998-07-29 10:05:45 | 2 | 2 | 2020-05-15 19:57:49 | 2020-05-15 19:57:49 | |
| <input type="checkbox"/> | | | 7 | 4.20 | 2011-03-15 00:31:47 | 3 | 2020-05-15 19:57:49 | 2020-05-15 19:57:49 | | |
| <input type="checkbox"/> | | | 8 | 5.82 | 1974-04-29 19:03:16 | 3 | 2020-05-15 19:57:49 | 2020-05-15 19:57:49 | | |
| <input type="checkbox"/> | | | 9 | 4.31 | 2013-01-07 10:56:14 | 3 | 2020-05-15 19:57:49 | 2020-05-15 19:57:49 | | |

We kunnen nog wel de PriceSeeder erbij zetten om extra random prijzen ervin te zetten, als toevoegen boven op de al gemaakte. De ProductSeeder kunnen we niet gebruiken, omdat we dan een Product aanmaken zonder dat er een Price aan gekoppeld is.

Nu onze database en inhoud ervan geheel klopt, kunnen we verder naar de crud.

Product Controller

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.

```
10     /** Display a listing of the resource. ...*/
11
12     public function index()
13     {
14
15         $products = Product::all();
16
17         return view( 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.

```
index.blade.php ×
1 @extends('layouts.layout')
2
3 @section('content')
4     <h1 class="mt-5">Products</h1>
5
6     @if (session('message'))
7         <div class="alert alert-success">
8             {{ session('message') }}
9         </div>
10    @endif
11
12    <nav class="nav">
13        <ul class="nav nav-tabs">
14            <li class="nav-item">
15                <a class="nav-link active" href="{{ route('products.index') }}">Index</a>
16            </li>
17            <li class="nav-item">
18                <a class="nav-link" href="{{ route('products.create') }}">Create</a>
19            </li>
20        </ul>
21    </nav>
```

De titel is aangepast naar Products.

De routes van de linkjes zijn ook naar de product routes.

```

23      <table class="table .table-striped">
24          <thead class="thead-dark">
25              <tr>
26                  <th scope="col">#</th>
27                  <th scope="col">Product</th>
28                  <th scope="col">Categorie</th>
29                  <th scope="col">Huidige prijs</th>
30                  <th scope="col">Product details</th>
31                  <th scope="col">Edit</th>
32                  <th scope="col">Delete</th>
33          </tr>
34      </thead>
35      <tbody>
36          @foreach($products as $product)
37              <tr>
38                  <td scope="row">{{ $product->id }}</td>
39                  <td>{{ $product->name }}</td>
40                  <td></td>
41                  <td></td>
42                  <td><a href="{{ route('products.show', ['product' => $product->id]) }}>Details</a></td>
43                  <td><a href="{{ route('products.edit', ['product' => $product->id]) }}>Edit</a></td>
44                  <td><a href="{{ route('products.delete', ['product' => $product->id]) }}>Delete</a></td>
45          </tr>
46      @endforeach
47      </tbody>
48  </table>
49 @endsection

```

De tabel is uitgebreid met een aantal kolommen. Zoals ik al zei, de naam van het product, de categorie en de huidige prijs zouden handig zijn in het overzicht. In de @foreach is dus een hoop aangepast, omdat we nu wel met producten te maken hebben.

Om nu de index te laten werken zullen we de routes nog moeten updaten

```

23 Route::get( uri: '/home',   action: 'HomeController@index')->name( name: 'home');
24
25 Route::get( uri: 'products/{product}/delete',   action: 'ProductController@delete')
26     ->name( name: 'products.delete');
27 Route::resource( name: '/products',   controller: 'ProductController');

```

Voor nu, eerst nog even buiten de middleware. Dit is in het begin makkelijker als je gaat testen en dus minder stappen nodig om de test uit te voeren. Later gaan we de beveiliging wel toevoegen. Ook voor products alvast de delete route toegevoegd.

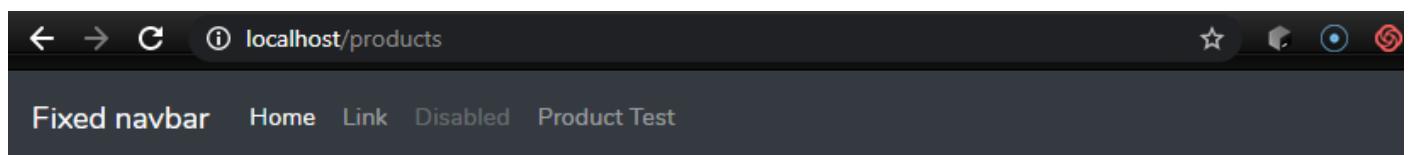
Als laatst nog even zorgen dat we wel vanuit onze site bij de pagina kunnen. Dan denk je, oohhh, in de dropdown even aanpassen

```
@hasanyrole('sales|admin')  
  
<li class="nav-item dropdown">  
    <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="dropdown"  
        aria-haspopup="true" aria-expanded="false">Admin  
    </a>  
    <div class="dropdown-menu" aria-labelledby="navbarDropdown">  
        <a class="dropdown-item" href="{{ route('categories.index') }}>Category Admin</a>  
        <a class="dropdown-item" href="{{ route('products.index') }}>Product Admin</a>  
        <div class="dropdown-divider"></div>  
        <a class="dropdown-item" href="#">Something else here</a>  
    </div>  
</li>  
@endhasanyrole
```

Maar, we wilde makkelijk testen, en door het hier te zetten moet je ingelogd zijn en de rol sales of admin hebben. Hierdoor voeg ik nog een extra link er buiten toe, maar wel met de duidelijkheid dat het voor testen is. (zodat deze later weer verwijderd kan worden)

```
<li class="nav-item">  
    <a class="nav-link" href="{{ route('products.index') }}>Product Test</a>  
</li>
```

Op de site zien we dan ook netjes deze link zonder ingelogd te zijn. De dropdown zien we niet. Verder hebben we een net overzicht van producten als we naar de link gaan. Deze moet nog wel aangepast worden met de categorie en huidige prijs erbij.



Products

| Products | | | | | | |
|----------|----------------------|-----------|---------------|-----------------|------|--------|
| | | Index | | Create | | |
| # | Product | Categorie | Huidige prijs | Product details | Edit | Delete |
| 1 | Dr. Dean Kilback | | | Details | Edit | Delete |
| 2 | Dr. Geovany Lindgren | | | Details | Edit | Delete |
| 3 | Doug Satterfield | | | Details | Edit | Delete |

Als eerst gaan we de categorie erbij zetten. Deze 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 relaties al beschreven:

```
class Product extends Model
{
    public function category(){
        return $this->belongsTo( related: Category::class);
    }
}
```

Hierdoor kunnen we nu categorie gebruiken

```
@foreach($products as $product)
<tr>
    <td scope="row">{{ $product->id }}</td>
    <td>{{ $product->name }}</td>
    <td>{{ $product->category->name }}</td>
    <td></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.

Products

[Index](#) [Create](#)

| # | Product | Categorie | Huidige prijs | Product details | Edit | Delete |
|---|----------------------|-----------|---------------|-------------------------|----------------------|------------------------|
| 1 | Dr. Dean Kilback | Schoenen | | Details | Edit | Delete |
| 2 | Dr. Geovany Lindgren | Schoenen | | Details | Edit | Delete |
| 3 | Doug Satterfield | Schoenen | | Details | Edit | Delete |

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

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

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

    // laatste prijs in product overzicht
    public function latest_price()
    {
        return $this->hasOne( related: 'App\Price')->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 controller gaan we gebruiken maken van Eager Loading. (<https://laravel.com/docs/7.x/eloquent-relationships#eager-loading>)

We gaan de relatie van latest_price gebruiken bij het inladen van producten. Je ziet dus nu Product::with() erbij staan voor de eager loading, waarbij je kan aangeven welke relatie je wilt gebruiken.

```
class ProductController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        $products = Product::with('latest_price')->get();
        return view('admin.products.index', compact('products'));
    }
}
```

In de template gebruiken we dan, net zoals bij de naam van de categorie, de relatie om de prijs te krijgen.

```
@foreach($products as $product)
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| {{ $product->id }} | {{ $product->name }} | {{ $product->category->name }} | {{ $product->latest_price->price }} | Details | Edit | Delete |

@endforeach
```

In ons overzicht op de pagina zie je dan nu ook de laatste prijs staan. De index is tot zo ver klaar.

| # | Product | Categorie | Huidige prijs | Product details | Edit | Delete |
|---|----------------------|-----------|---------------|-------------------------|----------------------|------------------------|
| 1 | Dr. Dean Kilback | Schoenen | 4.35 | Details | Edit | Delete |
| 2 | Dr. Geovany Lindgren | Schoenen | 3.87 | Details | Edit | Delete |
| 3 | Doug Satterfield | Schoenen | 4.31 | Details | Edit | Delete |

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.

```
    /** Show the form for creating a new resource. ...*/
    public function create()
    {
        $categories = Category::all();
        return view('admin.products.create', compact('categories'));
    }
```

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.

```
1  @extends('layouts.layout')
2
3  @section('content')
4      <h1 class="mt-5">Products</h1>
5
6      @if ($errors->any())
7          <div class="alert alert-danger">
8              <ul>
9                  @foreach ($errors->all() as $error)
10                      <li>{{ $error }}</li>
11                  @endforeach
12              </ul>
13          </div>
14      @endif
15
16      <nav class="nav">
17          <ul class="nav nav-tabs">
18              <li class="nav-item">
19                  <a class="nav-link" href="{{ route('products.index') }}">Index</a>
20              </li>
21              <li class="nav-item">
22                  <a class="nav-link active" href="{{ route('products.create') }}">Create</a>
23              </li>
24          </ul>
25      </nav>
```

Het formulier is natuurlijk wel een stukje groter dan bij categories.

```
27 <form method="POST" action="{{ route('products.store') }}>
28     @csrf
29     <div class="form-group">
30         <label for="name">Productname</label>
31         <input type="text" name="name" class="form-control" id="name" aria-describedby="nameHelp"
32             placeholder="Enter productname">
33     </div>
34     <div class="form-group">
35         <label for="description">Description</label>
36         <textarea class="form-control" name="description" id="description" rows="3"></textarea>
37     </div>
38     <div class="form-group">
39         <label for="price">Price</label>
40         <input type="text" class="form-control" name="price" id="price" aria-describedby="priceHelp"
41             placeholder="Enter price">
42     </div>
43     <div class="form-group">
44         <label for="category_id">Category</label>
45         <select name="category_id" id="category_id" class="form-control">
46             @foreach($categories as $category)
47                 <option value="{{ $category->id }}>{{ $category->name }}</option>
48             @endforeach
49         </select>
50     </div>
51     <button type="Submit" class="btn btn-primary">Submit</button>
52 </form>
53 @endsection
```

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.

Als we dan naar het resultaat gaan kijken, zien we een net formulier bij de create staan.

← → ⌂ ⓘ localhost/products/create

Fixed navbar Home Link Disabled Product Test

Products

Index Create

Productname

Enter productname

Description

Price

Enter price

Category

Schoenen

Submit

Bij de category hebben we netjes een dropdown gekregen. De create is dan ook hiermee klaar.

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.

```
33     /** Store a newly created resource in storage. ...*/
39     public function store(Request $request)
40     {
41         $product = new Product();
42         $product->name = $request->name;
43         $product->description = $request->description;
44         $product->category_id = $request->category_id;
45         $product->save();
46     }
```

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.

```
35     /** Store a newly created resource in storage. ...*/
41     public function store(Request $request)
42     {
43         $product = new Product();
44         $product->name = $request->name;
45         $product->description = $request->description;
46         $product->category_id = $request->category_id;
47         $product->save();

48

49         $price = new Price();
50         $price->price = $request->price;
51         $price->effdate = Carbon::now();
52         $price->product_id = $product->id;
53         $price->save();
54     }
```

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. 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:\laragon\www\workshop (master)
λ 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

```
7  class ProductStoreRequest extends FormRequest
8  {
9      /**
10     * Determine if the user is authorized to make this request. ...
11     */
12    public function authorize()
13    {
14        return true;
15    }
16
17
18
19    /**
20     * Get the validation rules that apply to the request. ...
21     */
22    public function rules()
23    {
24        return [
25            'name' => 'required|unique:products|max:45',
26            'description' => 'required',
27            'price' => 'required|numeric|max:999999.99'
28        ];
29    }
30
31
32}
```

Als we nu naar het geheel kijken van de store methode zien we dit.

```
36     /** Store a newly created resource in storage. ....*/
37
38     public function store(ProductStoreRequest $request)
39     {
40
41         $product = new Product();
42         $product->name = $request->name;
43         $product->description = $request->description;
44         $product->category_id = $request->category_id;
45         $product->save();
46
47
48         $price = new Price();
49         $price->price = $request->price;
50         $price->effdate = Carbon::now();
51         $price->product_id = $product->id;
52         $price->save();
53
54
55
56         return redirect()->route('products.index')->with('message', 'Product toegevoegd');
57     }

```

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.

Products

Index Create

Productname

Description

Price

Category

We krijgen ook netjes een bericht dat het product is toegevoegd.

Products

Product toegevoegd

[Index](#)

[Create](#)

| # | Product | Categorie | Huidige prijs | Product details | Edit | Delete |
|---|----------------------|-----------|---------------|-------------------------|----------------------|------------------------|
| 1 | Dr. Dean Kilback | Schoenen | 4.35 | Details | Edit | Delete |
| 2 | Dr. Geovany Lindgren | Schoenen | 3.87 | Details | Edit | Delete |

Probeer ik een grotere prijs dan die we hebben ingevuld bij de request, krijg ik netjes een foutmelding

Products

- The price may not be greater than 999999.99.

Het enige is, de velden die we net hebben ingevuld in het formulier zijn leeg. Dit is eigenlijk niet netjes. Hiervoor gaan we in de layout ervoor zorgen dat we de data die we hadden ingevuld in het formulier er weer inzetten.

Binnen Laravel hebben we hiervoor de optie in onze blade bestanden: value="{{old('name')}}"

Met old() kunnen we dus de geposte waarde terugkrijgen in ons formulier, zodra er een foutmelding is gekomen.

We gaan de create.blade dus aanpassen op de volgende manier:

```
<form method="POST" action="{{ route('products.store') }}>
    @csrf
    <div class="form-group">
        <label for="name">Productname</label>
        <input type="text" name="name" class="form-control" id="name" aria-describedby="nameHelp"
               placeholder="Enter productname" value="{{old('name')}}">
    </div>
    <div class="form-group">
        <label for="description">Description</label>
        <textarea class="form-control" name="description" id="description" rows="3">{{old('description')}}
```

Bij text inputs is het niet zo moeilijk. Je zet daar gewoon de old() in de value, waardoor de waarde erin zal komen. Bij de dropdown doen we deze controle binnen de @foreach. We controlleren of de \$category->id hetzelfde is als de oude waarde. Zodra dat is, zetten we 'selected' erbij.

Zodra we dit gaan testen, waarbij we een te grote prijs neerzetten, zie je dat de ingevulde waarde netjes in het formulier blijven staan.

Products

- The price may not be greater than 999999.99.

[Index](#) [Create](#)

Productname

Description

Price

Category

Submit

Nu we dit hebben zijn we klaar met de store, want validatie wordt nu goed verwerkt.

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.

```
59     /** Display the specified resource. ...*/
60
61     public function show(Product $product)
62     {
63
64         return view('admin.products.show', compact('product'));
65     }
66
67 }
```

In de view hebben we de show van categorie gekopieerd en aangepast.

```
1 @extends('layouts.layout')
2
3 @section('content')
4     <h1 class="mt-5">Products</h1>
5
6     <nav class="nav">
7         <ul class="nav nav-tabs">
8             <li class="nav-item">
9                 <a class="nav-link" href="{{ route('products.index') }}">Index</a>
10            </li>
11            <li class="nav-item">
12                <a class="nav-link" href="{{ route('products.create') }}">Create</a>
13            </li>
14            <li class="nav-item">
15                <a class="nav-link active" href="{{ route('products.show', [
16                    'product' => $product->id
17                ]) }}>Product Details</a>
18            </li>
19        </ul>
20    </nav>
```

Dus even netjes het menu aangepast zodat het over de products gaat.

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.

```
22 <div class="card">
23   <div class="card-header">
24     Product
25   </div>
26   <div class="card-body">
27     <h2 class="card-title">{{ $product->name }}</h2>
28     <p class="card-text">{{ $product->description }}</p>
29     <p class="card-text">Categorie: {{ $product->category->name }}</p>
30     <p class="card-text">Price: {{ $product->latest_price->price }}</p>
31   </div>
32 </div>
33 @endsection
```

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.

```
public function price()
{
    return $this->hasMany( related: 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.

```
<p class="card-text">Prices: </p>
<table class="table .table-striped">
    <thead class="thead-dark">
        <tr>
            <th scope="col">#</th>
            <th scope="col">Prijs</th>
            <th scope="col">Ingangsdatum</th>
        </tr>
    </thead>
    <tbody>
        @foreach($product->price->sortByDesc('effdate') as $item)
            <tr>
                <td scope="row">{{ $item->id }}</td>
                <td>{{ $item->price }}</td>
                <td>{{ $item->effdate }}</td>
            </tr>
        @endforeach
    </tbody>
</table>

</div>
```

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:

Products

[Index](#) [Create](#) [Product Details](#)

Product

Dr. Dean Kilback

Totam quam nobis at aut molestias unde. Veniam consequatur et omnis molestias. Ipsam quo quasi repellat nihil atque quibusdam aperiam. Saepe corrupti aliquam voluptatem ea explicabo dolorem. Voluptas ducimus reiciendis aut ullam magnam eos atque neque. Eum tempore tempore excepturi sint culpa sint dolor. Aut vel dolores quidem eos ut nihil. Officiis eum non voluptas iure sit. Ducimus est officia illum consequatur dolorem eum. Exercitationem nobis qui voluptatibus est esse aliquid. Hic aut repudiandae aut.

Categorie: Schoenen

Prices:

| # | Prijs | Ingangsdatum |
|---|-------|---------------------|
| 2 | 4.35 | 2013-02-21 12:49:22 |
| 3 | 3.43 | 2011-07-12 17:04:22 |
| 1 | 5.25 | 1984-11-05 12:19:12 |

Volgens mij een mooi resultaat om in detail een product met de bijbehorende prijzen te zien.

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.

```
70     /** Show the form for editing the specified resource. ...*/
71
72     public function edit(Product $product)
73     {
74
75         $categories = Category::all();
76
77         return view('admin.products.edit', compact('product', 'categories'));
78     }
79
80 }
```

In de edit.blade eerst goed checken of alle links op orde zijn.

```
1 @extends('layouts.layout')
2
3 @section('content')
4     <h1 class="mt-5">Products</h1>
5
6     @if($errors->any())
7         <div class="alert alert-danger">
8             <ul>
9                 @foreach ($errors->all() as $error)
10                     <li>{{ $error }}</li>
11                 @endforeach
12             </ul>
13         </div>
14     @endif
15
16     <nav class="nav">
17         <ul class="nav nav-tabs">
18             <li class="nav-item">
19                 <a class="nav-link" href="{{ route('products.index') }}">Index</a>
20             </li>
21             <li class="nav-item">
22                 <a class="nav-link" href="{{ route('products.create') }}">Create</a>
23             </li>
24             <li class="nav-item">
25                 <a class="nav-link active" href="{{ route('products.edit', [
26                     'product' => $product->id
27                 ]) }}">Edit Product</a>
28             </li>
29         </ul>
30     </nav>
```

De form is ook niet al te lastig.

```
31   <form method="POST" action="{{ route('products.update', ['product' => $product->id]) }}>
32     @method('PUT')
33     @csrf
34     <div class="form-group">
35       <label for="name">Product name</label>
36       <input type="text" name="name" class="form-control" id="name" aria-describedby="productNameHelp"
37           value="{{ old('name', $product->name) }}>
38     </div>
39     <div class="form-group">
40       <label for="description">Description</label>
41       <textarea name="description" id="description" rows="3"
42           class="form-control"> {{ old('description', $product->description) }}</textarea>
43     </div>
44     <div class="form-group">
45       <label for="price">Price</label>
46       <input type="text" class="form-control" name="price" id="price" aria-describedby="priceHelp"
47           value="{{ old('price', $product->latest_price->price) }}>
48     </div>
49     <div class="form-group">
50       <label for="category_id">Category</label>
51       <select name="category_id" id="category_id" class="form-control">
52         @foreach($categories as $category)
53           <option value="{{ $category->id }}>
54             @if(old('category_id',$product->category_id) == $category->id)
55               selected
56             @endif
57             >{{ $category->name }}</option>
58         @endforeach
59       </select>
60     </div>
61     <button type="submit" class="btn btn-primary">Update</button>
62   </form>
63 @endsection
```

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!

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.

Verder heeft een product een naam die uniek is. Hierdoor zullen we een update request nodig hebben. Deze gaan we eerst aanmaken.

```
D:\laragon\www\workshop (master)
└ 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.

```
7  class ProductUpdateRequest extends FormRequest
8  {
9      /** Determine if the user is authorized to make this request. ...*/
10     public function authorize()
11     {
12         return true;
13     }
14
15     /** Get the validation rules that apply to the request. ...*/
16     public function rules()
17     {
18         $product = $this->route('param: 'product');
19
20         return [
21             'name' => 'required|max:45|unique:products,name,' . $product->id,
22             'description' => 'required',
23             'price' => 'required|numeric|max:999999.99'
24         ];
25     }
26
27 }
```

Als we dan naar de update methode gaan kijken ziet deze er zo uit.

```
83     /* Update the specified resource in storage. ...*/
90     public function update(ProductUpdateRequest $request, Product $product)
91     {
92         $product->name = $request->name;
93         $product->description = $request->description;
94         $product->category_id = $request->category_id;
95         $product->save();
96
97         if ($product->latest_price->price != $request->price){
98             $price = new Price();
99             $price->price = $request->price;
100            $price->effdate = Carbon::now();
101            $price->product_id = $product->id;
102            $price->save();
103        }
104
105        return redirect()->route('products.index')
106            ->with('message', 'Product geupdate');
107    }
```

We hebben de ProductUpdateRequest gebruikt, dus vergeet niet dat deze bij use moet komen boven de class. De gegevens van \$product worden opgeslagen bij product. Er wordt gekeken of de prijs die we invullen anders is dan de huidige prijs. Zo ja, dan wordt er een nieuwe prijs aan het product gekoppeld.

We veranderen alles wat bij id=1 stond. Het resultaat ziet er goed uit.

Products

Product geupdate

| Huidige prijs | | | | | | |
|---------------|----------------------|-------------|---------|-----------------|------|--------|
| # | Product | Categorie | | Product details | Edit | Delete |
| 1 | Een msi laptop | Cale Pouros | 3000.10 | Details | Edit | Delete |
| 2 | Dr. Geovany Lindgren | Schoenen | 3.87 | Details | Edit | Delete |

Als we dan proberen een error te krijgen, door een te grote prijs te gebruiken. Expres de product naam hetzelfde gehouden, om te checken of we wel dezelfde naam kunnen gebruiken.

Products

- The price may not be greater than 999999.99.

En als we een product hetzelfde willen noemen als een ander product, krijgen we netjes uitgelegd dat de naam al in gebruik is. De edit is nu klaar.

Products

- The name has already been taken.

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.

```
109     /** Show the form for deleting the specified resource. ...*/
115     public function delete(Product $product)
116     {
117         return view( view: 'admin.products.delete', compact('product'));
118     }
```

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.

Het menu even correct gemaakt, dat we bij de delete zitten.

```
1 @extends('layouts.layout')
2
3 @section('content')
4     <h1 class="mt-5">Products</h1>
5
6     @if($errors->any())
7         <div class="alert alert-danger">
8             <ul>
9                 @foreach ($errors->all() as $error)
10                     <li>{{ $error }}</li>
11                 @endforeach
12             </ul>
13         </div>
14     @endif
15
16     <nav class="nav">
17         <ul class="nav nav-tabs">
18             <li class="nav-item">
19                 <a class="nav-link" href="{{ route('products.index') }}">Index</a>
20             </li>
21             <li class="nav-item">
22                 <a class="nav-link" href="{{ route('products.create') }}">Create</a>
23             </li>
24             <li class="nav-item">
25                 <a class="nav-link active" href="{{ route('products.delete', [
26                     'product' => $product->id]) }}">Delete Product</a>
27             </li>
28         </ul>
29     </nav>
```

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!

```
31      <form method="POST" action="{{ route('products.destroy', ['product' => $product->id]) }}>
32          @method('DELETE')
33          @csrf
34          <div class="form-group">
35              <label for="name">Product name</label>
36              <input type="text" name="name" class="form-control" id="name" aria-describedby="productNameHelp"
37                  value="{{ $product->name }}" disabled="disabled">
38          </div>
39          <div class="form-group">
40              <label for="description">Description</label>
41              <textarea name="description" id="description" rows="3"
42                  class="form-control" disabled="disabled"> {{ $product->description }}</textarea>
43          </div>
44          <div class="form-group">
45              <label for="price">Price</label>
46              <input type="text" class="form-control" name="price" id="price" aria-describedby="priceHelp"
47                  value="{{ $product->latest_price->price }}" disabled="disabled">
48          </div>
49          <div class="form-group">
50              <label for="category">Category</label>
51              <input type="text" class="form-control" name="category" id="pcategory" aria-describedby="categoryHelp"
52                  value="{{ $product->category->name }}" disabled="disabled">
53          </div>
54          <button type="submit" class="btn btn-primary">Delete</button>
55      </form>
56  @endsection
```

Als we gaan kijken hoe dit er dan uit ziet, zien we een net formulier met gegevens erin, die we kunnen deleten.

Products

[Index](#)[Create](#)[Delete Product](#)

Product name

Een msi laptop

Description

Een killing i9 zit erin

Price

3000.10

Category

Cale Pouros

[Delete](#)

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('message', '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.

Products

Product deleted

[Index](#) [Create](#)

| # | Product | Categorie | Huidige prijs | Product details | Edit | Delete |
|---|----------------------|-----------|---------------|-------------------------|----------------------|------------------------|
| 2 | Dr. Geovany Lindgren | Schoenen | 3.87 | Details | Edit | Delete |
| 3 | Doug Satterfield | Schoenen | 4.31 | Details | Edit | Delete |
| 4 | Hulda Green | Schoenen | 5.16 | Details | Edit | Delete |

En weet je wat het mooie is. De prices die er aan gekoppeld zijn, staan ook niet meer in de database.

✓ Weergave van records 0 - 24 (90 totaal, Query duurde 0,0003 seconden.)

```
SELECT * FROM `prices`
```

| 1 | > | >> | Toon alles | Aantal rijen: 25 | Filter rijen: Zoek in deze tabel | Sorteren op sleutel: | Geli |
|--------------------------|--------------------------|--------------------------|-----------------------------|------------------|----------------------------------|----------------------|---|
| + Opties | | | | | | | |
| | Wijzigen | Kopiëren | Verwijderen | 4 | 3.87 | 2019-07-17 09:44:17 | 2 2020-05-15 19:57:49 2020-05-15 19:57:49 |
| | Wijzigen | Kopiëren | Verwijderen | 5 | 4.47 | 2002-07-27 13:51:56 | 2 2020-05-15 19:57:49 2020-05-15 19:57:49 |
| | Wijzigen | Kopiëren | Verwijderen | 6 | 2.37 | 1998-07-29 10:05:45 | 2 2020-05-15 19:57:49 2020-05-15 19:57:49 |
| | Wijzigen | Kopiëren | Verwijderen | 7 | 4.20 | 2011-03-15 00:31:47 | 3 2020-05-15 19:57:49 2020-05-15 19:57:49 |
| | Wijzigen | Kopiëren | Verwijderen | 8 | 5.82 | 1974-04-29 19:03:16 | 3 2020-05-15 19:57:49 2020-05-15 19:57:49 |

Dit is het mooie aan database relaties goed configureren. In de migration hebben we dit aangegeven bij prices.

```
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->unsignedBigInteger('product_id');
17             $table->foreign('product_id')
18                 ->references('id')->on('products')
19                 ->onDelete('cascade')
20                 ->onUpdate('cascade');
21             $table->timestamps();
22         });
23     }
24 }
```

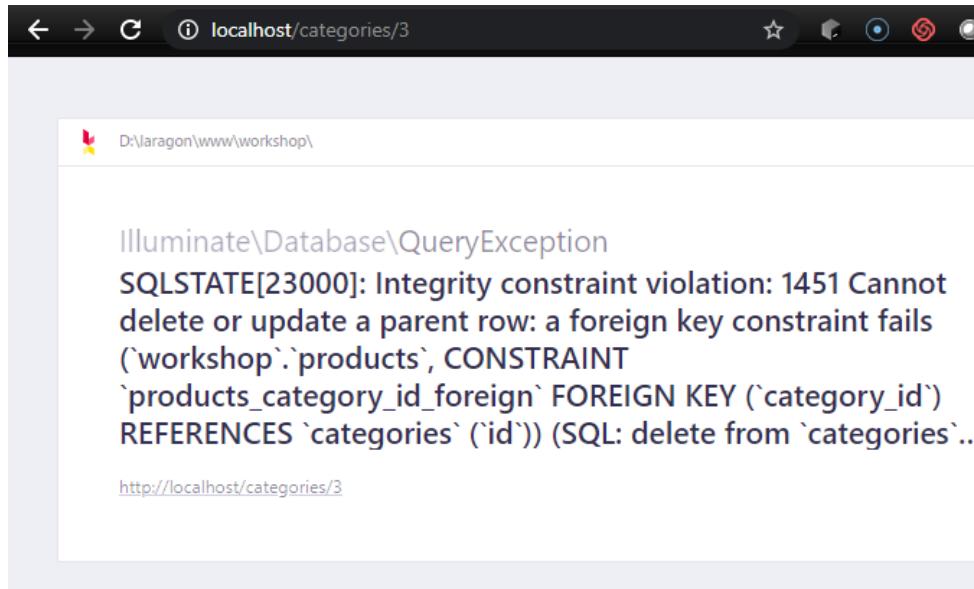
Dus, zodra een product wordt verwijderd, is de relatie bekent, staat er onDelete('cascade') erbij, waardoor ook meteen de prices die aan het product stonden gekoppeld verwijderd worden. De database blijft dan netjes op orde, want anders zouden we prices hebben die niet aan een product gekoppeld stonden.

Dit heet ook wel, data integriteit.

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.



A screenshot of a web browser window. The address bar shows 'localhost/categories/3'. The page content displays an error message:

```
Illuminate\Database\QueryException  
SQLSTATE[23000]: Integrity constraint violation: 1451 Cannot  
delete or update a parent row: a foreign key constraint fails  
(`workshop`.`products`, CONSTRAINT  
'products_category_id_foreign' FOREIGN KEY (`category_id`)  
REFERENCES `categories`(`id`)) (SQL: delete from `categories`..  
http://localhost/categories/3
```

Deze error komt namelijk doordat we in de database een restrictie hebben opgelegd.

In de product migration staat dit



```
7 class CreateProductsTable extends Migration
8 {
9     /** Run the migrations. ...*/
10    public function up()
11    {
12        Schema::create('products', function (Blueprint $table) {
13            $table->id();
14            $table->string('name', 45);
15            $table->text('description');
16            // foreign key naar category
17            $table->unsignedBigInteger('category_id');
18            $table->foreign('category_id')
19                ->references('id')->on('categories')
20                ->onDelete('restrict')
21                ->onUpdate('restrict');
22            $table->timestamps();
23        });
24    }
25 }
```

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.

```
110      /** Remove the specified resource from storage. ...*/
111  
112      public function destroy(Category $category)
113      {
114          try {
115              $category->delete();
116          }catch (Throwable $e){
117              report($e);
118              return redirect()->route( route: 'categories.index')->with('message', 'Categorie is niet leeg.
119                                  Er mogen geen producten meer in de categorie staan.');
120          }
121      }
122      return redirect()->route( route: 'categories.index')->with('message', 'Categorie verwijderd');
123  }
124  
125  }
126  
```

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.

```
3   namespace App\Http\Controllers;
4
5   use App\Category;
6   use App\Http\Requests\CategoryStoreRequest;
7   use App\Http\Requests\CategoryUpdateRequest;
8   use Illuminate\Http\Request;
9   use Throwable;
10
11  class CategoryController extends Controller
12  {
```

Als we het nu weer proberen zien we een melding dat het niet gelukt is.

Categories

Categorie is niet leeg. Er mogen geen producten meer in de categorie staan.

De is nog wel in het groen, terwijl groen lijkt alsof het is gelukt.

In de index.blade van categories gaan we dus nog een wijziging moeten maken. We zorgen ervoor dat we een alert-danger hebben voor session('wrong')

```
@section('content')
    <h1 class="mt-5">Categories</h1>

    @if (session('message'))
        <div class="alert alert-success">
            {{ session('message') }}
        </div>
    @endif

    @if(session('wrong'))
        <div class="alert alert-danger">
            {{ session('wrong') }}
        </div>
    @endif
```

We updaten de delete methode. Let op dat alleen als het fout gaat je dan wrong gebruikt.

```
111     /** Remove the specified resource from storage. ...*/
117     public function destroy(Category $category)
118     {
119         try {
120             $category->delete();
121         } catch (Throwable $e){
122             report($e);
123             return redirect()->route( route: 'categories.index')->with('wrong', 'Categorie is niet leeg.
124                                         Er mogen geen producten meer in de categorie staan.');
125         }
126     }
127 }
```

Bij het testen zien we dan ook netjes de melding in de goede kleur staan.

Categories

Categorie is niet leeg. Er mogen geen producten meer in de categorie staan.

Alles werkt nu weer zoals het de bedoeling is.

Unit & Feature testing

Omgeving

Nu we klaar zijn met de product crud gaan we aan de slag met testen. Je hebt net 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.



An elegant PHP Testing Framework

Pest is a Testing Framework with a focus on simplicity. It was carefully crafted to bring the joy of testing to PHP.

Voor Pest zullen we goed naar onze composer.json moeten kijken of het voldoet aan de minimale eisen, zodat alles gaat werken. (we volgen de installatie volgens de documentatie: <https://pestphp.com/docs/installation/>)

Installation

[Edit this page →](#)

Requires [PHP 7.3+](#)

1. First, your composer.json file must have these options: `"minimum-stability": "dev"`, and `"prefer-stable": true`.

Als eerst, we hebben php 7.4.5, dus we voldoen aan de eis om minimaal php 7.3 te hebben.

In onze composer.json staat ook dit:

```
51     "minimum-stability": "dev",
52     "prefer-stable": true,
```

We voldoen dus aan de eisen van stap 1.

Bij stap 2 staat er dit:

2. Then, make sure your PHPUnit dependency is set to `^9.0`:

```
composer require phpunit/phpunit:^9.0 --dev --update-with-dependencies
```

Pest will use your current `phpunit.xml`. If you don't have one, make sure you [download this file](#) and place it on the root of your project.

Hier voldoen we nog niet aan

```
20     "require-dev": {
21         "facade/ignition": "^2.0", 2.0.2
22         "fzaninotto/faker": "^1.9.1", v1.9.1
23         "mockery/mockery": "^1.3.1", 1.3.1
24         "nunomaduro/collision": "^4.1", v4.2.0
25         "phpunit/phpunit": "^8.5" 8.5.4
```

Hiervoor gaan we dus de update doen van phpunit door middel van composer. Dit doen we met: `composer require phpunit/phpunit:^9.0 --dev --update-with-dependencies`

```

D:\laragon\www\workshop (master)
λ composer require phpunit/phpunit:"^9.0" --dev --update-with-dependencies
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 2 installs, 19 updates, 0 removals
- Updating sebastian/version (2.0.1 => 3.0.0): Downloading (100%)
- Updating sebastian/type (1.1.3 => 2.0.0): Downloading (100%)
- Updating sebastian/resource-operations (2.0.1 => 3.0.0): Downloading (100%)
- Updating sebastian/recursion-context (3.0.0 => 4.0.0): Downloading (100%)
- Updating sebastian/object-reflector (1.1.1 => 2.0.0): Downloading (100%)
- Updating sebastian/object-enumerator (3.0.3 => 4.0.0): Downloading (100%)
- Updating sebastian/global-state (3.0.0 => 4.0.0): Downloading (100%)
- Updating sebastian/exporter (3.1.2 => 4.0.0): Downloading (100%)
- Updating sebastian/environment (4.2.3 => 5.1.0): Downloading (100%)
- Updating sebastian/diff (3.0.2 => 4.0.1): Downloading (100%)
- Updating sebastian/comparator (3.0.2 => 4.0.0): Downloading (100%)
- Installing sebastian/code-unit (1.0.2): Downloading (100%)
- Updating phpunit/php-timer (2.1.2 => 3.1.4): Downloading (100%)
- Updating phpunit/php-text-template (1.2.1 => 2.0.0): Downloading (100%)
- Installing phpunit/php-invoker (3.0.0): Downloading (100%)
- Updating phpunit/php-file-iterator (2.0.2 => 3.0.1): Downloading (100%)
- Updating phpunit/php-token-stream (3.1.1 => 4.0.1): Downloading (100%)
- Updating sebastian/code-unit-reverse-lookup (1.0.1 => 2.0.0): Downloading (100%)
- Updating phpunit/php-code-coverage (7.0.10 => 8.0.2): Downloading (100%)
- Updating symfony/polyfill-ctype (v1.15.0 => v1.17.0): Downloading (100%)
- Updating phpunit/phpunit (8.5.4 => 9.1.5): Downloading (100%)
phpunit/php-invoker suggests installing ext-pcntl (*)
Writing lock file
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
Discovered Package: facade/ignition
Discovered Package: fideloper/proxy
Discovered Package: fruitcake/laravel-cors
Discovered Package: laravel/tinker
Discovered Package: laravel/ui
Discovered Package: nesbot/carbon
Discovered Package: nunomaduro/collision
Discovered Package: spatie/laravel-permission
Package manifest generated successfully.

```

Je ziet dat er meteen allerlei updates meekomen door de optie --update-with-dependencies.

In de composer.json zien we wel meteen dat er een update is geweest.

```

20      "require-dev": {
21          "facade/ignition": "^2.0", 2.0.2
22          "fzaninotto/faker": "^1.9.1", v1.9.1
23          "mockery/mockery": "^1.3.1", 1.3.1
24          "nunomaduro/collision": "^4.1", v4.2.0
25          "phpunit/phpunit": "^9.0" 9.1.5

```

Voor Laravel, zullen we de Collision dependency nog moeten updaten.

composer require nunomaduro/collision:"^5.0" --dev --update-with-dependencies

```
D:\laragon\www\workshop (master)
λ composer require nunomaduro/collision:"^5.0" --dev --update-with-dependencies
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 0 installs, 4 updates, 0 removals
- Updating symfony/service-contracts (v2.0.1 => v2.1.2): Downloading (100%)
- Updating symfony/polyfill-php73 (v1.15.0 => v1.17.0): Downloading (100%)
- Updating symfony/polyfill-mbstring (v1.15.0 => v1.17.0): Downloading (100%)
- Updating nunomaduro/collision (v4.2.0 => v5.0.0-BETA1): Downloading (100%)
Writing lock file
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
Discovered Package: facade/ignition
Discovered Package: fideloper/proxy
Discovered Package: fruitcake/laravel-cors
Discovered Package: laravel/tinker
Discovered Package: laravel/ui
Discovered Package: nesbot/carbon
Discovered Package: nunomaduro/collision
Discovered Package: spatie/laravel-permission
Package manifest generated successfully.
```

Nu we aan de eisen voldoen kunnen we Pest binnen halen met `composer require pestphp/pest --dev`

```
D:\laragon\www\workshop (master)
λ composer require pestphp/pest --dev
Using version ^0.1.5 for pestphp/pest
./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 pestphp/pest (v0.1.5): Downloading (100%)
Writing lock file
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
```

Pest is nu binnen gehaald. Binnen Laravel zullen we Pest nog moeten installeren. Dit kan nu via artisan, want deze opties zijn er bij gekomen:

```
pest
pest:dataset          Create a new dataset file
pest:install           Creates Pest resources in your current PHPUnit test suite
pest:test              Create a new test file
```

Met `php artisan pest:install` zorgen we ervoor dat alles goed staat.

```
D:\laragon\www\workshop (master)
λ php artisan pest:install

[OK] `tests/Pest.php` created successfully.

[OK] `tests/Helpers.php` created successfully.
```

Gebruik van Pest / Phpunit

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:\laragon\www\workshop (master)
λ ./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:\laragon\www\workshop (master)
λ .\vendor\bin\pest

  PASS  Tests\Unit\ExampleTest
    ✓ basic test

  FAIL  Tests\Feature\ExampleTest
    • basic test

    ---

    • Tests\Feature\ExampleTest > basic test
      Expected status code 200 but received 302.
      Failed asserting that 200 is identical to 302.

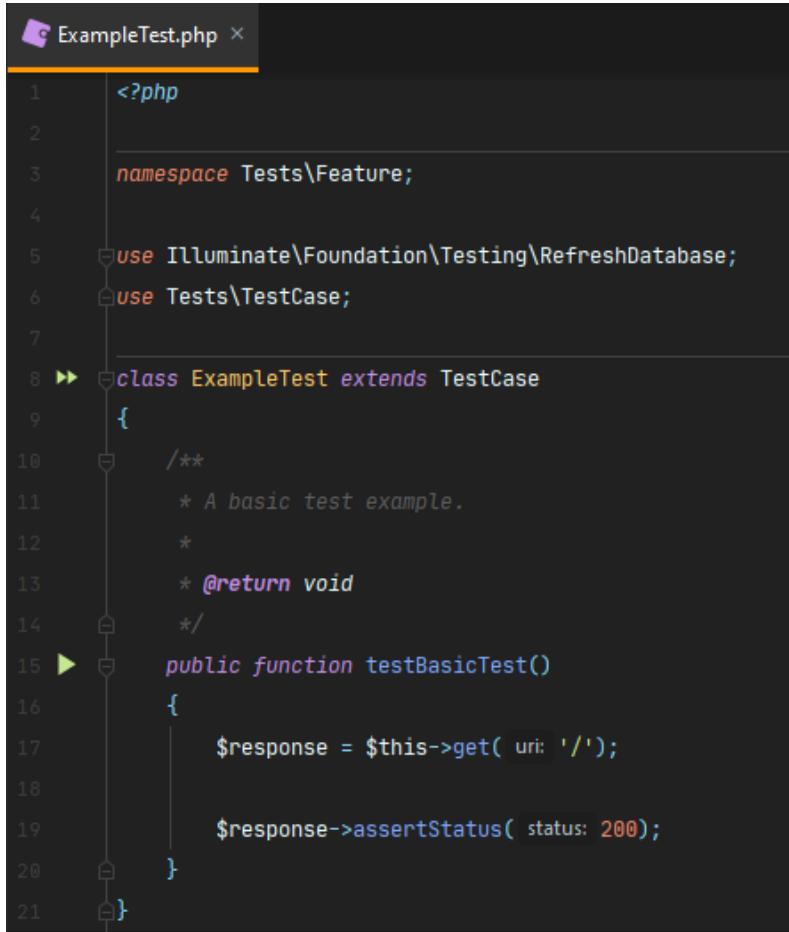
  at D:\laragon\www\workshop\vendor\phpunit\phpunit\src\Framework\Constraint\Constraint.php:119
  115|         if (!empty($description)) {
  116|             $failureDescription = $description . "\n" . $failureDescription;
  117|         }
  118|
→ 119|         throw new ExpectationFailedException(
  120|             $failureDescription,
  121|             $comparisonFailure
  122|         );
  123|     }

  1  D:\laragon\www\workshop\vendor\phpunit\phpunit\src\Framework\Constraint\IsIdentical.php:93
  PHPUnit\Framework\Constraint\Constraint::fail("Expected status code 200 but received 302.")

  2  D:\laragon\www\workshop\vendor\phpunit\phpunit\src\Framework\Assert.php:2228
  PHPUnit\Framework\Constraint\IsIdentical::evaluate("Expected status code 200 but received 302.")

Tests: 1 failed, 1 passed
Time: 0.21s
```

In de example test, te vinden in `tests/Feature/ExampleTest.php`, zie je dit staan.



```
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      * @return void
14      */
15     public function testBasicTest()
16     {
17         $response = $this->get('/');
18
19         $response->assertStatus( status: 200);
20     }
21 }
```

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:\laragon\www\workshop (master)
λ .\vendor\bin\phpunit
PHPUnit 9.1.5 by Sebastian Bergmann and contributors.

.F
2 / 2 (100%)

Time: 00:00.199, Memory: 20.00 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:\laragon\www\workshop\vendor\laravel\framework\src\Illuminate\Testing\TestResponse.php:185
D:\laragon\www\workshop\tests\Feature\ExampleTest.php:19

FAILURES!
Tests: 2, Assertions: 2, 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.

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

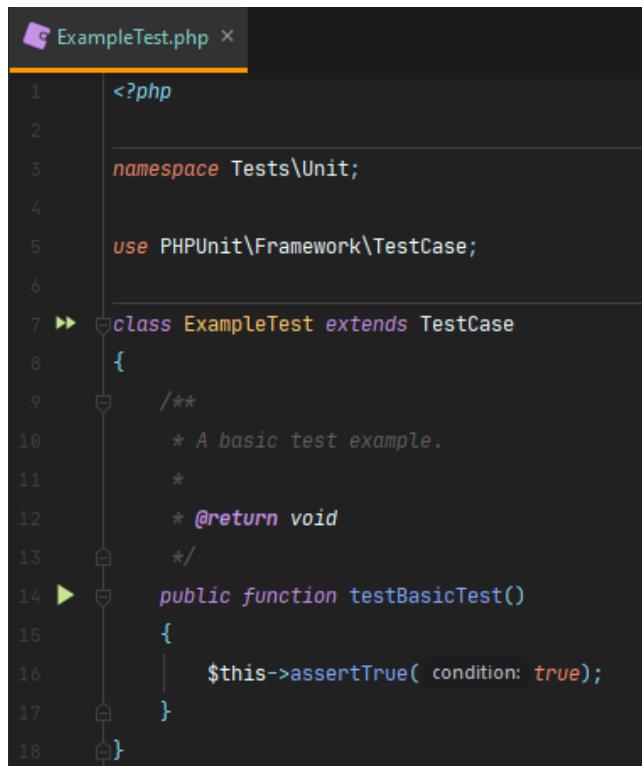
validatie is niet gelukt

500 Internal server error

Algemene server error

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`



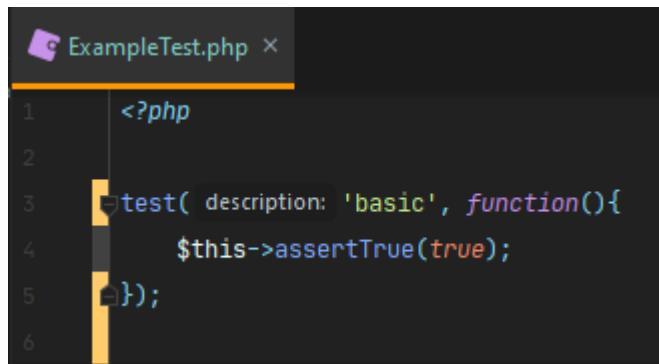
```
<?php

namespace Tests\Unit;

use PHPUnit\Framework\TestCase;

class ExampleTest extends TestCase
{
    /**
     * A basic test example.
     *
     * @return void
     */
    public function testBasicTest()
    {
        $this->assertTrue( 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^e test. Met Pest hoeven we dit allemaal niet meer in de test zelf te schrijven, waardoor een test veel simpeler wordt.



```
<?php

test( 'basic', function(){
    $this->assertTrue(true);
});
```

We gebruiken de functie `test`, waarin we de naam van de test aangeven. Verder voeren we dezelfde test uit. En dit is alles wat we nodig hebben.

Om even de error die we hadden bij Feature test niet steeds te krijgen, zullen we de methode daarin even als commentaar zetten

```
15     /*public function testBasicTest()
16     {
17         $response = $this->get('/');
18
19         $response->assertStatus(200);
20     }*/

```

Als we nu de gehele test uitvoeren wordt de unittest alleen gedaan. Je zal zien dat de 3 regels nu hetzelfde resultaat hebben als eerst.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest

  PASS TestsUnitExampleTest
  ✓ basic

  Tests: 1 passed
  Time: 0.04s

```

Het kan zelfs nog iets korter. Omdat assertTrue globaal beschikbaar is, kunnen we \$this-> weghalen. Je zal zien dat het nog steeds werkt

```
3     test( description: 'basic', function(){
4         assertTrue( condition: true);
5     });

```

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest

  PASS TestsUnitExampleTest
  ✓ basic

  Tests: 1 passed
  Time: 0.04s

```

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.

```
3     test( description: 'basic')->assertTrue( condition: true);

D:\laragon\www\workshop (master)
λ .\vendor\bin\pest

  PASS TestsUnitExampleTest
  ✓ basic

  Tests: 1 passed
  Time: 0.04s

```

Als we de conditie nu is false maken, zie je ook dat het resultaat aangeeft wat er dan precies fout is.

```
3 | test( description: 'basic')->assertTrue( condition: false);
```

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest

FAIL TestsUnitExampleTest
• basic

---
• TestsUnitExampleTest > basic
Failed asserting that false is true.

at D:\laragon\www\workshop\tests\Unit\ExampleTest.php:3
1| <?php
2|
→ 3| test('basic')->assertTrue(false);
4|
5|

1 D:\laragon\www\workshop\vendor\phpunit\phpunit\src\Framework\Constraint\Constraint.php:53
PHPUnit\Framework\Constraint\Constraint::fail("")

2 D:\laragon\www\workshop\vendor\phpunit\phpunit\src\Framework\Assert.php:2228
PHPUnit\Framework\Constraint\Constraint::evaluate("")
```

```
Tests: 1 failed
Time: 0.06s
```

Om ervoor te zorgen dat we niet steeds de foutmelding krijgen bij deze test, zetten we de conditie weer op true.

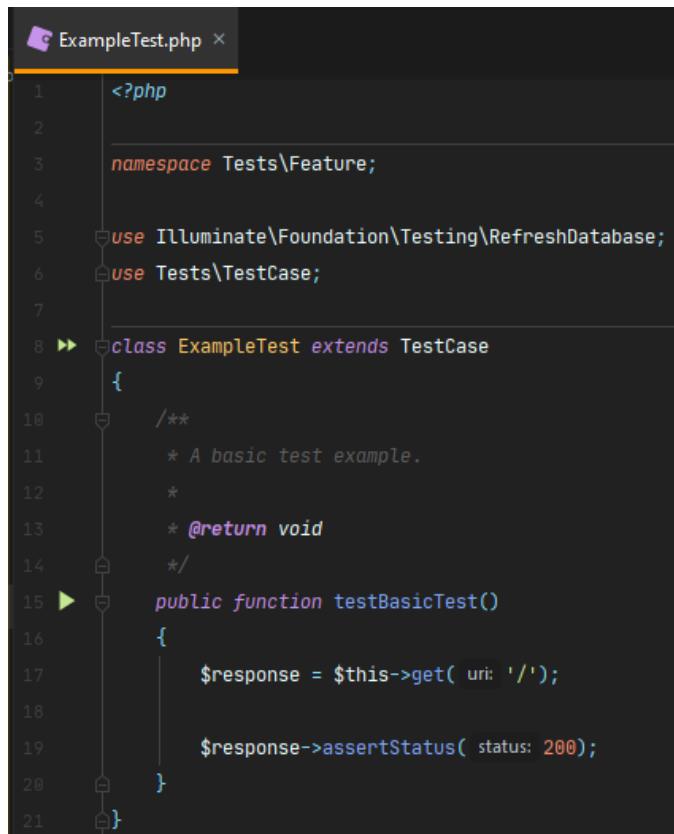
```
3 | test( description: 'basic')->assertTrue( condition: true);
```

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: (hier is de basistest dan nu niet meer als commentaar gezet)



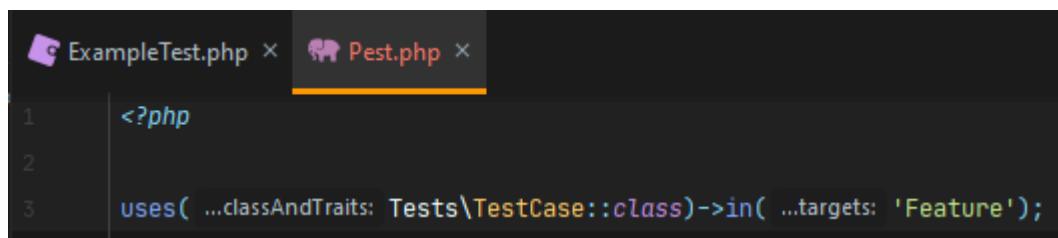
```
<?php

namespace Tests\Feature;

use Illuminate\Foundation\Testing\RefreshDatabase;
use Tests\TestCase;

class ExampleTest extends TestCase
{
    /**
     * A basic test example.
     *
     * @return void
     */
    public function testBasicTest()
    {
        $response = $this->get('/');
        $response->assertStatus( status: 200);
    }
}
```

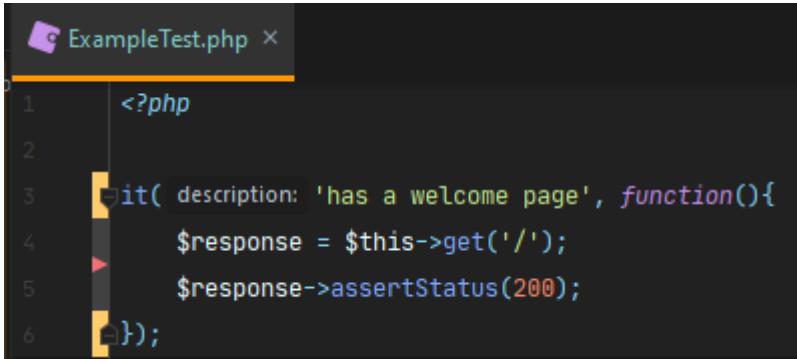
Voor een feature test hebben we wel alles nodig van Laravel. Eigenlijk heb je een boot nodig van Laravel. Gelukkig is Pest zo gemaakt dat dit niet meer hoeft. In Pest.php staat dat we de class TestCase altijd moeten gebruiken.



```
<?php

uses( ...classAndTraits: Tests\TestCase::class)->in( ...targets: 'Feature');
```

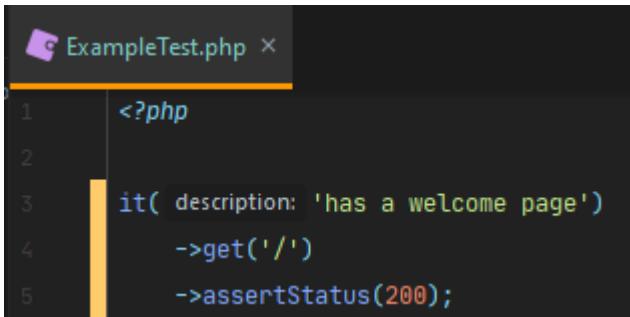
We kunnen dus de gehele test omschrijven voor wat bij Pest nodig is. Dit kan dan zo.



```
ExampleTest.php
1 <?php
2
3 it( description: 'has a welcome page', function(){
4     $response = $this->get('/');
5     $response->assertStatus(200);
6 });


```

En ja, tuurlijk kan je hier ook de chain gebruiken



```
ExampleTest.php
1 <?php
2
3 it( description: 'has a welcome page')
4     ->get('/')
5     ->assertStatus(200);


```

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.

```
✓ asserts true is true
```

```
✓ it asserts true is true
```

Als we de test nu uitvoeren zien we dit:

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest

  PASS TestsUnitExampleTest
    ✓ basic

  FAIL TestsFeatureExampleTest
    • it has a welcome page

    ---

    • TestsFeatureExampleTest > it has a welcome page
      Expected status code 200 but received 302.
      Failed asserting that 200 is identical to 302.

  at D:\laragon\www\workshop\tests\Feature\ExampleTest.php:5
    1| <?php
    2|
    3|     it('has a welcome page')
    4|         ->get('/')
  →  5|         ->assertStatus(200);
    6|

  1  D:\laragon\www\workshop\vendor\phpunit\phpunit\src\Framework\Constraint\IsIdentical.php:93
      PHPUNIT\Framework\Constraint::fail("Expected status code 200 but received 302.")

  2  D:\laragon\www\workshop\vendor\phpunit\phpunit\src\Framework\Assert.php:2228
      PHPUNIT\Framework\Constraint\IsIdentical::evaluate("Expected status code 200 but received 302.")

  Tests: 1 failed, 1 passed
  Time: 0.22s
```

Je ziet dat de unittest er wel doorkomt, maar de feature test niet. Logisch, want dit hadden we ook voordat we het als commentaar hadden gezet. Er wordt nog steeds verwacht dat we ingelogd zijn.

Wat we nu heel even doen, is juist een handige optie gebruiken binnen Pest, namelijk de skip(). Helaas is deze op dit moment niet beschikbaar als we de chain gebruiken. Hierdoor zullen we toch weer even terug moeten naar de vorige versie.

De chain versie met skip() werkt niet. De test wordt nog steeds uitgevoerd

```
3  it( description: 'has a welcome page')
  4    ->get('/')
  5    ->assertStatus(200)
  6    ->skip();
```

Als we de chain niet gebruiken, kan je wel de skip() gebruiken. Je zien ook dat bij de test er 1 is overgeslagen.

```
3  it( description: 'has a welcome page', function(){
  4    $response = $this->get('/');
  5    $response->assertStatus(200);
  6  })->skip();
```

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest

  PASS TestsUnitExampleTest
    ✓ basic

  WARN TestsFeatureExampleTest
    s it has a welcome page

  Tests: 1 skipped, 1 passed
  Time: 0.18s
```

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. De test zou er dan bijvoorbeeld zo uit zien.

```
12     it( description: 'validate emails', function($email){  
13         assertNotEmpty($email);  
14     });
```

Nu kunnen we met de variabel \$email data erin brengen. Zonder Pest, dus met alleen phpUnit zou dit zo gaan:

```
9      function getEmails(){  
10         return ['m.koningstein@tcrmbo.nl', 'info@tcrmbo.nl'];  
11     }  
12  
13     /**  
14      * @dataProvider getEmails()  
15     */  
16     it( description: 'validate emails', function($email){  
17         assertNotEmpty($email);  
18     });
```

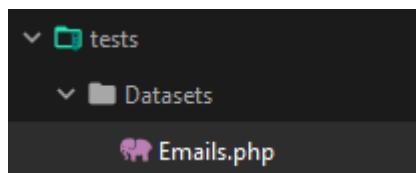
Met Pest kan het veel korter. We hebben al geen aparte functie nodig, maar kunnen het gewoon meegeven. Dit doen we met de with() functie.

```
9     it( description: 'validate emails', function($email){  
10         assertNotEmpty($email);  
11     })->with(['m.koningstein@tcrmbo.nl', 'info@tcrmbo.nl']);
```

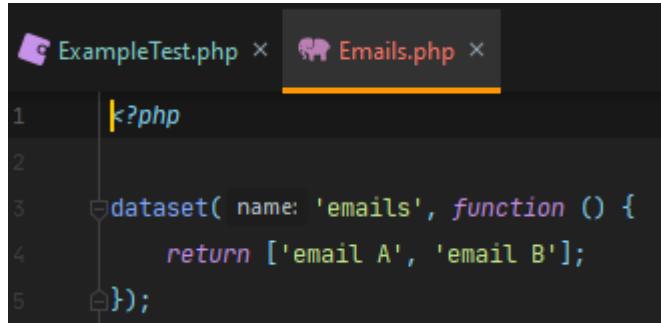
Nu is dit eigenlijk ook nog niet netjes. Stel je voor dat ik hier een array van 15 mailadressen wil checken. De code wordt dan weer niet ok. Hiervoor kunnen we met Pest een dataset aanmaken met [php artisan pest:dataset emails](#)

```
D:\laragon\www\workshop (master)  
λ php artisan pest:dataset emails  
  
[OK] `tests/Datasets/Emails.php` created successfully.
```

In de test directory is er een map gekomen voor datasets, waarin het bestand Emails.php staat.

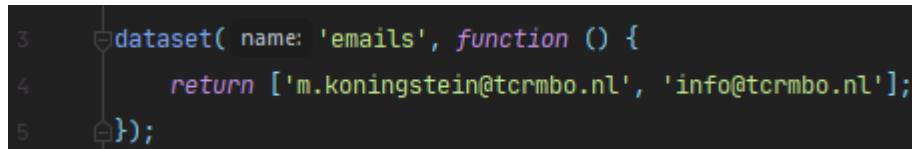


In Emails.php staat nu al een dataset voorbeeld



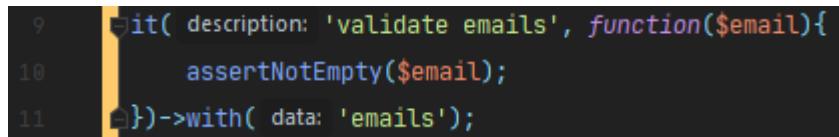
```
<?php
dataset( name: 'emails', function () {
    return ['email A', 'email B'];
});
```

Hierin gaan we onze eigen mailadressen inzetten.



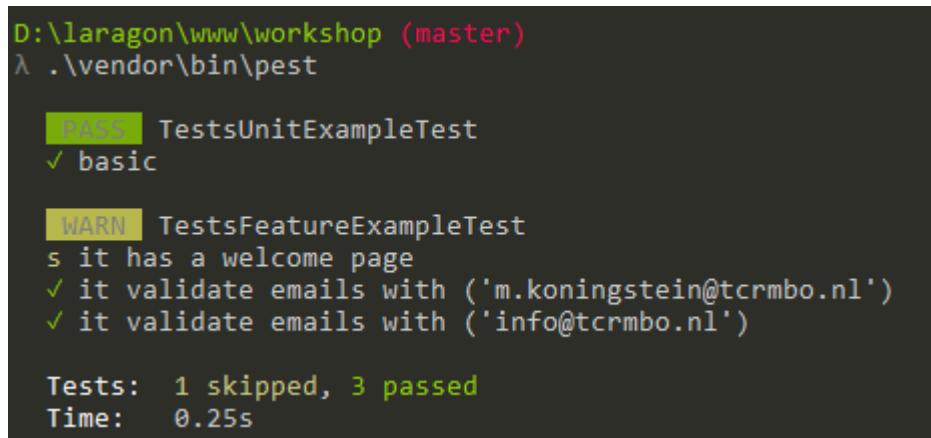
```
dataset( name: 'emails', function () {
    return ['m.koningstein@tcrmbo.nl', 'info@tcrmbo.nl'];
});
```

Nu kunnen we deze dataset in onze test gebruiken



```
it( description: 'validate emails', function($email){
    assertNotEmpty($email);
})->with( data: 'emails');
```

Als we nu is naar onze test gaan kijken, zien we dat de unittest nog steeds goed is. De feature test staat een warning omdat we 1 test skippen. Wel zie je dat onze emailtest voor beide adressen is geslaagd. Nog al logisch, want we testen alleen of die niet leeg is.



```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest

  PASS  Tests\Unit\ExampleTest
    ✓ basic

  WARN  Tests\Feature\ExampleTest
    s it has a welcome page
    ✓ it validate emails with ('m.koningstein@tcrmbo.nl')
    ✓ it validate emails with ('info@tcrmbo.nl')

Tests:  1 skipped, 3 passed
Time:  0.25s
```

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:

```
 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=
```

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.

```
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="mysql"/>
25     <server name="DB_DATABASE" value="testworkshop"/>
26     <server name="MAIL_MAILER" value="array"/>
27     <server name="QUEUE_CONNECTION" value="sync"/>
28     <server name="SESSION_DRIVER" value="array"/>
29     <server name="TELESCOPE_ENABLED" value="false"/>
30 </?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 testworkshop 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 'Databases' section of the phpMyAdmin interface. At the top, there are tabs for 'Databases', 'SQL', 'Status', 'Gebruikersaccounts', and 'Exporteren'. Below the tabs, the word 'Databases' is displayed in large letters. A button labeled 'Database aanmaken' with a question mark icon is visible. Below this, there is a text input field containing 'testworkshop', a dropdown menu set to 'latin1_swedish_ci', and a button labeled 'Aanmaken'.

In mysql hebben we dan 2 databases, namelijk workshop en testworkshop. 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. 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. Je ziet ook bij regel 3 dat binnen testing deze class zit.

```
use Illuminate\Foundation\Testing\RefreshDatabase;
uses( ...classAndTraits: Tests\TestCase::class, RefreshDatabase::class)->in( ...targets: 'Feature');
```

Bij de featuretest gaan we een nieuwe functie erbij maken waarbij je bijvoorbeeld een user in een database zet.

```
it( description: 'has users', function(){
    factory( class: App\User::class)->create();
    $this->assertDatabaseHas('users', ['id' => 1]);
});
```

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.

Als we de test nu gaan uitvoeren zal je merken dat de test goed verloopt

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest

  PASS  Tests\Unit\ExampleTest
    ✓ basic

  WARN  Tests\Feature\ExampleTest
    s it has a welcome page
    ✓ it validate emails with ('m.koningstein@tcrmbo.nl')
    ✓ it validate emails with ('info@tcrmbo.nl')
    ✓ it has users

Tests: 1 skipped, 4 passed
Time: 0.83s
```

Wat misschien wel een beetje vaag is, is als je naar de testworkshop database gaat kijken, we wel alle tabellen zien, maar alles is leeg (behalve de migrations tabel)

| Tabel | Actie | Rijen |
|-----------------------|---|-------|
| categories | ★ Verkennen Structuur Zoeken Invoegen Legen Verwijderen 0 | 0 |
| failed_jobs | ★ Verkennen Structuur Zoeken Invoegen Legen Verwijderen 0 | 0 |
| migrations | ★ Verkennen Structuur Zoeken Invoegen Legen Verwijderen 7 | 7 |
| model_has_permissions | ★ Verkennen Structuur Zoeken Invoegen Legen Verwijderen 0 | 0 |
| model_has_roles | ★ Verkennen Structuur Zoeken Invoegen Legen Verwijderen 0 | 0 |
| password_resets | ★ Verkennen Structuur Zoeken Invoegen Legen Verwijderen 0 | 0 |
| permissions | ★ Verkennen Structuur Zoeken Invoegen Legen Verwijderen 0 | 0 |
| prices | ★ Verkennen Structuur Zoeken Invoegen Legen Verwijderen 0 | 0 |
| products | ★ Verkennen Structuur Zoeken Invoegen Legen Verwijderen 0 | 0 |
| roles | ★ Verkennen Structuur Zoeken Invoegen Legen Verwijderen 0 | 0 |
| role_has_permissions | ★ Verkennen Structuur Zoeken Invoegen Legen Verwijderen 0 | 0 |
| users | ★ Verkennen Structuur Zoeken Invoegen Legen Verwijderen 0 | 0 |
| 12 tabellen | Som | 7 |

Dit komt, omdat na de test alles weer leeggehaald wordt. Tijdens de test moet er echt een gebruiker zijn geweest anders was de test er niet doorheen gekomen.

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.

```
12     it( description: 'has users', function(){
13         factory( class: App\User::class)->create();
14         $this->assertDatabaseHas('users', ['id' => 1]);
15     });
16
17     it( description: 'has users', function(){
18         factory( class: App\User::class)->create();
19         $this->assertDatabaseHas('users', ['id' => 1]);
20     });

```

Als we dat de test uitvoeren krijgen we ook netjes een melding

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest

Pest\Exceptions\TestAlreadyExist

A test with the description `it has users` already exist in the filename
`D:\laragon\www\workshop\tests\Feature\ExampleTest.php` .
```

Hierdoor gaan we nu eerst de naamgeving van de test veranderen. Natuurlijk kunnen we nu ook voor user met id = 2 testen.

```
12     it( description: 'has users', function(){
13         factory( class: App\User::class)->create();
14         $this->assertDatabaseHas('users', ['id' => 1]);
15     });
16
17     it( description: 'has users 2', function(){
18         factory( class: App\User::class)->create();
19         $this->assertDatabaseHas('users', ['id' => 2]);
20     });

```

Als we nu de test uitvoeren zie je dat nu alles is zoals we het verwachten.

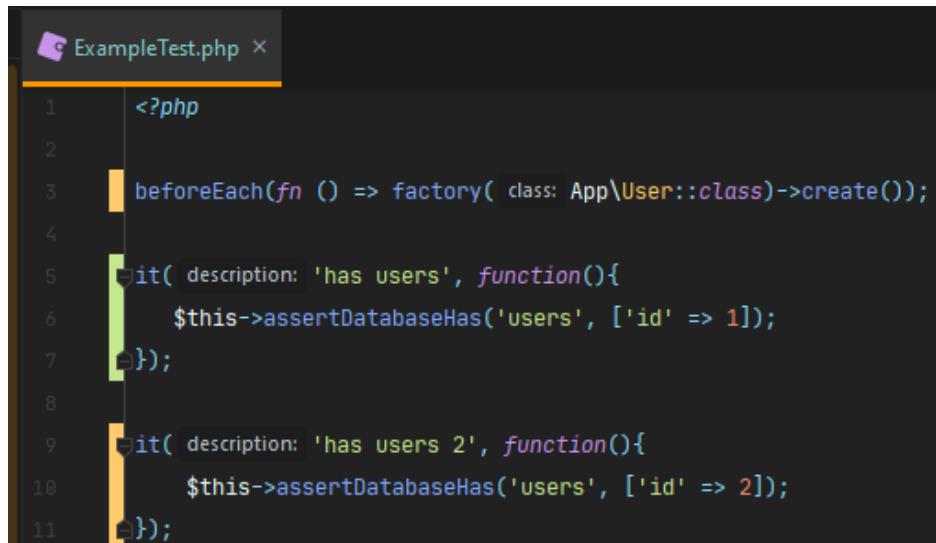
```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest

  PASS | TestsUnitExampleTest
  ✓ basic

  WARN | TestsFeatureExampleTest
  s it has a welcome page
  ✓ it validate emails with ('m.koningstein@tcrmbo.nl')
  ✓ it validate emails with ('info@tcrmbo.nl')
  ✓ it has users
  ✓ it has users 2

Tests: 1 skipped, 5 passed
Time: 0.92s
```

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. De andere tests heb ik even weggehaald om het niet al te moeilijk nu te maken.



```
<?php

beforeEach(fn () => factory( class: App\User::class)->create());

it( description: 'has users', function(){
    $this->assertDatabaseHas('users', ['id' => 1]);
});

it( description: 'has users 2', function(){
    $this->assertDatabaseHas('users', ['id' => 2]);
});
```

Het resultaat is nu goed. We krijgen netjes de users die we verwachten.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest

  PASS | TestsUnitExampleTest
  ✓ basic

  PASS | TestsFeatureExampleTest
  ✓ it has users
  ✓ it has users 2

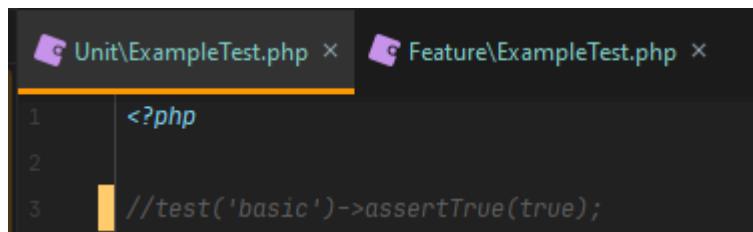
Tests: 3 passed
Time: 0.81s
```

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.

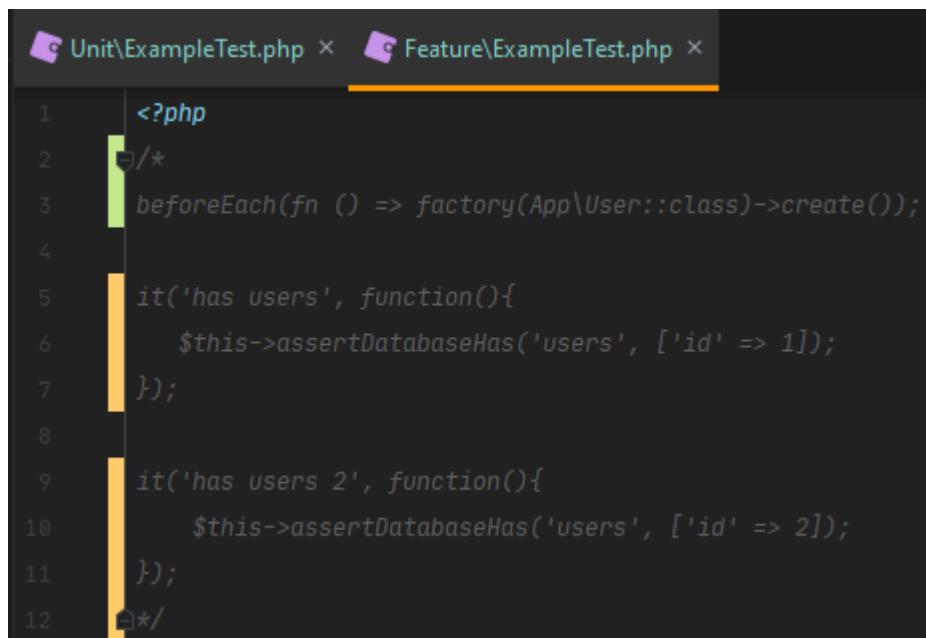
Product Tests

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 naar kijken. Voordeel van products is ook dat we het nu nog zo hebben geprogrammeerd dat je niet ingelogd hoeft te zijn, maar dat je eigenlijk wel permissie moet hebben om bepaalde functionaliteit uit te voeren.

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]);
});
*/
```

Product Index

Als we naar products gaan kijken, gaan we gewoon aan de slag met de 1^e methode waar we terecht zouden komen. We maken netjes een test hiervoor aan. Dit hoeft nu niet met php artisan, want we gebruiken Pest. Hierdoor kunnen we gewoon met een leeg php document beginnen voor de test. We maken dus in de map Feature de test *ProductIndexTest.php* aan.

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.

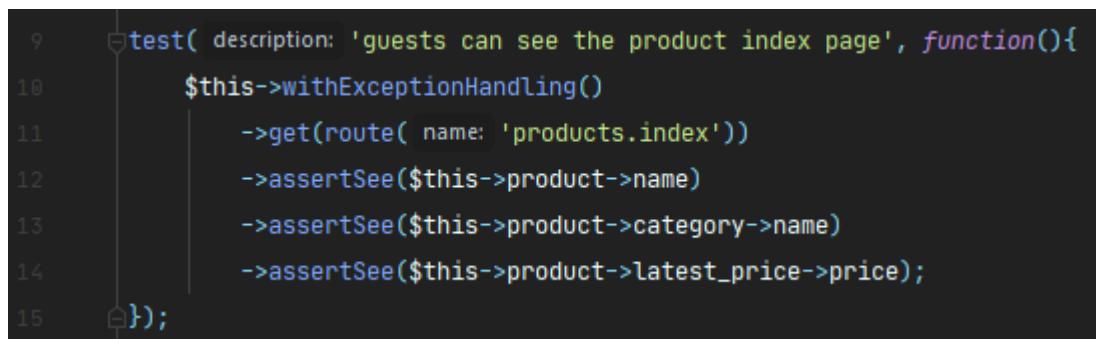


```
ProductIndexTest.php
1 <?php
2
3 beforeEach(function () {
4     $this->category = factory( 'App\Category'::class)->create();
5     $this->product = factory( 'App\Product'::class)->create();
6     $this->price = factory( 'App\Price'::class)->create();
7 });

});
```

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.

Voor nu, hebben we het gemaakt dat iedereen de product index kan zien. Dit is dus ook de description van de test. We werken met named routes in het project, dus deze kunnen we ook gewoon in onze testen gebruiken.



```
test( description: 'guests can see the product index page', function(){
    $this->withExceptionHandling()
        ->get(route( name: 'products.index'))
        ->assertSee($this->product->name)
        ->assertSee($this->product->category->name)
        ->assertSee($this->product->latest_price->price);
});
```

Met assertSee() kan je kijken of je een bepaalde tekst op de pagina kan zien als gebruiker. We weten dat we standaard de productnaam, categorienaam en huidige prijs zien in het overzicht. Dit testen we dus nu ook.

Extra Testing Configuratie

Als we nu het zo willen maken dat alleen degene met de permissie de product index kunnen zien, moeten we goed gaan kijken wat nodig is.

Waar we naar moeten kijken, is het werken met migrations. Laravel vertelt het volgende hierover:

Database Migrations

When your test requires migrations, like the authentication example above, you should never use the `RefreshDatabase` trait. The `RefreshDatabase` trait leverages database transactions which will not be applicable across HTTP requests. Instead, use the `DatabaseMigrations` trait:

We zullen dus dit gaan verwerken in de standaard voor onze test. Als we gaan kijken naar onze `Pest.php` komt deze er dan zo uit te zien.

```
1  <?php
2
3  //use Illuminate\Foundation\Testing\RefreshDatabase;
4  use Illuminate\Foundation\Testing\DatabaseMigrations;
5
6  uses( ...classAndTraits: Tests\TestCase::class, DatabaseMigrations::class)
7      ->in( ...targets: 'Feature');
```

We gebruiken nu niet meer de `RefreshDatabase`, maar de `DatabaseMigrations`.

Verder moeten we de permissies registeren voordat alle testen gaan plaatsvinden. Dit kunnen we doen in de `beforeEach()` die we al in `ProductIndexTest` hebben staan.

```
3  beforeEach(function () {
4      $this->seed('RoleAndPermissionSeeder');
5      $this->seed('TestingUserSeeder');
6      $this->category = factory( class: App\Category::class)->create();
7      $this->product = factory( class: App\Product::class)->create();
8      $this->price = factory( class: App\Price::class)->create();
9  });
```

Voordat we verder gaan kunnen we kijken of we niks kapot hebben gemaakt met deze opties. We voeren gewoon de test even uit die we nu hebben.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest

  PASS TestsFeatureProductIndexTest
    ✓ guests can see the product index page

  Tests: 1 passed
  Time: 0.79s
```

Zoals je ziet werkt de test nog gewoon

Wel een verschil die we in de test database zien, is dat we alleen nog standaard de migration tabel zien.

| | Tabel | Actie |
|--------------------------|------------|-------|
| <input type="checkbox"/> | migrations | ★ ┌ ┘ |
| | 1 tabel | Som |

Dit betekent niet dat de tabellen niet hebben bestaan, want anders was de test niet gelukt. Alleen de migrations worden na de test gewoon weer teruggedraaid. Hierdoor zie je gewoon geen tabellen meer na een test.

De rollen en permissies hebben we nu geregeld. Het zou handig zijn als we de 3 users met een bepaalde role hebben zodat we alles goed uit kunnen testen. Voor de tests maken we dan een TestingUserSeeder aan.

In de TestingUserSeeder zet ik de 3 users die we standaard aanmaken. (deze kan ik ook weghalen bij UserSeeder, als we deze TestingUserSeeder ook aanroepen in de DatabaseSeeder).

```
1  <?php
2
3  use Illuminate\Database\Seeder;
4  use App\User;
5
6  class TestingUserSeeder extends Seeder
7  {
8      /** Run the database seeds. ...*/
9      public function run()
10     {
11         factory( User::class,  amount: 1)->create([
12             'name'=>'Customer',
13             'email'=>'customer@test.com',
14             'password'=> bcrypt( value: 'test1234'))->each(function (User $user) {
15                 $user->assignRole( ...roles: 'customer');
16             });
17
18         factory( User::class,  amount: 1)->create([
19             'name'=>'Sales',
20             'email'=>'sales@test.com',
21             'password'=> bcrypt( value: 'test1234'))->each(function (User $user) {
22                 $user->assignRole( ...roles: 'sales');
23             });
24
25         factory( User::class,  amount: 1)->create([
26             'name'=>'Admin',
27             'email'=>'admin@test.com',
28             'password'=> bcrypt( value: 'test1234'))->each(function (User $user) {
29                 $user->assignRole( ...roles: 'admin');
30             });
31
32     }
33
34 }
```

We weten nu dat de permissions tot zo ver zijn geregeld en dat we users hebben die een bepaalde permissie hebben. Wel is het zo dat we voor Product nog niks in de RoleAndPermissionSeeder hebben staan. Deze vullen we dan ook aan met create, edit en delete permissie.

```
7  class RoleAndPermissionSeeder extends Seeder
8  {
9      /** Run the database seeds. ...*/
10     public function run()
11     {
12
13         app()['cache']->forget( key: 'spatie.permission.cache');
14
15
16         Permission::create(['name' => 'create category']);
17         Permission::create(['name' => 'edit category']);
18         Permission::create(['name' => 'delete category']);
19
20
21         Permission::create(['name' => 'create product']);
22         Permission::create(['name' => 'edit product']);
23         Permission::create(['name' => 'delete product']);
24
25
26         $role = Role::create(['name' => 'customer']);
27
28
29         $role = Role::create(['name' => 'sales']);
30         $role->givePermissionTo('create category', 'edit category');
31         $role->givePermissionTo('create product', 'edit product');
32
33
34         $role = Role::create(['name' => 'admin']);
35         $role->givePermissionTo(Permission::all());
36     }
37 }
```

We hebben nu 3 gebruikers en hun rollen. We zorgen nog wel even dat de role sales de permissie krijgt om een product aan te maken en te wijzigen.

Nu kunnen we terug naar de product test om te kijken wat er verder nog nodig is.

Verder met Product Index Test

Voor de product index test hadden we al de index getest zonder beveiliging. Nu zullen we eerst de beveiliging erop moeten zetten, om daarna dan ook de testen verder aan te passen en werkelijk te kunnen testen.

Eerst kunnen we in de routes de beveiliging maken, dat je de rol sales of admin moet hebben om al bij de routes te komen voor products.

```
25     Route::group(['middleware' => ['role:sales|admin']], function () {
26         // Categories CRUD
27         Route::get('categories/{category}/delete', [action: 'CategoryController@delete'])
28             ->name('categories.delete');
29         Route::resource('name: /categories', controller: 'CategoryController');
30
31         // Product CRUD
32         Route::get('products/{product}/delete', [action: 'ProductController@delete'])
33             ->name('products.delete');
34         Route::resource('name: /products', controller: 'ProductController');
35     });

```

Daarna kunnen we de koppeling maken tussen de permissions en de methodes in de product controller.

```
13 class ProductController extends Controller
14 {
15     /** Set permissions on methods ...*/
16     public function __construct()
17     {
18         $this->middleware('auth');
19         $this->middleware('permission:create product', ['only' => ['create', 'store']]);
20         $this->middleware('permission:edit product', ['only' => ['edit', 'update']]);
21         $this->middleware('permission:delete product', ['only' => ['delete', 'destroy']]);
22     }
23 }
24 }
```

Als we nu zouden testen, gaat de test niet goed. Dit klopt, want je krijgt namelijk een redirect naar de login pagina.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest

FAIL TestsFeatureProductIndexTest
• guests can see the product index page

---
• TestsFeatureProductIndexTest > guests can see the product index page
Failed asserting that '<!DOCTYPE html>\n
```

We passen dan ook onze test aan zodat we met de admin gaan inloggen voordat we naar de pagina gaan kijken.
De ProductIndexTest ziet er dan zo uit.

```
3     beforeEach(function () {
4         $this->seed('RoleAndPermissionSeeder');
5         $this->seed('TestingUserSeeder');
6         $this->category = factory( class: App\Category::class)->create();
7         $this->product = factory( class: App\Product::class)->create();
8         $this->price = factory( class: App\Price::class)->create();
9     });
10
11    test( description: 'admin can see the product index page', function(){
12        $admin = App\User::find(3);
13
14        actingAs($admin)
15            ->get(route( name: 'products.index'))
16            ->assertSee($this->product->name)
17            ->assertSee($this->product->category->name)
18            ->assertSee($this->product->latest_price->price);
19    });

```

De test beschrijving is nu, dat we het als admin bekijken. We weten dat de roles, permissions en users zijn ingeladen, dus kunnen ook de admin ophalen. Omdat dit de 3^e user in onze seed is, zal deze id = 3 krijgen.

Daarna geven we aan in onze test, dat je moet acteren als zijn deze user. Nu we als de admin naar de pagina gaan kijken zou het moeten zijn dat de test goed gaat. Als we de test uitvoeren zien we ook netjes dat de test is gelukt.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest

[PASS] Tests\Feature\ProductIndexTest
✓ admin can see the product index page

Tests: 1 passed
Time: 0.87s
```

Als we als admin het nu kunnen testen, zouden we ook als sales en customer kunnen testen. 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.

```
21     test( description: 'sales can see the product index page', function(){
22         $sales = App\User::find(2);
23
24         actingAs($sales)
25             ->get(route( name: 'products.index'))
26             ->assertSee($this->product->name)
27             ->assertSee($this->product->category->name)
28             ->assertSee($this->product->latest_price->price);
29     });

```

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.

Fixed navbar Home Link Disabled Product Test Customer ▾

Error

User does not have the right roles.

Bij het testen kunnen we dus zeggen, dat je er niet bij mag komen als customer.

```
31     test( description: 'customer can not see the product index page', function(){
32         $customer = App\User::find(1);
33
34         actingAs($customer)
35             ->get(route( name: 'products.index'))
36             ->assertForbidden();
37     });

```

Als we de test nu uitproberen krijgen we ook netjes terug dat de test is geslaagd.

```
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:   2.01s
```

De laatste test is dat voor iemand die niet is ingelogd. Als je niet ingelogd bent, wordt je doorverwezen naar de login pagina.

```
39     test( description: 'guest can not see the product index page', function(){
40         $this->get(route( name: 'products.index'))
41             ->assertRedirect(route( name: 'login'));
42     );
});
```

Let even op dat je niet kan beginnen met get(), dus hebben we nu wel \$this nodig.

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)

| D:\laragon\www\workshop (master) | | | | | |
|----------------------------------|-----------|------------------------------|--------------------|---|--|
| \ php artisan route:list | | | | | |
| Domain | Method | URI | Name | Action | Middleware |
| | GET HEAD | / | start | App\Http\Controllers\HomeController@index | web,auth |
| | GET HEAD | api/user | Closure | App\Http\Controllers\Auth\LoginController@getLogin | api,auth:api |
| | POST | categories | categories.store | App\Http\Controllers\CategoryController@store | web,role:sales admin,auth,permission:create category |
| | GET HEAD | categories | categories.index | App\Http\Controllers\CategoryController@index | web,role:sales admin,auth |
| | GET HEAD | categories/create | categories.create | App\Http\Controllers\CategoryController@create | web,role:sales admin,auth,permission:create category |
| | DELETE | categories/{category} | categories.destroy | App\Http\Controllers\CategoryController@destroy | web,role:sales admin,auth,permission:delete category |
| | PUT PATCH | categories/{category} | categories.update | App\Http\Controllers\CategoryController@update | web,role:sales admin,auth,permission:edit category |
| | GET HEAD | categories/{category} | categories.show | App\Http\Controllers\CategoryController@show | web,role:sales admin,auth |
| | GET HEAD | categories/{category}/delete | categories.delete | App\Http\Controllers\CategoryController@delete | web,role:sales admin,auth,permission:delete category |
| | GET HEAD | categories/{category}/edit | categories.edit | App\Http\Controllers\CategoryController@edit | web,role:sales admin,auth,permission:edit category |
| | GET HEAD | home | home | App\Http\Controllers\HomeController@index | web,auth |
| | GET HEAD | login | login | App\Http\Controllers\Auth\LoginController@showLoginForm | web,guest |
| | POST | login | login | App\Http\Controllers\Auth\LoginController@login | web,guest |
| | POST | logout | logout | App\Http\Controllers\Auth\LoginController@logout | web |
| | POST | password/confirm | password.confirm | App\Http\Controllers\Auth\ConfirmPasswordController@confirm | web,auth |
| | GET HEAD | password/confirm | password_email | App\Http\Controllers\Auth\ForgotPasswordController@sendResetLinkEmail | web |
| | POST | password/reset | password.update | App\Http\Controllers\Auth\ResetPasswordController@reset | web |
| | GET HEAD | password/reset | password.request | App\Http\Controllers\Auth\ResetPasswordController@showLinkRequestForm | web |
| | GET HEAD | password/reset/{token} | password.reset | App\Http\Controllers\Auth\ResetPasswordController@showResetForm | web |
| | GET HEAD | products | products.index | App\Http\Controllers\ProductController@index | web,role:sales admin,auth |
| | POST | products | products.store | App\Http\Controllers\ProductController@store | web,role:sales admin,auth,permission:create product |
| | GET HEAD | products/create | products.create | App\Http\Controllers\ProductController@create | web,role:sales admin,auth,permission:create product |
| | GET HEAD | products/{product} | products.show | App\Http\Controllers\ProductController@show | web,role:sales admin,auth |
| | PUT PATCH | products/{product} | products.update | App\Http\Controllers\ProductController@update | web,role:sales admin,auth,permission:edit product |
| | DELETE | products/{product} | products.destroy | App\Http\Controllers\ProductController@destroy | web,role:sales admin,auth,permission:delete product |
| | GET HEAD | products/{product}/delete | products.delete | App\Http\Controllers\ProductController@delete | web,role:sales admin,auth,permission:delete product |
| | GET HEAD | products/{product}/edit | products.edit | App\Http\Controllers\ProductController@edit | web,role:sales admin,auth,permission:edit product |
| | GET HEAD | register | register | App\Http\Controllers\Auth\RegisterController@showRegistrationForm | web,guest |
| | POST | register | | App\Http\Controllers\Auth\RegisterController@register | web,guest |

Wat ook handig is, is dat je meteen zien hoe de permissies zijn geregeld met middleware.

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.

Als we straks meer tests gaan hebben wordt het misschien een beetje onoverzichtelijk als we alles in 1 map hebben. Hierdoor kunnen we binnen de map Feature de map Products aanmaken en daarin onze test zetten.

Natuurlijk nog wel even gekeken of onze test het nog steeds doet binnen de map.

```
D:\laragon\www\workshop (master)
\ vendor\bin\pest

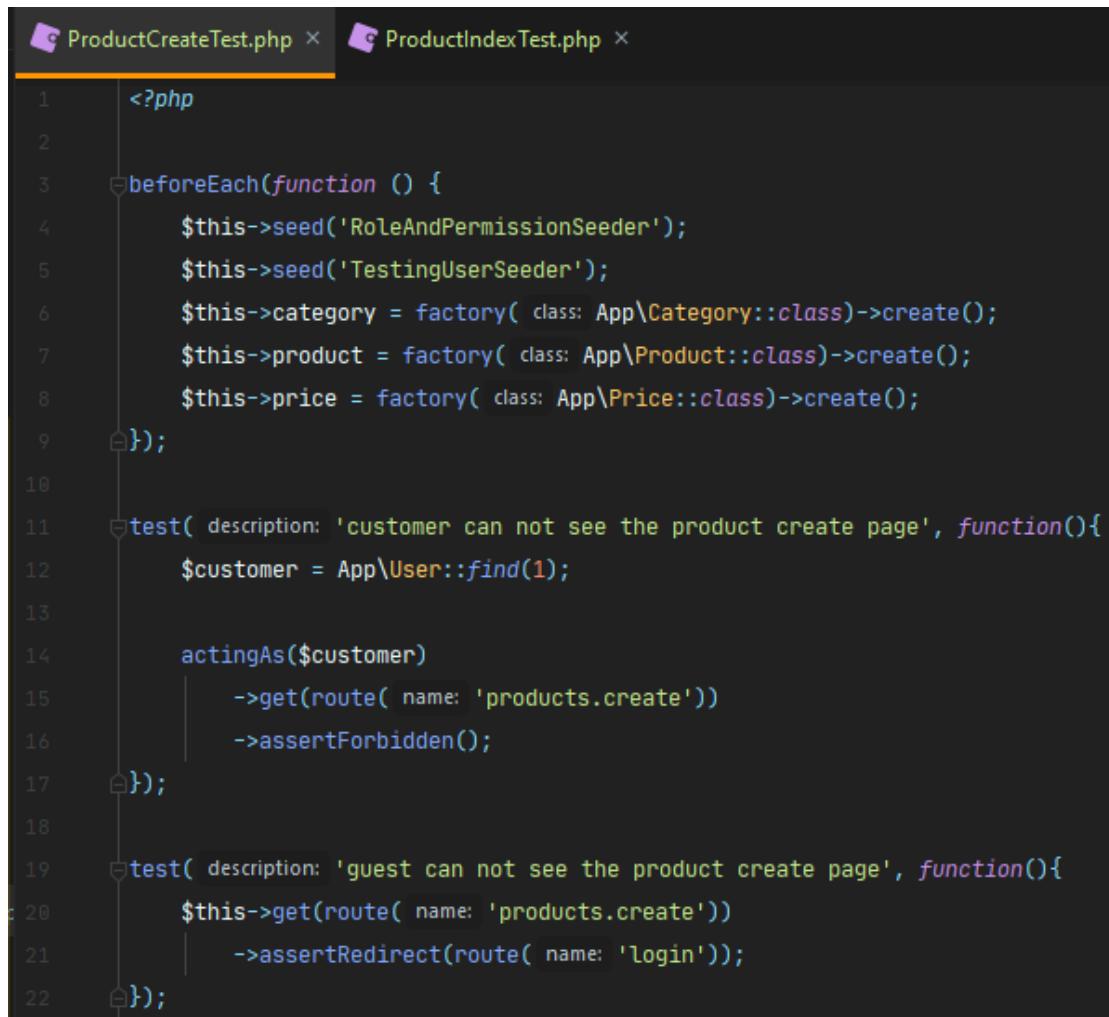
PASS TestsFeatureProductsProductIndexTest
✓ admin can see the product index page
✓ sales can see the product index page
✓ customer can not see the product index page
✓ guest can not see the product index page

Tests: 4 passed
Time: 2.64s
```

Product Create

Eerst maken we het testbestand aan in de map tests/Feature/Products. We noemen het bestand ProductCreateTest. 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.

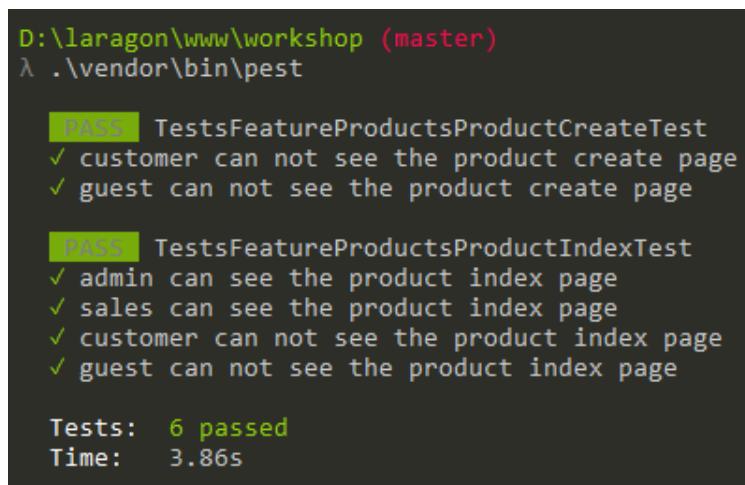
De admin en sales mogen er wel bij. Deze moeten dan een formulier zien om een product in te vullen. Hier zullen we wel nog even moeten nadenken hoe we dit kunnen testen. Eerst dus even de standaard onderdelen in de test zetten en de tests voor guest en customer, want die zijn nu simpel.



The screenshot shows two tabs open in a code editor: 'ProductCreateTest.php' and 'ProductIndexTest.php'. The 'ProductCreateTest.php' tab is active and contains the following PHP code:

```
1 <?php
2
3 beforeEach(function () {
4     $this->seed('RoleAndPermissionSeeder');
5     $this->seed('TestingUserSeeder');
6     $this->category = factory( class: App\Category::class)->create();
7     $this->product = factory( class: App\Product::class)->create();
8     $this->price = factory( class: App\Price::class)->create();
9 });
10
11 test( description: 'customer can not see the product create page', function(){
12     $customer = App\User::find(1);
13
14     actingAs($customer)
15         ->get(route( name: 'products.create'))
16         ->assertForbidden();
17 });
18
19 test( description: 'guest can not see the product create page', function(){
20     $this->get(route( name: 'products.create'))
21         ->assertRedirect(route( name: 'login'));
22 });
```

Als we gaan testen krijgen we een goed resultaat:



```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest

PASS TestsFeatureProductsProductCreateTest
✓ customer can not see the product create page
✓ guest can not see the product create page

PASS TestsFeatureProductsProductIndexTest
✓ 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: 6 passed
Time: 3.86s
```

Nu nog de test voor de admin. We halen de admin op en gaan naar de named route products.create

```
24     test('admin can see the product create page', function(){
25         $admin = App\User::find(3);
26
27         actingAs($admin)
28             ->get(route( name: 'products.create'))
29             ->assertViewIs( value: 'admin.products.create');
30     });

```

We kunnen controleren of we de goede view zien. Voorlopig is dit even genoeg. Voor sales is deze test zo goed als hetzelfde. Alleen even de testbeschrijving aanpassen en de sales user ophalen.

```
32     test('sales can see the product create page', function(){
33         $sales = App\User::find(2);
34
35         actingAs($sales)
36             ->get(route( name: 'products.create'))
37             ->assertViewIs( value: 'admin.products.create');
38     });

```

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.

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, maar laat ik nu niet meer zien. Als we gaan kijken naar de store, zullen we ook moeten testen of we erbij kunnen met de POST methode.

```
12  ⌂ test( description: 'customer can not create a product', function () {
13      $customer = App\User::find(1);
14
15      actingAs($customer)
16          ->get(route( name: 'products.create'))
17          ->assertForbidden();
18
19      actingAs($customer)
20          ->post(route( name: 'products.store'))
21          ->assertForbidden();
22  });
23
24  ⌂ test( description: 'guest can not create a product', function () {
25      $this->get(route( name: 'products.create'))
26          ->assertRedirect(route( name: 'login'));
27
28      $this->post(route( name: 'products.store'))
29          ->assertRedirect(route( name: 'login'));
30  });
31
```

Nu dit wordt getest, kunnen we naar degene die het wel mogen zien, de admin en sales. De is wel wat lastiger, want we moeten wel ingelogd zijn als admin, maar daarna moeten we een post doen richting de store methode

```
32  ⌂ test( description: 'admin can create a product', function () {
33      $admin = App\User::find(3);
34      $category = App\Category::find(1);
35
36      actingAs($admin);
37      $product = factory( class: 'App\Product', ['category_id' => $category->id])->create();
38      $response = $this->post(route( name: 'products.store'), $product->toArray());
39
40      $this->get(route( name: 'products.index'))
41          ->assertSee($product->name)
42          ->assertSee($product->description);
43  });
44
```

Zoals je kan zien maken we een product aan met behulp van de factory. Let op dat deze niet zomaar in de database komt, want normaal gebruiken we daar de seed voor. Nu maken we gebruik van de methode post() naar de named route products.store, waar we de product als array meegeven. Op deze manier wordt namelijk hetzelfde gedaan als normaal. Zelf zouden we een formulier invullen en opsturen naar de store methode, wat dan aankomt als array. Standaard is dit in php natuurlijk de \$_POST. In Laravel zien we dit alleen niet meer. In de store methode gebruiken we namelijk de request.

Op het eind van de test checken we de index of ons nieuw aangemaakte product er op staat. Als dit het geval is slaagt de test.

```
D:\laragon\www\workshop (master)
\ .\vendor\bin\pest

[PASS] TestsFeatureProductsProductCreateTest
✓ 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] TestsFeatureProductsProductIndexTest
✓ 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] TestsFeatureProductsProductStoreTest
✓ customer can not create a product
✓ guest can not create a product
✓ admin can create a product

Tests: 11 passed
Time: 7.43s
```

We zien dat de test er netjes doorheen komt. Nu kunnen we natuurlijk ook deze test uitproberen voor de customer. Die mag natuurlijk niks posten, maar toch nog even voor de zekerheid.

```
58     test( description: 'customer can not create a product by posting', function () {
59         $customer = App\User::find(1);
60         $category = App\Category::find(1);
61
62         actingAs($customer);
63         $product = factory( class: 'App\Product', ['category_id' => $category->id])->create();
64         $response = $this->post(route( name: 'products.store'), $product->toArray());
65
66         $this->get(route( name: 'products.index'))
67             ->assertForbidden();
68     });

```

We gebruiken dus nu de customer, doen precies hetzelfde als bij de admin en sales, het enige is nu, dat ik verwacht dat ik een melding krijg dat ik niet de juiste permissies heb.

Als we dit testen, komt deze test er ook goed door.

```
PASS TestsFeatureProductsProductStoreTest
✓ customer can not create a product
✓ guest can not create a product
✓ admin can create a product
✓ sales can create a product
✓ customer can not create a product by posting
```

Dit betekent dus dat de beveiliging goed zijn werk doet.

Wel gaan we eerst even zorgen dat we niet constant alle testen uitvoeren. Dit doen we door middel van een test in een group te plaatsen. Als we even kijken naar de laatste test, zetten we achter de test in welke groep deze zit, met behulp van `->group('ProductStore')`

```
58     test( description: 'customer can not create a product by posting', function () {
59         $customer = App\User::find(1);
60         $category = App\Category::find(1);
61
62         actingAs($customer);
63         $product = factory( class: 'App\Product', ['category_id' => $category->id])->create();
64         $response = $this->post(route( name: 'products.store'), $product->toArray());
65
66         $this->get(route( name: 'products.index'))
67             ->assertForbidden();
68     })->group( ...groups: 'ProductStore');
```

Dit kunnen we voor alle tests natuurlijk doen. Dus dan hebben we de volgende groepen:

- ProductIndex
- ProductCreate
- ProductStore

Let op, dat de beforeEach zelf geen test is. Deze kunnen we niet in een group zetten.

We kunnen nu bijvoorbeeld alleen de test voor de ProductStore doen.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=ProductStore

PASS TestsFeatureProductsProductStoreTest
✓ customer can not create a product
✓ guest can not create a product
✓ admin can create a product
✓ sales can create a product
✓ customer can not create a product by posting

Tests: 5 passed
Time: 3.35s
```

ProductStoreCheck met phunit

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.

Helaas op het moment van schrijven is Pest nog in beta. Hierdoor is nog niet alles mogelijk om met Pest te testen. Gelukkig kan je phunit en pest bij elkaar gebruiken. We gaan dus nu een aantal tests met phunit schrijven. Hierdoor leer je ook beide manieren, wat natuurlijk beter is.

Omdat we nu met phunit 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:\laragon\www\workshop (master)
λ php artisan make:test Products/ProductStoreCheckTest
Test created successfully.
```

Met Products/ zorgen we ervoor dat de test wel gewoon netjes in de map Products bij de andere tests komt.

De test ziet er nu zo uit

```
1  <?php
2
3  namespace Tests\Feature\Products;
4
5  use Illuminate\Foundation\Testing\RefreshDatabase;
6  use Illuminate\Foundation\Testing\WithFaker;
7  use Tests\TestCase;
8
9  ► class ProductStoreCheckTest extends TestCase
10 {
11     /**
12      * A basic feature test example.
13      *
14      * @return void
15     */
16    ► public function testExample()
17    {
18        $response = $this->get( uri: '/');
19
20        $response->assertStatus( status: 200);
21    }
22}
```

We gaan dit aanpassen, omdat we bijvoorbeeld niet met RefreshDatabase werken etc.

De basis van de test in phpunit is dan dit.

```
1 <?php
2
3 namespace Tests\Feature\Products;
4
5 use Illuminate\Foundation\Testing\DatabaseMigrations;
6 use Illuminate\Foundation\Testing\WithFaker;
7 use Tests\TestCase;
8 use App\Category;
9 use App\Product;
10 use App\Price;
11
12 ► class ProductStoreCheckTest extends TestCase
13 {
14     use DatabaseMigrations;
15
16     ► public function setUp(): void
17     {
18         // first include all the normal setUp operations
19         parent::setUp();
20
21         $this->seed( class: 'RoleAndPermissionSeeder' );
22         $this->seed( class: 'TestingUserSeeder' );
23         $this->category = factory( class: Category::class )->create();
24         $this->product = factory( class: Product::class )->create();
25         $this->price = factory( class: Price::class )->create();
26     }
27
28     /** A basic feature test example. ...*/
29     ► public function testExample()
30     {
31         $response = $this->get( uri: '/' );
32         $response->assertStatus( status: 302 );
33     }
34 }
```

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.

De test staat gewoon bij de uitvoer.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest

  PASS  Tests\Feature\Products\ProductStoreCheckTest
    ✓ example

  PASS  TestsFeatureProductsProductCreateTest
    ✓ 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  TestsFeatureProductsProductIndexTest
    ✓ 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  TestsFeatureProductsProductStoreTest
    ✓ customer can not create a product
    ✓ guest can not create a product
    ✓ admin can create a product
    ✓ sales can create a product
    ✓ customer can not create a product by posting
    ✓ admin can create a product without name

Tests: 15 passed
Time: 9.47s
```

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 = [])
{
    $this->withExceptionHandling();
    $product = make( class: 'App\Product', [
        $overrides
    ]);
    return $this->post(route( name: 'products.store'), $product->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 */` ervoor.

```
/** @test */
function a_product_requires_a_name()
```

Nu zie je in PostProduct dat ik de functie make() gebruik. Dit is een zelfgemaakte afkorting om de code wat cleaner te krijgen. Om ervoor te zorgen dat je dit steeds kan gebruiken gaan we dit op de volgende manier doen.

Als eerst gaan we de composer.json aanpassen. Wat we hiermee zetten is dat we binnen de map tests een map utilities maken, waarin een functions.php komt. Let op dat we het in autoload-dev zetten. Dus als de site online staat zal het niet gebruikt gaan worden.

```
48     "autoload-dev": {
49         "psr-4": {
50             "Tests\\": "tests/"
51         },
52         "files": ["tests/utilities/functions.php"]
53     },
54     "minimum-stability": "dev",
```

De functions.php zal er zo uit komen te zien.

```
1 <?php
2
3 function create($class, $attributes = [], $times = null)
4 {
5     return factory($class, $times)->create($attributes);
6 }
7
8 function make($class, $attributes = [], $times = null)
9 {
10    return factory($class, $times)->make($attributes);
11 }
```

Er zijn nu dus 2 functies, 1 voor create en 1 voor make. Wat we normaal moeten doen om bijvoorbeeld 5 gebruikers aan te maken is dit.

```
factory( class: App\User, amount: 5)->create(['name' => 'Piet']);
```

Dus als we het zo meteen gaan schrijven ziet er iets cleaner uit

```
make( class: App\User, ['name' => 'Piet'], times: 5);
```

Als we dan nog even terug kijken naar onze functie

```
public function postProduct($overrides = [])
{
    $this->withExceptionHandling();
    $product = make( class: 'App\Product', [
        $overrides
    ]);
    return $this->post(route( name: 'products.store'), $product->toArray());
}
```

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.

```
29      /** @test */
30      function a_product_requires_a_name()
31      {
32          $admin = \App\User::find(3);
33          $this->actingAs($admin);
34          $this->postProduct(['name' => null])
35              ->assertStatus( status: 422);
36      }

```

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.

Volgens de http status zou er dan 422 uit moeten komen, want we komen als het goed is niet door de validatie heen.

Helaas gaat het niet goed.

```
D:\laragon\www\workshop (master)
└ .\vendor\bin\pest

FAIL  Tests\Feature\ProductStoreCheckTest
• a product requires a name

PASS  Tests\Feature\Products\CreateTest
✓ 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\IndexTest
✓ 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\StoreTest
✓ customer can not create a product
✓ guest can not create a product
✓ admin can create a product
✓ sales can create a product
✓ customer can not create a product by posting
✓ admin can create a product without name

---
• Tests\Feature\ProductStoreCheckTest > a product requires a name
Expected status code 422 but received 302.
Failed asserting that 422 is identical to 302.

at D:\laragon\www\workshop\vendor\phpunit\phpunit\src\Framework\Constraint\Constraint.php:119
115|         if (!empty($description)) {
116|             $failureDescription = $description . "\n" . $failureDescription;
117|
118|
→ 119|             throw new ExpectationFailedException(
120|                 $failureDescription,
121|                 $comparisonFailure
122|
123|             );
1

1  D:\laragon\www\workshop\vendor\phpunit\phpunit\src\Framework\Constraint\IsIdentical.php:93
PHPUnit\Framework\Constraint\Constraint::fail("Expected status code 422 but received 302.")

2  D:\laragon\www\workshop\vendor\phpunit\phpunit\src\Framework\Assert.php:2228
PHPUnit\Framework\Constraint\IsIdentical::evaluate("Expected status code 422 but received 302.")

Tests: 1 failed, 14 passed
Time: 9.48s
```

We krijgen een 302 error, terwijl we een 422 hadden verwacht. En dit is vaag, want de test zou goed moeten zijn. Helaas is dit nog niet het geval, maar we gaan onderzoeken waarom dit is.

Onderzoek naar status fout

Binnen Laravel heb je een hele handige debugbar, namelijk de laravel debugbar.

(<https://github.com/barryvdh/laravel-debugbar>)

The screenshot shows the GitHub repository page for 'barryvdh / laravel-debugbar'. The repository has 355 issues and 17 pull requests. It includes tabs for Code, Issues (355), Pull requests (17), ZenHub, Actions, and Security. The repository description is 'Laravel Debugbar (Integrates PHP Debug Bar)'. There are buttons for Sponsor and Used by, and a count of 35k users.

Deze 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:\laragon\www\workshop (master)
λ composer require barryvdh/laravel-debugbar --dev
Using version ^3.3 for barryvdh/laravel-debugbar
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
```

Daarna even een check op dit:

The Debugbar will be enabled when APP_DEBUG is true .

Binnen de .env file kunnen we dit vinden:

```
APP_DEBUG=true
```

Deze optie staat dus voor ons al aan.

Binnen de config/app.php gaan we dan zorgen dat de ServiceProvider gebruikt kan worden.

```
165      */
166      * Package Service Providers...
167      */
168      Spatie\Permission\PermissionServiceProvider::class,
169      Barryvdh\Debugbar\ServiceProvider::class,
```

En wat meer naar beneden staat de Facade

```
230     'View' => Illuminate\Support\Facades\View::class,
231     'Debugbar' => Barryvdh\Debugbar\Facade::class,
232 ],
```

De laatste stap, eigenlijk nu niet nodig, maar misschien handig voor later. We publishen de config voor de debugbar.

```
D:\laragon\www\workshop (master)
λ 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.

The screenshot shows a browser window with a dark theme. At the top, there's a navigation bar with links: 'Fixed navbar', 'Home', 'Link', 'Disabled', 'Product Test', 'Login', and 'Register'. Below the navigation bar is a main content area with a title 'Login' and a form for entering an 'E-Mail Address' and 'Password'. There's also a 'Remember Me' checkbox and two buttons: 'Login' and 'Forgot Your Password?'. At the bottom of the page, there's a red horizontal bar containing various icons, likely from the Laravel Debugbar. To the right of this bar, the text 'GET login' is followed by performance metrics: '18MB', '134ms', and '7.4.5'.

Hier gaan we zo gebruik van maken. We gaan proberen handmatig de test uit te voeren en kijken wat er werkelijk gebeurd.

Om de test na te doen, loggen we in als admin en gaan we naar de products create pagina.

The screenshot shows a web application interface for creating a product. At the top, there's a header with a logo, a search bar, and several icons. Below the header is a navigation bar with links like 'Home', 'Link', 'Disabled', 'Product Test', and 'Admin'. The main content area has a title 'Products' and a sub-section 'Create'. It contains four input fields: 'Productname' (empty), 'Description' (containing 'Dit is een voorbeeld test'), 'Price' (containing '6.66'), and 'Category' (containing 'Spencer Torp IV'). A '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 same 'Products' create form after a submission attempt. A prominent red error message box is displayed, containing a single bullet point: '• The name field is required.' This indicates that the validation for the required product name field has been triggered.

Maar waarom krijgen we dan bij de phpunit test geen 422??? Laten we verder kijken met de debugbar. Dit omdat een 302 een redirect is, dus misschien gaat in onze test iets verkeerd met bijvoorbeeld de ingelogde user.

Als we op het mapje rechts onder klikken krijgen we dit te zien

| DATE | METHOD | URL | IP | FILTER DATA |
|---------------------|--------|------------------|-----|-------------|
| 2020-06-03 17:50:06 | POST | /products | ::1 | Show URL |
| 2020-06-03 17:50:06 | GET | /products/create | ::1 | Show URL |

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)

Products

The name field is required.

POST products 20MB 234ms 7.4.5 #2 products (opened) (17:50) [File](#) [View](#)

The given data was invalid.
D:\Laragon\www\workshop\vendor\Laravel\framework\src\Illuminate\Foundation\Http\FormRequest.php#130

```
/*
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.

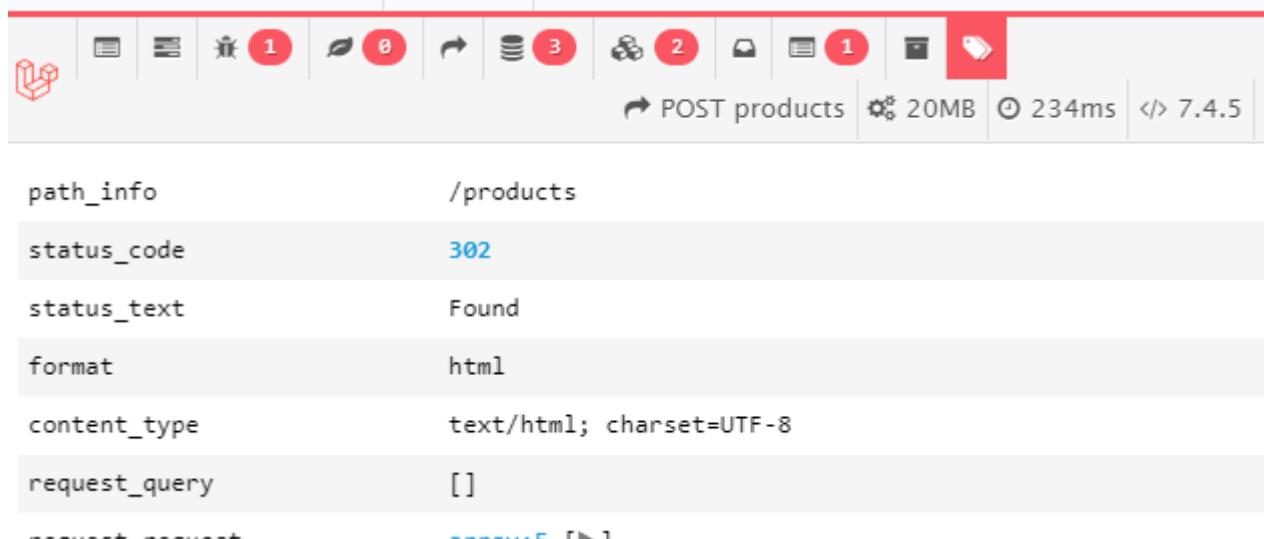
Als we naar de Gate gaan, zie je hoe het zit met permissies

POST products 20MB

```
★ 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.

Dit zou dus ook zo gebeuren met onze phpunit test. Maar nu gaan we kijken bij Request.



The screenshot shows the Network tab of a browser developer tools interface. At the top, there are several icons representing different network operations. Below the icons, the details of a specific request are listed:

| | |
|---------------|--------------------------|
| path_info | /products |
| status_code | 302 |
| status_text | Found |
| format | html |
| content_type | text/html; charset=UTF-8 |
| request_query | [] |

At the bottom of the Network tab, there are buttons for 'cancel' and 'refresh'.

Daar zien we opeens de status_code 302. Dat is onze foutmelding !

Als je verder online gaat zoeken wat het probleem is, zal je erachter komen dat we een json object moeten sturen om de 422 uit de validatie te krijgen. Een hele kleine wijziging in postProduct zal ervoor zorgen dat we werkelijk de 422 kunnen krijgen.

```
38     public function postProduct($overrides = [])
39     {
40         $this->withExceptionHandling();
41         $product = make( class: 'App\Product', [
42             $overrides
43         ]);
44         //return $this->post(route('products.store'), $product->toArray());
45         return $this->postJson(route( name: 'products.store'), $product->toArray());
46     }
}
```

Express even de oude code als commentaar gezet. Dus post() moet postJson() worden. Je moet namelijk een json object sturen als je het terug verwacht. En ja, dat verwachten we.

Als we de test uitvoeren zal je zien dat er nu geen error meer is. De test komt er nu dus uit zoals we verwachten, met een status 422.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest

  PASS  Tests\Feature\ProductStoreCheckTest
    ✓ a product requires a name
```

Nu we dit resultaat hebben kunnen we natuurlijk redelijk wat testen maken.

```
39 ► ◇ function a_product_name_can_be_max_45_characters()
40 {
41     $admin = \App\User::find(3);
42     $this->actingAs($admin);
43     $this->postProduct(['name' => '1234567890123456789012345678901234567890123456'])
44         ->assertStatus( status: 422);
45 }
46
47 /** @test */
48 ► ◇ function a_product_name_can_must_be_unique()
49 {
50     $admin = \App\User::find(3);
51     $product = \App\Product::find(1);
52     $this->actingAs($admin);
53     $this->postProduct(['name' => $product->name])
54         ->assertStatus( status: 422);
55 }
56
57 /** @test */
58 ► ◇ function a_product_requires_a_description()
59 {
60     $admin = \App\User::find(3);
61     $this->actingAs($admin);
62     $this->postProduct(['description' => null])
63         ->assertStatus( status: 422);
64 }
65
66
67 /** @test */
68 ► ◇ function a_product_price_should_be_a_number()
69 {
70     $admin = \App\User::find(3);
71     $this->actingAs($admin);
72     $this->postProduct(['price' => 'abc'])
73         ->assertStatus( status: 422);
74 }
75
76
77 /** @test */
78 ► ◇ function a_product_price_can_be_max_999999_99()
79 {
80     $admin = \App\User::find(3);
81     $this->actingAs($admin);
82     $this->postProduct(['price' => 1000000.00])
83         ->assertStatus( status: 422);
84 }
```

Als we nu de test uitvoeren zal je zien dat inderdaad alle testen correct zijn.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest

  PASS  Tests\Feature\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

  PASS  TestsFeatureProductsProductCreateTest
    ✓ 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  TestsFeatureProductsProductIndexTest
    ✓ 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  TestsFeatureProductsProductStoreTest
    ✓ customer can not create a product
    ✓ guest can not create a product
    ✓ admin can create a product
    ✓ sales can create a product
    ✓ customer can not create a product by posting
    ✓ admin can create a product without name

Tests: 21 passed
Time: 13.11s
```

Let ook even op, dat je het best de validatie erbij kunt pakken zodat je weet welke opties je allemaal kan testen.

```
14  public function rules()
15  {
16      return [
17          'name' => 'required|unique:products|max:45',
18          'description' => 'required',
19          'price' => 'required|numeric|max:999999.99'
20      ];
21  }
```

Voor nu is de productStoreCheckTest klaar. Alle opties hebben we nu getest.

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.

Net zoals bij de product create kunnen we dit gewoon met Pest doen. (leuk heh, Pest en phpunit door elkaar gebruiken)

```
ProductEditTest.php ×

1 <?php
2
3 beforeEach(function () {
4     $this->seed('RoleAndPermissionSeeder');
5     $this->seed('TestingUserSeeder');
6     $this->category = factory( class: App\Category::class)->create();
7     $this->product = factory( class: App\Product::class)->create();
8     $this->price = factory( class: App\Price::class)->create();
9 });
10
11 test( description: 'customer can not see the product edit page', function(){
12     $customer = App\User::find(1);
13
14     actingAs($customer)
15         ->get(route( name: 'products.edit', ['product' => $this->product->id]))
16         ->assertForbidden();
17 })->group( ...groups: 'ProductEdit');
```

De beforeEach is hetzelfde als altijd. Gewoon de basisgegevens klaar hebben in de database.

Bij de test kijken we of de customer erbij kan. Wel even opletten dat bij deze route de id van het product erbij moet. Deze kunnen we natuurlijk krijgen van het product wat we in de beforeEach hebben aangemaakt.

De test voor de gast is dan ook niet al te moeilijk.

```
19 test( description: 'guest can not see the product edit page', function(){
20     $this->get(route( name: 'products.edit', ['product' => $this->product->id]))
21         ->assertRedirect(route( name: 'login'));
22 })->group( ...groups: 'ProductEdit');
```

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.

```
24  ⌂ test( description: 'admin can see the product edit page', function(){
25      $admin = App\User::find(3);
26
27      actingAs($admin)
28          ->get(route( name: 'products.edit', ['product' => $this->product->id]))
29          ->assertViewIs( value: 'admin.products.edit')
30          ->assertSee($this->category->name)
31          ->assertSee($this->product->name)
32          ->assertSee($this->product->description)
33          ->assertSee($this->product->latest_price->price);
34  })->group( ...groups: 'ProductEdit');
35
36  ⌂ test( description: 'sales can see the product edit page', function(){
37      $sales = App\User::find(2);
38
39      actingAs($sales)
40          ->get(route( name: 'products.edit', ['product' => $this->product->id]))
41          ->assertViewIs( value: 'admin.products.edit')
42          ->assertSee($this->category->name)
43          ->assertSee($this->product->name)
44          ->assertSee($this->product->description)
45          ->assertSee($this->product->latest_price->price);
46  })->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:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=ProductEdit

  PASS  TestsFeatureProductsProductEditTest
    ✓ 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:   2.72s
```

De ProductEditTest is dan klaar.

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. Ook hier weer eerst de beforeEach

```
3  beforeEach(function () {
4      $this->seed('RoleAndPermissionSeeder');
5      $this->seed('TestingUserSeeder');
6      $this->category = factory( class: App\Category::class)->create();
7      $this->product = factory( class: App\Product::class)->create();
8      $this->price = factory( class: App\Price::class)->create();
9  });
10
11 test( description: 'customer can not update a product', function () {
12     $customer = App\User::find(1);
13
14     actingAs($customer)
15         ->patchJson(route( name: 'products.update', ['product' => $this->product->id]))
16         ->assertForbidden();
17 })->group( ...groups: 'ProductUpdate');
18
19 test( description: 'guest can not update a product', function () {
20     $this->patchJson(route( name: 'products.update', ['product' => $this->product->id]))
21         ->assertStatus(401);
22 })->group( ...groups: 'ProductUpdate');
```

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.

```
24     test( description: 'admin can update a product', function () {
25         $admin = App\User::find(3);
26         $newcategory = create( class: 'App\Category');
27
28         actingAs($admin);
29         $this->patchJson(route( name: 'products.update', ['product' => $this->product->id]),
30                         ['name' => 'Een productnaam',
31                          'description' => 'Blablabla',
32                          'price' => 1.00,
33                          'category_id' => $newcategory->id]
34         );
35
36         $this->product = $this->product->fresh();
37
38         $this->get(route( name: 'products.index'))
39             ->assertSee($this->product->name)
40             ->assertSee($this->product->category->name)
41             ->assertSee($this->product->latest_price->price);
42     })->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 je het zeker wilt weten, kan je natuurlijk ook dit doen:

```
38         $this->get(route( name: 'products.index'))
39             ->assertSee('Een productnaam')
40             ->assertSee(1.00)
41             ->assertSee($newcategory->name);
42
43         $this->get(route( name: 'products.index'))
44             ->assertSee($this->product->name)
45             ->assertSee($this->product->category->name)
46             ->assertSee($this->product->latest_price->price);
```

Als we de test uitvoeren zal je zien dat de test nu werkt.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=ProductUpdate

  PASS TestsFeatureProductsProductUpdateTest
  ✓ customer can not update a product
  ✓ guest can not update a product
  ✓ admin can update a product

Tests: 3 passed
Time: 2.11s
```

Voor Sales lijkt de test natuurlijk wel op die van de admin. De waardes die we meegeven voor de wijziging zijn alleen wat anders.

```
50   test( description: 'sales can update a product', function () {
51     $sales = App\User::find(2);
52     $newcategory = create( class: 'App\Category', ['name' => 'Nog een Categorie']);
53
54     actingAs($sales);
55     $this->patchJson(route( name: 'products.update', ['product' => $this->product->id]),
56       ['name' => 'Een 3e product',
57        'description' => 'Dit is een beschrijving',
58        'price' => 2.00,
59        'category_id' => $newcategory->id]
60     );
61
62     $this->product = $this->product->fresh();
63
64     $this->get(route( name: 'products.index'))
65       ->assertSee($this->product->name)
66       ->assertSee($this->product->category->name)
67       ->assertSee($this->product->latest_price->price);
68   })->group( ...groups: 'ProductUpdate');
```

En ook nu zien we als de test wordt uitgevoerd dat alles goed is.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=ProductUpdate

  PASS TestsFeatureProductsProductUpdateTest
  ✓ 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: 2.75s
```

ProductUpdateCheck met phpunit

Nu de ProductUpdateTest klaar is, gaan we aan de slag om de validatie ervan uit te testen. Hiervoor is het weer makkelijk om met een functie te werken, dus doen we dit met phpunit. Dus maken we deze test aan met artisan.

```
D:\laragon\www\workshop (master)
└─ php artisan make:test Products/ProductUpdateCheckTest
    Test created successfully.
```

De basis passen we dan aan, precies zoals we bij de store hebben gedaan. Ook al gebruik ik User nog niet, die zal ik straks wel nodig hebben.

```
1  <?php
2
3  namespace Tests\Feature\Products;
4
5  use Illuminate\Foundation\Testing\DatabaseMigrations;
6  use Illuminate\Foundation\Testing\WithFaker;
7  use Tests\TestCase;
8  use App\Category;
9  use App\Product;
10 use App\Price;
11 use App\User;
12
13 ► class ProductUpdateCheckTest extends TestCase
14 {
15     use DatabaseMigrations;
16
17     public function setUp(): void
18     {
19         // first include all the normal setUp operations
20         parent::setUp();
21
22         $this->seed( class: 'RoleAndPermissionSeeder' );
23         $this->seed( class: 'TestingUserSeeder' );
24         $this->category = factory( class: Category::class )->create();
25         $this->product = factory( class: Product::class )->create();
26         $this->price = factory( class: Price::class )->create();
27     }
28 }
```

Daarna gaan we werken aan de functie die we kunnen hergebruiken voor alle validatie tests. Je zal merken dat er niet veel verschil zit met die van de store. We hebben nog wel wat anders nodig

```
93     public function patchProduct($overrides = [])
94     {
95         $this->withExceptionHandling();
96         $product = edit( class: 'App\Product', [
97             $overrides
98         ]);
99         return $this->patchJson(route( name: 'products.update',
100             ['product' => $this->product->id]),
101             $product->toArray());
102    }
```

Het enige wat anders is, is dat het nu patchJson is en dat we sowieso de product id meeesturen in de route, zodat we ook bij de update methode terecht komen. De validatie van de store en update zijn gelijk. We gebruiken wel andere bestanden (Requests), dus we moeten het wel netjes testen.

Verder zie je een functie edit(). Deze heb ik extra aangemaakt in de functions.php.

```
1 <?php
2
3     function create($class, $attributes = [], $times = null)
4     {
5         return factory($class, $times)->create($attributes);
6     }
7
8     function make($class, $attributes = [], $times = null)
9     {
10        return factory($class, $times)->make($attributes);
11    }
12
13    function edit($class, $attributes = [])
14    {
15        return $class::find(1)->make($attributes);
16    }
```

Standaard hebben we namelijk 1 rij aangemaakt voordat we een edit uitvoeren. Deze moeten we dan ook ophalen. Je kan met make dan de wijzigingen aanbrengen.

De testen zien er dan zo uit

```
29      /** @test */
30      function a_product_update_requires_a_name()
31      {
32          $admin = \App\User::find(3);
33          $this->actingAs($admin);
34          $this->patchProduct(['name' => null])
35              ->assertStatus( status: 422);
36      }
37
38      /** @test */
39      function a_product_name_update_can_be_max_45_characters()
40      {
41          $admin = \App\User::find(3);
42          $this->actingAs($admin);
43          $this->patchProduct(['name' => '1234567890123456789012345678901234567890123456'])
44              ->assertStatus( status: 422);
45      }
46
47      /** @test */
48      function a_product_name_update_can_must_be_unique()
49      {
50          $admin = \App\User::find(3);
51          $product = \App\Product::find(1);
52          $this->actingAs($admin);
53          $this->patchProduct(['name' => $product->name])
54              ->assertStatus( status: 422);
55      }
56
57      /** @test */
58      function a_product_update_requires_a_description()
59      {
60          $admin = \App\User::find(3);
61          $this->actingAs($admin);
62          $this->patchProduct(['description' => null])
63              ->assertStatus( status: 422);
64      }
```

```

66      /** @test */
67      ► function a_product_update_requires_a_price()
68      {
69          $admin = \App\User::find(3);
70          $this->actingAs($admin);
71          $this->patchProduct(['price' => null])
72              ->assertStatus( status: 422);
73      }
74
75      /** @test */
76      ► function a_product_update_price_should_be_a_number()
77      {
78          $admin = \App\User::find(3);
79          $this->actingAs($admin);
80          $this->patchProduct(['price' => 'abc'])
81              ->assertStatus( status: 422);
82      }
83
84      /** @test */
85      ► function a_product_price_update_can_be_max_999999_99()
86      {
87          $admin = \App\User::find(3);
88          $this->actingAs($admin);
89          $this->patchProduct(['price' => 1000000.00])
90              ->assertStatus( status: 422);
91      }

```

Je ziet dat ik de functienamen netjes heb aangepast naar een update check. Verder gebruiken we de methode patchProduct() en controleren we of er een validatie error komt bij het updaten van een product.

Helaas komen al onze nieuwe testen er nog niet doorheen

FAIL Tests\Feature\Products\ProductUpdateCheckTest

- a product update requires a name
- a product name update can be max 45 characters
- a product name update can must be unique
- a product update requires a description
- a product update requires a price
- a product update price should be a number
- a product price update can be max 999999 99

Als we naar de errormelding kijken zien we dit

```
• Tests\Feature\Products\ProductUpdateCheckTest > a product update requires a name  
Illuminate\Database\Eloquent\MassAssignmentException  
Add [0] to fillable property to allow mass assignment on [App\Product].
```

We doen nu aan MassAssignment bij product en dat is niet toegestaan. De vraag is wel altijd of je dit wilt toestaan. We doen het nu even wel, waardoor we de model van Product een update geven.

In de model zorgen we dat name en description in \$fillable staan.

```
11 class Product extends Model  
12 {  
13       
14     protected $fillable = [  
15         'name', 'description'  
16     ];  
17 }
```

Als we dan opnieuw testen gaat wel alles goed

```
PASS Tests\Feature\Products\ProductUpdateCheckTest  
✓ a product update requires a name  
✓ a product name update can be max 45 characters  
✓ a product name update must be unique  
✓ a product update requires a description  
✓ a product update requires a price  
✓ a product update price should be a number  
✓ a product price update can be max 999999 99
```

De gehele ProductUpdateCheckTest is nu klaar.

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.

Hierdoor testen we de sales niet als de customer.

```
1  <?php
2
3  beforeEach(function () {
4      $this->seed('RoleAndPermissionSeeder');
5      $this->seed('TestingUserSeeder');
6      $this->category = factory( class: App\Category::class)->create();
7      $this->product = factory( class: App\Product::class)->create();
8      $this->price = factory( class: App\Price::class)->create();
9  });
10
11  test( description: 'sales can not see the product delete page', function(){
12      $sales = App\User::find(2);
13
14      actingAs($sales)
15          ->get(route( name: 'products.delete', ['product' => $this->product->id]))
16          ->assertForbidden();
17  })->group( ...groups: 'ProductDelete');
18
19  test( description: 'customer can not see the product delete page', function(){
20      $customer = App\User::find(1);
21
22      actingAs($customer)
23          ->get(route( name: 'products.delete', ['product' => $this->product->id]))
24          ->assertForbidden();
25  })->group( ...groups: 'ProductDelete');
```

De admin kan het natuurlijk wel zien. Daardoor kunnen we controleren of we gegevens van het product nu kunnen zien.

```
27     test( description: 'guest can not see the product delete page', function(){
28         $this->get(route( name: 'products.delete', ['product' => $this->product->id]))
29             ->assertRedirect(route( name: 'login'));
30     })->group( ...groups: 'ProductDelete');
31
32     test( description: 'admin can see the product delete page', function(){
33         $admin = App\User::find(3);
34
35         actingAs($admin)
36             ->get(route( name: 'products.delete', ['product' => $this->product->id]))
37             ->assertViewIs( value: 'admin.products.delete')
38             ->assertSee($this->category->name)
39             ->assertSee($this->product->name)
40             ->assertSee($this->product->description)
41             ->assertSee($this->product->latest_price->price);;
42     })->group( ...groups: 'ProductDelete');
```

Als we nu deze tests allemaal uitvoeren, zien we dat alles naar behoren werkt.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=ProductDelete

  PASS TestsFeatureProductsProductsDeleteTest
    ✓ sales can not see the product delete page
    ✓ customer can not see the product delete page
    ✓ guest can not see the product delete page
    ✓ admin can see the product delete page

Tests:  4 passed
Time:  2.68s
```

De ProductDeleteTest is dan ook meteen klaar.

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.

```
1  <?php
2
3  beforeEach(function () {
4      $this->seed('RoleAndPermissionSeeder');
5      $this->seed('TestingUserSeeder');
6      $this->category = factory( class: App\Category::class)->create();
7      $this->product = factory( class: App\Product::class)->create();
8      $this->price = factory( class: App\Price::class)->create();
9  });
10
11 test( description: 'sales can not delete a product', function(){
12     $sales = App\User::find(2);
13     actingAs($sales);
14     $this->json('DELETE', route( name: 'products.destroy', ['product' => $this->product->id]))
15         ->assertForbidden();
16 })->group( ...groups: 'ProductDestroy');
17
18 test( description: 'customer can not delete a product', function(){
19     $customer = App\User::find(1);
20     actingAs($customer);
21     $this->json('DELETE', route( name: 'products.destroy', ['product' => $this->product->id]))
22         ->assertForbidden();
23 })->group( ...groups: 'ProductDestroy');
24
25 test( description: 'guest can not delete a product', function(){
26     $this->json('DELETE', route( name: 'products.destroy', ['product' => $this->product->id]))
27         ->assertStatus(401);
28 })->group( ...groups: 'ProductDestroy');
29
30 test( description: 'admin can delete a product', function(){
31     $admin = App\User::find(3);
32     actingAs($admin);
33     $this->json('DELETE', route( name: 'products.destroy', ['product' => $this->product->id]));
34     $this->assertDatabaseMissing('products', ['id' => $this->product->id]);
35     $this->assertDatabaseMissing('prices', ['id' => $this->price->id]);
36 })->group( ...groups: 'ProductDestroy');
```

Bij de admin verwachten we dat het wel goed gaat. We kunnen dan ook kijken of het product en price weg zijn uit de database. Omdat price een relatie heeft met product en daar Cascade wordt gebruikt, wordt dus automatisch de prijs verwijderd.

De gehele test van destroy zal nu werken

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=ProductDestroy

  PASS TestsFeatureProductsProductDestroyTest
  ✓ sales can not delete a product
  ✓ customer can not delete a product
  ✓ guest can not delete a product
  ✓ admin can delete a product

Tests: 4 passed
Time: 2.65s
```

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:\laragon\www\workshop (master)
λ .\vendor\bin\pest

Tests: 43 passed
Time: 26.79s
```

Alles testen werken nu. Mooi, dit betekent wel dat je ondertussen weet hoe je om moet gaan met Feature Tests.

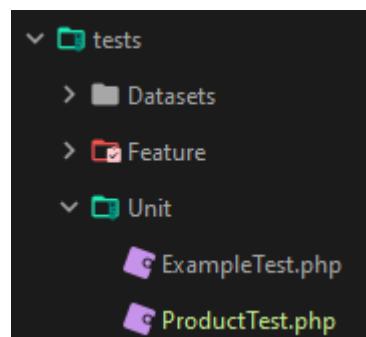
Unit Testing

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.

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.

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 permissions nodig voor product.



```
1  <?php
2
3  beforeEach(function () {
4      $this->category = factory( class: App\Category::class)->create();
5      $this->product = factory( class: App\Product::class)->create();
6      $this->price = factory( class: App\Price::class)->create();
7  });
8
```

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.

```
9  test( description: 'a product has prices', function(){
10     $this->assertInstanceOf('Illuminate\Database\Eloquent\Collection', $this->product->price);
11 })->group( ...groups: 'ProductUnit');
12
13 test( description: 'a product is inside a category', function(){
14     $this->assertInstanceOf('Illuminate\Database\Eloquent\Collection', $this->category->product);
15 })->group( ...groups: 'ProductUnit');
```

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()`.

```
17 test( description: 'a product name is a string', function(){
18     $this->assertIsString($this->product->name);
19 })->group( ...groups: 'ProductUnit');
20
21 test( description: 'a product description is a string', function(){
22     $this->assertIsString($this->product->description);
23 })->group( ...groups: 'ProductUnit');
```

We kunnen dan ook nog de standaard attributen testen. De id is natuurlijk een integer.

```
27     test( description: 'a product id is an int', function(){
28         $this->assertIsInt($this->product->id);
29     })->group( ...groups: 'ProductUnit');
30
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');
```

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.

Let wel op, dat we wel Carbon even bovenin moeten toevoegen met use.

```
3     use Carbon\Carbon;
4
5     beforeEach(function () {
6         $this->category = factory( class: App\Category::class)->create();
```

Als we nu de gehele unittest uitvoeren zien we dat alles goed is

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=ProductUnit

PASS TestsUnitProductTest
✓ a product has prices
✓ 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

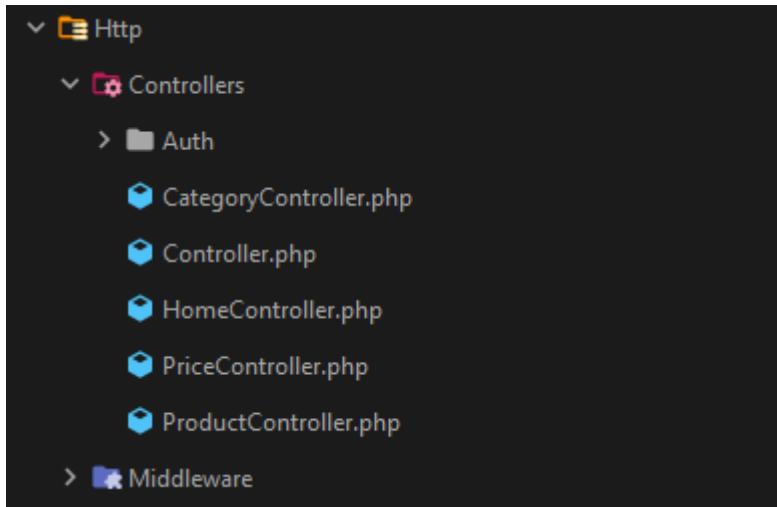
Tests: 7 passed
Time: 3.94s
```

De unit test is dus best snel klaar.

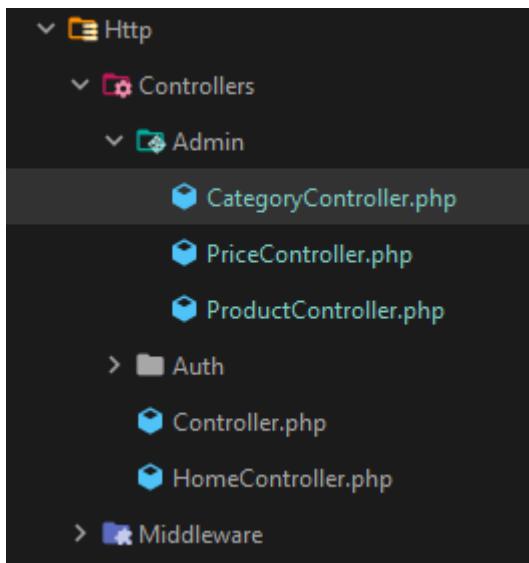
Wijzigingen en testen

Nu we binnen het testen de feature en unit testen hebben besproken, komen we op het volgende punt. Wanneer testen we nu precies. Hierdoor gaan we een aantal wijzigingen doorvoeren en kijken we wat er gaat gebeuren met de testen.

Als we naar onze webshop kijken hebben we nu de categorie en product admin klaar. Binnen de views hebben we wel al een admin map, maar bij voorbeeld de controllers niet. Als je een grotere website maakt is de organisatie van bestanden erg belangrijk, zodat je weet waar je wat kan vinden.



Alle controllers die we voor de admin gebruiken gaan we dan ook in de map Admin stoppen. De nieuwe situatie is dat ook dit.



Als we nu zouden testen krijgen we bij ongeveer alle testen wel een fout. Dit is ook erg logisch. Omdat we handmatig de bestanden verplaatst hebben, zullen we eerst in de console een dump-autoload moeten doen.

```
D:\laragon\www\workshop (master)
λ composer dump-autoload
Generating optimized autoload files> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
Discovered Package: barryvdh/laravel-debugbar
Discovered Package: facade/ignition
Discovered Package: fideloper/proxy
Discovered Package: fruitcake/laravel-cors
Discovered Package: laravel/tinker
Discovered Package: laravel/ui
Discovered Package: nesbot/carbon
Discovered Package: nunomaduro/collision
Discovered Package: pestphp/pest
Discovered Package: spatie/laravel-permission
Package manifest generated successfully.
Generated optimized autoload files containing 4649 classes
```

Daarnaast zullen we in alle 3 de controllers het volgende moeten toepassen

```
1 <?php
2
3 namespace App\Http\Controllers\Admin;
4
5 use App\Http\Controllers\Controller;
```

De namespace veranderd, omdat we in een nieuwe map Admin zitten. Wel willen we de basis Controller nog gebruiken, dus deze zetten we op dat moment bij use.

Daarna gaan we alle tests uitvoeren, maar wel met een extra optie. Namelijk dat de tests moeten stoppen als er 1 op failure komt.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --stop-on-failure
```

Je zal merken dat alle tests er nu wel door komen. Op dit moment 50 tests en alles werkt.

```
Tests: 50 passed
Time: 30.33s
```

Als we toch is even bij de ProductController de use als comment zetten en dan de test uitvoeren zie je het volgende

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --stop-on-failure

FAIL Tests\Feature\ProductStoreCheckTest
• a product requires a name

Expected status code 422 but received 500.
Failed asserting that 422 is identical to 500.

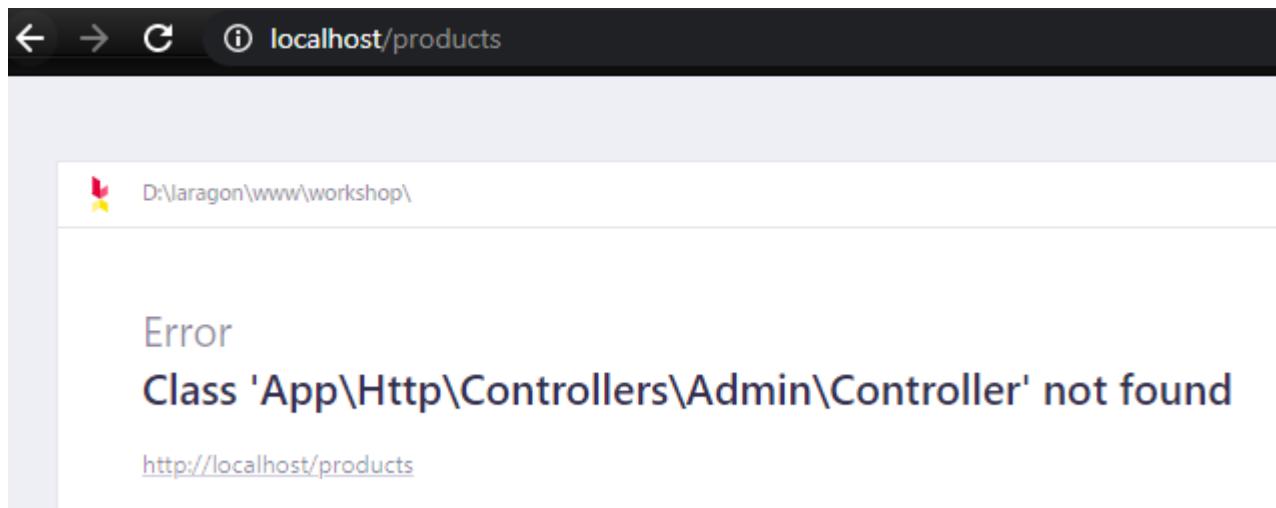
at D:\laragon\www\workshop\vendor\phpunit\phpunit\src\Framework\Constraint\Constraint.php:119
115|         if (!empty($description)) {
116|             $failureDescription = $description . "\n" . $failureDescription;
117|         }
118|
→ 119|         throw new ExpectationFailedException(
120|             $failureDescription,
121|             $comparisonFailure
122|         );
123|     }

1  D:\laragon\www\workshop\vendor\phpunit\phpunit\src\Framework\Constraint\IsIdentical.php:93
PHPUnit\Framework\Constraint\Constraint::fail("Expected status code 422 but received 500.")

2  D:\laragon\www\workshop\vendor\phpunit\phpunit\src\Framework\Assert.php:2228
PHPUnit\Framework\Constraint\IsIdentical::evaluate("Expected status code 422 but received 500.")

Tests: 1 failed, 49 pending
Time: 0.87s
```

Een 500 error, wat is als algemene server error. Ook uit de test halen we niet echt veel, want het is nog steeds erg vaag. Als we dan is gewoon op de site zelf testen, door bijvoorbeeld naar de products te gaan zien we dit.



Hier hebben we dus wel wat aan. Feature en Unit tests geven dus niet altijd weer waar het precies fout gaat. Je weet wel dat er wat fout is. In deze fout lees je ook dat de controller niet te vinden is.

Ook kan je meteen op de pagina lezen waar de fout zit.

```
D:\laragon\www\workshop\app\Http\Controllers\Admin\ProductController.php:19 ✎

4
5     //use App\Http\Controllers\Controller;
6
7     use App\Http\Requests\ProductStoreRequest;
8
9     use App\Http\Requests\ProductUpdateRequest;
10
11    use App\Price;
12
13    use App\Product;
14
15    use App\Category;
16
17    use Carbon\Carbon;
18
19    use Illuminate\Http\Request;
20
21    use Illuminate\Http\Response;

22
23 /**
24 * Class ProductController
25 * @package App\Http\Controllers
26 */
27
28 class ProductController extends Controller
```

Op regel 19 zit de fout met Controller. Deze is namelijk onbekend zonder de use, die nu even als comment staat. Natuurlijk zorgen we er wel weer even voor dat we de use gewoon weer gebruiken zodat de pagina weer werkt.

We hebben nu heel mooi gezien wat tests voor ons kunnen doen.

TDD met orders

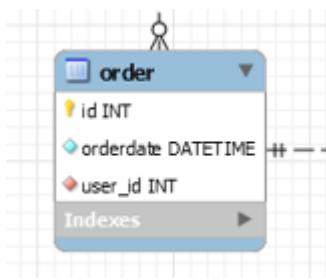
Nu we weten wat tests voor ons kunnen betekenen gaan we nog een stap verder, namelijk met Test Driven Development, ofwel TDD. Niet iedereen houd ervan, maar het zorgt er wel voor dat je heel gestructureerd gaat programmeren.

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!

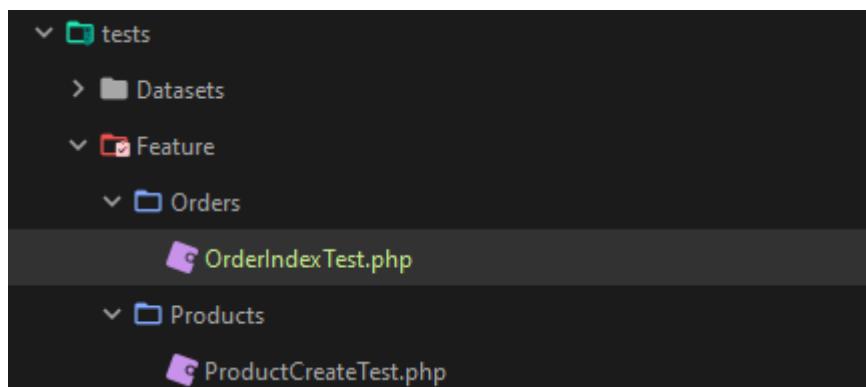
Maar de order heeft de velden id, orderdate en user_id.

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.



Index

We gaan simpel beginnen, namelijk met de test van de index. Hiervoor maken we in de map *tests/Feature* de map Orders aan. Daarin maken we de *OrderIndexTest.php* aan.



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()`

```
1  <?php
2
3  beforeEach(function () {
4      $this->seed('RoleAndPermissionSeeder');
5      $this->seed('TestingUserSeeder');
6      $this->order = factory( class: App\Order::class)->create();
7  });
8
```

Users maken we al aan met de seed *TestingUserSeeder*. Dus we hoeven alleen de order nog apart aan te maken.

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.

```
9  test( description: 'admin can see the order index page', function(){
10     $admin = App\User::find(3);
11
12     $this->withoutExceptionHandling();
13
14     $this->actingAs($admin)
15         ->get(route( name: 'orders.index'))
16         ->assertSee($this->order->id)
17         ->assertSee($this->order->orderdate)
18         ->assertSee($this->order->user->name)
19         ->assertStatus(200);
20 })->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.

```
22  test( description: 'sales can see the order index page', function(){
23     $sales = App\User::find(2);
24
25     $this->withoutExceptionHandling();
26
27     actingAs($sales)
28         ->get(route( name: 'orders.index'))
29         ->assertSee($this->order->id)
30         ->assertSee($this->order->orderdate)
31         ->assertSee($this->order->user->name)
32         ->assertStatus( status: 200);
33 })->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.

```
35     test( description: 'customer can not see the order index page', function(){
36         $customer = App\User::find(1);
37         actingAs($customer)
38             ->get(route( name: 'orders.index'))
39             ->assertForbidden();
40     })->group( ...groups: 'OrderIndex');
41
42     test( description: 'guest can not see the order index page', function(){
43         $this->get(route( name: 'orders.index'))
44             ->assertRedirect(route( name: 'login'));
45     })->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.

Het resultaat van de test is dit.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderIndex --stop-on-failure

FAIL Tests\Feature\Orders\OrderIndexTest
• admin can see the order index page

Error

Class 'App\Order' not found

at D:\laragon\www\workshop\vendor\laravel\framework\src\Illuminate\Database\Eloquent\FactoryBuilder.php:295
291     */
292     protected function makeInstance(array $attributes = [])
293     {
294         return Model::unguarded(function () use ($attributes) {
→ 295             $instance = new $this->class(
296                 $this->getRawAttributes($attributes)
297             );
298
299             if (isset($this->connection)) {
1   D:\laragon\www\workshop\vendor\laravel\framework\src\Illuminate\Database\Eloquent\Concerns\GuardedAttributes.php:148
 Illuminate\Database\Eloquent\FactoryBuilder::Illuminate\Database\Eloquent\{closure}()
2   D:\laragon\www\workshop\vendor\laravel\framework\src\Illuminate\Database\Eloquent\FactoryBuilder.php:304
 Illuminate\Database\Eloquent\Model::unguarded(Object(Closure))

Tests: 1 failed, 3 pending
Time: 0.80s
```

Ok, de class App\Order is er niet. Dit is de model van Order, die we dus nu gaan aanmaken.

```
D:\laragon\www\workshop (master)
λ 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:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderIndex --stop-on-failure

FAIL TestsFeatureOrdersOrderIndexTest
• admin can see the order index page

InvalidArgumentException

Unable to locate factory for [App\Order].
```

Uit de test blijkt dat er geen factory is voor order. De factory gaan we dus aanmaken

```
D:\laragon\www\workshop (master)
λ php artisan make:factory --model=Order OrderFactory
Factory created successfully.
```

Factory is aangemaakt. Nu de test weer.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderIndex --stop-on-failure

FAIL TestsFeatureOrdersOrderIndexTest
• admin can see the order index page

Illuminate\Database\QueryException

SQLSTATE[42S02]: Base table or view not found: 1146 Table 'testworkshop.orders' doesn't exist (SQL: insert into `orders`(`updated_at`, `created_at`) values (2020-06-11 07:39:23, 2020-06-11 07:39:23))
```

De tabel bestaat niet. Hiervoor moeten we een migration aanmaken en ook migraten.

```
D:\laragon\www\workshop (master)
λ php artisan make:migration --create=orders create_orders_table
Created Migration: 2020_06_11_100332_create_orders_table
```

Voor de zekerheid meteen een migrate + seed. Niet nodig voor de test, maar wel voor live omgeving.

```
D:\laragon\www\workshop (master)
λ 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 (0.02 seconds)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (0.02 seconds)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (0.03 seconds)
Migrating: 2020_05_10_143304_create_permission_tables
Migrated: 2020_05_10_143304_create_permission_tables (0.28 seconds)
Migrating: 2020_05_10_191303_create_categories_table
Migrated: 2020_05_10_191303_create_categories_table (0.01 seconds)
Migrating: 2020_05_15_084411_create_products_table
Migrated: 2020_05_15_084411_create_products_table (0.06 seconds)
Migrating: 2020_05_15_085736_create_prices_table
Migrated: 2020_05_15_085736_create_prices_table (0.04 seconds)
Migrating: 2020_06_11_074244_create_orders_table
Migrated: 2020_06_11_074244_create_orders_table (0.02 seconds)
Seeding: RoleAndPermissionSeeder
Seeded: RoleAndPermissionSeeder (0.07 seconds)
Seeding: TestingUserSeeder
Seeded: TestingUserSeeder (0.18 seconds)
Seeding: UserSeeder
Seeded: UserSeeder (0.1 seconds)
Seeding: CategorySeeder
Seeded: CategorySeeder (0.23 seconds)
Database seeding completed successfully.
```

Hierna weer de test uitvoeren

```
D:\laragon\www\workshop (master)
λ .\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.

```
37 |     Route::resource('name: '/admin/orders', 'Admin\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.

De test weer

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderIndex --stop-on-failure

FAIL TestsFeatureOrdersOrderIndexTest
• 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:\laragon\www\workshop (master)
λ php artisan make:controller --model=Order Admin\OrderController
Controller created successfully.
```

De test weer

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderIndex --stop-on-failure

FAIL TestsFeatureOrdersOrderIndexTest
• admin can see the order index page

Failed asserting that '' contains "1".
```

Er staat nu niet goed bij hoe of wat, maar als we onze test er nog is bij houden kunnen we kijken wat we verwachten.

```
9  test( description: 'admin can see the order index page', function(){
10     $admin = App\User::find(3);
11
12     $this->actingAs($admin)
13         ->get(route( name: 'orders.index'))
14         ->assertSee($this->order->id)
15         ->assertSee($this->order->orderdate)
16         ->assertSee($this->order->user->name)
17         ->assertStatus(200);
18 })->group( ...groups: 'OrderIndex');
```

De id zal 1 zijn, omdat we maar 1 order hebben. Dus hij ziet geen order id. Logisch natuurlijk, want we roepen niet eens een view aan in de methode. De index methode in onze controller is namelijk geheel leeg.

Deze gaan we dan ook netjes maken.

```
5  use App\Http\Controllers\Controller;
6  use App\User;
7  use App\Order;
8  use Illuminate\Http\Request;
9
10 class OrderController extends Controller
11 {
12     /**
13      * Display a listing of the resource. ...
14     */
15     public function index()
16     {
17         $orders = Order::with('user')->get();
18         return view('admin.orders.index', compact('orders'));
19     }
20 }
21 }
```

We gebruiken natuurlijk eager loading om de username te krijgen. Als we opnieuw gaan testen krijgen we dit.

```
D:\laragon\www\workshop (master)
λ .\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\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.

```
7  class Order extends Model
8  {
9      /**
10      * Get the user that owns the order.
11      *
12      * @return \App\User
13      */
14  }
```

De volgende test geeft dan dit.

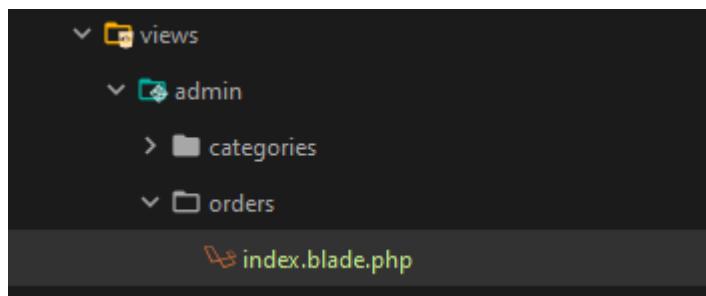
```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderIndex --stop-on-failure

FAIL TestsFeatureOrdersOrderIndexTest
• admin can see the order index page

InvalidArgumentException

View [admin.orders.index] not found.
```

In de methode staat namelijk dat we een view hebben in admin/orders/index.blade.php. Deze maken we dan ook netjes aan. We laten de view nog wel even leeg.



De volgende test geeft dit

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderIndex --stop-on-failure

FAIL TestsFeatureOrdersOrderIndexTest
• 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.

We beginnen met de view zoals we ook bij voorbeeld bij products hebben gedaan.

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

  2  @section('content')
    <h1 class="mt-5">Orders</h1>

  3  <nav class="nav">
    <ul class="nav nav-tabs">
      <li class="nav-item">
        <a class="nav-link active" href="{{ route('orders.index') }}">Index</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="{{ route('orders.create') }}">Create</a>
      </li>
    </ul>
  </nav>
```

Wel laat ik expres even de message weg, want dit gaan we straks weer ergens in een test terug krijgen.
In de tabel laat ik ook even nog de gebruiker en datum weg, zodat we zo weten of nu de assert op id er doorheen komt.

```
17  <table class="table .table-striped">
18    <thead class="thead-dark">
19      <tr>
20        <th scope="col">#</th>
21        <th scope="col">Gebruiker</th>
22        <th scope="col">Datum</th>
23        <th scope="col">Order details</th>
24        <th scope="col">Edit</th>
25        <th scope="col">Delete</th>
26      </tr>
27    </thead>
28    <tbody>
29      @foreach($orders as $order)
30        <tr>
31          <td scope="row">{{ $order->id }}</td>
32          <td></td>
33          <td></td>
34          <td><a href="{{ route('orders.show', ['order' => $order->id]) }}">Details</a></td>
35          <td><a href="{{ route('orders.edit', ['order' => $order->id]) }}">Edit</a></td>
36          <td><a href="{{ route('orders.delete', ['order' => $order->id]) }}">Delete</a></td>
37        </tr>
38      @endforeach
39    </tbody>
40  </table>
41  @endsection
```

Als we dit testen krijgen we een andere foutmelding

```
D:\laragon\www\workshop (master)
\ vendor\bin\pest --group=OrderIndex --stop-on-failure

FAIL TestsFeatureOrdersOrderIndexTest
• admin can see the order index page

Facade\Ignition\Exceptions\ViewException

Route [orders.delete] not defined. (View: D:\laragon\www\workshop\resources\views\admin\orders\index.blade.php)
```

Dit klopt ook, want bij de andere cruds hebben we hiervoor een aparte route voor gemaakt. Deze maken we dus ook even voor de order.

```
37     // Order Crud
38     Route::get('uri: 'admin/orders/{order}/delete', 'action: 'Admin\OrderController@delete')
39         ->name('orders.delete');
40     Route::resource('name: '/admin/orders', 'controller: 'Admin\OrderController');
```

Bij de volgende test komt dit er uit.

```
D:\laragon\www\workshop (master)
\ vendor\bin\pest --group=OrderIndex --stop-on-failure

FAIL TestsFeatureOrdersOrderIndexTest
• admin can see the order index page

ErrorException

Trying to get property 'name' of non-object

at D:\laragon\www\workshop\tests\Feature\Orders\OrderIndexTest.php:18
14|     $this->actingAs($admin)
15|         ->get(route('orders.index'))
16|         ->assertSee($this->order->id)
17|         ->assertSee($this->order->orderdate)
→ 18|         ->assertSee($this->order->user->name)
19|             ->assertStatus(200);
20| } )->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.

```
14     public function up()
15     {
16         Schema::create('orders', function (Blueprint $table) {
17             $table->id();
18
19             $table->unsignedBigInteger('user_id');
20             $table->foreign('user_id')
21                 ->references('id')->on('users')
22                 ->onDelete('cascade')
23                 ->onUpdate('cascade');
24             $table->timestamps();
25         });
26     }
```

De volgende test geeft dit.

```
D:\laragon\www\workshop (master)
λ .\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 (2020-06-11 14:17:24, 2020-06-11 14:17:24))
```

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.

```
5      use App\Order;
6
7      use App\User;
8
9      use Faker\Generator as Faker;
10
11     $factory->define(Order::class, function (Faker $faker) {
12
13         return [
14             // "user_id" => User::all()->random()->id
15         ];
16     });
17 }
```

Als we nu testen krijgen we in de test

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderIndex --stop-on-failure

FAIL TestsFeatureOrdersOrderIndexTest
• admin can see the order index page

Failed asserting that '<!doctype html>\n<html lang="en" class="h-100">\n<head>\n    <meta charset="utf-8">\n    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">\n\n        <title>Customer Orders</title>\n    <link href="/css/app.css" rel="stylesheet">\n</head>\n<body>\n    <div>\n        <h1>Customer Orders</h1>\n        <p>This is the orders index page. It lists all the orders for the current user.\n        </p>\n        <table border="1">\n            <thead>\n                <tr>\n                    <th>ID</th>\n                    <th>User Name</th>\n                    <th>Details</th>\n                    <th>Edit</th>\n                    <th>Delete</th>\n                </tr>\n            </thead>\n            <tbody>\n                <tr>\n                    <td>1</td>\n                    <td>Customer</td>\n                    <td><a href="/orders/1">Details</a></td>\n                    <td><a href="/orders/1/edit">Edit</a></td>\n                    <td><a href="/orders/1/delete">Delete</a></td>\n                </tr>\n            </tbody>\n        </table>\n    </div>\n</body>\n</html>' contains "Customer".
```

Er staat heel veel html in het resultaat van de test. Er wordt dan iets niet gevonden. Ergens tussen de html vinden we dit.

```
\n    ' contains "Customer".
```

Dit betekent dat de Customer, bij ons een naam van een user, niet op de pagina staat. We zullen dus de index.blade moeten aanpassen zodat deze erop komt te staan.

```
@foreach($orders as $order)
    <tr>
        <td scope="row">{{ $order->id }}</td>
        <td></td>
        <td>{{ $order->user->name }}</td>
        <td><a href="{{ route('orders.show', ['order' => $order->id]) }}">Details</a></td>
        <td><a href="{{ route('orders.edit', ['order' => $order->id]) }}">Edit</a></td>
        <td><a href="{{ route('orders.delete', ['order' => $order->id]) }}">Delete</a></td>
    </tr>
@endforeach
```

Als we testen komt er dit uit.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderIndex --stop-on-failure

FAIL TestsFeatureOrdersOrderIndexTest
✓ 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.

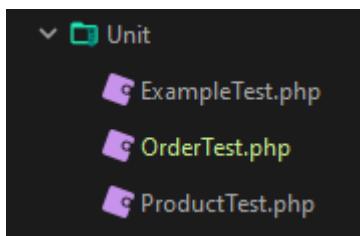
Als we even terug kijken naar de unit test van product stond dit erin.

```
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.

Order Unit Test

Bij de Unit tests maken we dan een OrderTest aan.



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.

```
11  test( description: 'a order orderdate is a datetime', function () {
12      $this->assertInstanceOf(Carbon::class, $this->order->orderdate);
13  })->group( ...groups: 'OrderUnit');
14
15  test( description: 'a order user id is an int', function () {
16      $this->assertIsInt($this->order->user_id);
17  })->group( ...groups: 'OrderUnit');
18
19  test( description: 'a order id is an int', function () {
20      $this->assertIsInt($this->order->id);
21  })->group( ...groups: 'OrderUnit');
22
23  test( description: 'a order created at is a datetime', function () {
24      $this->assertInstanceOf(Carbon::class, $this->order->created_at);
25  })->group( ...groups: 'OrderUnit');
26
27  test( description: 'a order updated at is a datetime', function () {
28      $this->assertInstanceOf(Carbon::class, $this->order->updated_at);
29  })->group( ...groups: 'OrderUnit');
```

De order unit tests zijn nu in de groep OrderUnit gezet, zodat we deze samen met de feature test kunnen uitvoeren.

Verder met TDD van Index

Nu we de unit test hebben, kunnen we deze samen met de OrderIndex test uitvoeren.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderIndex,OrderUnit --stop-on-failure

FAIL TestsUnitOrderTest
• 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.

```
$factory->define( class: Order::class, function (Faker $faker) {
    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:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderIndex,OrderUnit --stop-on-failure

FAIL TestsUnitOrderTest
• 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, 2020-06-11 17:50:21, 2020-06-11 17:50:21, 2020-06-11 17:50:21))
```

We gaan dus de migration aanpassen dat we orderdate in de tabel hebben

```
public function up()
{
    Schema::create('orders', function (Blueprint $table) {
        $table->id();
        $table->dateTime('orderdate');
        $table->unsignedBigInteger('user_id');
        $table->foreign('user_id')
            ->references('id')->on('users')
            ->onDelete('cascade')
            ->onUpdate('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:\laragon\www\workshop (master)
\ vendor\bin\pest --group=OrderIndex,OrderUnit --stop-on-failure

  PASS TestsUnitOrderTest
    ✓ 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 TestsFeatureOrdersOrderIndexTest
  • admin can see the order index page

  Failed asserting that '<!doctype html>\n<html lang="en" class="h-100">\n
```

Hier komt weer een heel stuk html in voor, dus we missen iets in de view.

```
\n
' contains "2020-06-11 17:52:00".
```

De orderdate staat er niet in. Deze zetten we netjes in de foreach.

```
@foreach($orders as $order)
  <tr>
    <td scope="row">{{ $order->id }}</td>
    <td>{{ $order->orderdate }}</td>
    <td>{{ $order->user->name }}</td>
    <td><a href="{{ route('orders.show', ['order' => $order->id]) }}">Details</a></td>
    <td><a href="{{ route('orders.edit', ['order' => $order->id]) }}">Edit</a></td>
    <td><a href="{{ route('orders.delete', ['order' => $order->id]) }}">Delete</a></td>
  </tr>
@endforeach
```

Als we weer testen krijgen we dit

```
D:\laragon\www\workshop (master)
\ vendor\bin\pest --group=OrderIndex,OrderUnit --stop-on-failure

  PASS TestsUnitOrderTest
    ✓ 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 TestsFeatureOrdersOrderIndexTest
  ✓ 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.
```

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.

```
26 |     Permission::create(['name' => 'index order']);
27 |     Permission::create(['name' => 'create order']);
28 |     Permission::create(['name' => 'edit order']);
29 |     Permission::create(['name' => 'delete order']);
```

Deze permissies geven we aan de sales

```
33 |     $role = Role::create(['name' => 'sales']);
34 |     $role->givePermissionTo('create category', 'edit category');
35 |     $role->givePermissionTo('create product', 'edit product');
36 |     $role->givePermissionTo('index 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.

```
9  class OrderController extends Controller
10 {
11     /**
12      * Set permissions on methods
13     */
14
15    public function __construct()
16    {
17        $this->middleware( middleware: 'auth');
18        $this->middleware( middleware: 'permission:index order', ['only' => ['index']]);
19        $this->middleware( middleware: 'permission:create order', ['only' => ['create', 'store']]);
20        $this->middleware( middleware: 'permission:edit order', ['only' => ['edit', 'update']]);
21        $this->middleware( middleware: 'permission:delete order', ['only' => ['delete', 'destroy']]);
22    }
}
```

Als we de test nu uitvoeren zien we dit

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderIndex,OrderUnit --stop-on-failure

  PASS TestsUnitOrderTest
    ✓ 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 TestsFeatureOrdersOrderIndexTest
    ✓ 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

    ...
    • TestsFeatureOrdersOrderIndexTest > guest can not see the order index page
      Illuminate\Auth\AuthenticationException
        Unauthenticated.
```

We zijn er bijna, alleen nog zorgen dat je minimaal ingelogd moet zijn om erbij te komen. We verhuizen de routes binnen de middleware om dat te regelen.

```
25   Route::group(['middleware' => ['role:sales|admin']], function () {
26     // Order Crud
27     Route::get( uri: 'admin/orders/{order}/delete',  action: 'Admin\OrderController@delete')
28       ->name( name: 'orders.delete');
29     Route::resource( name: '/admin/orders',  controller: 'Admin\OrderController');
```

Als we nu testen zien we dat we aan alle testen voldoen.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderIndex,OrderUnit --stop-on-failure

  PASS TestsUnitOrderTest
    ✓ 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 TestsFeatureOrdersOrderIndexTest
    ✓ 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: 5.91s
```

Voordat we het nu op de site gaan bekijken, zullen we eerst een migrate en seed moeten doen. We hebben namelijk een nieuwe tabel met allerlei kolommen erin.

```
D:\laragon\www\workshop (master)
└ php artisan migrate:fresh --seed
```

Als we nu even op de site gaan kijken bij orders, als bijv admin, zien we dit

Ja, er staan geen orders in. We hebben namelijk geen seed voor de database. Met de test gebruiken we namelijk de testdatabase en deze wordt na gebruik steeds leeg gemaakt.

Voor de zekerheid kunnen we nog even alle tests laten uitvoeren.

```
D:\laragon\www\workshop (master)
└ .\vendor\bin\pest --stop-on-failure
```

We zien dat alle tests nu slagen!

```
Tests: 59 passed
Time: 39.32s
```

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.

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.

We hebben net als bij de index permissies, 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.

```
3  beforeEach(function () {
4      $this->seed('RoleAndPermissionSeeder');
5      $this->seed('TestingUserSeeder');
6      $this->order = factory( class: App\Order::class)->create();
7  });
8
9  test( description: 'admin can see the order create page', function(){
10     $admin = App\User::find(3);
11     $this->withoutExceptionHandling();
12     actingAs($admin)
13         ->get(route( name: 'orders.create'))
14         ->assertViewIs( value: 'admin.orders.create');
15 })->group( ...groups: 'OrderCreate');
```

```
17  test( description: 'sales can see the order create page', function(){
18     $sales = App\User::find(2);
19     $this->withoutExceptionHandling();
20     actingAs($sales)
21         ->get(route( name: 'orders.create'))
22         ->assertViewIs( value: 'admin.orders.create');
23 })->group( ...groups: 'OrderCreate');

25  test( description: 'customer can not see the order create page', function(){
26     $customer = App\User::find(1);

27
28     actingAs($customer)
29         ->get(route( name: 'orders.create'))
30         ->assertForbidden();
31 })->group( ...groups: 'OrderCreate');

33  test( description: 'guest can not see the order create page', function(){
34     $this->get(route( name: 'orders.create'))
35         ->assertRedirect(route( name: 'login'));
36 })->group( ...groups: 'OrderCreate');
```

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:\laragon\www\workshop (master)
\ .\vendor\bin\pest --group=OrderCreate,OrderUnit --stop-on-failure

PASS TestsUnitOrderTest
✓ 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 TestsFeatureOrdersOrderCreateTest
• 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.

```
41     public function create()
42     {
43         //
44         $users = User::all();
45         return view( view: 'admin.orders.create', compact( varname: 'users'));
46     }
```

Natuurlijk hebben we straks de users nodig, omdat deze aan een order gekoppeld zijn. Ik stuur dus alvast deze info mee. De volgende test geeft dit weer.

```
D:\laragon\www\workshop (master)
\ .\vendor\bin\pest --group=OrderCreate,OrderUnit --stop-on-failure

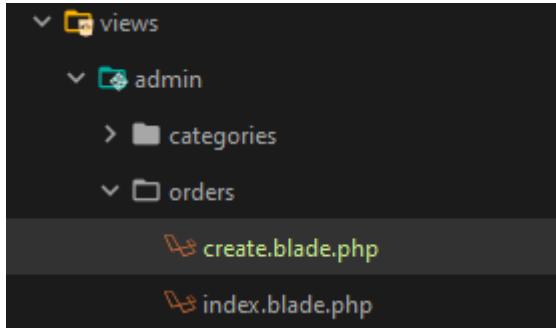
PASS TestsUnitOrderTest
✓ a order orderdate is a datetime
✓ a order user id is an int
✓ a order id is an int
✓ a order created at is a datetime
✓ a order updated at is a datetime

FAIL TestsFeatureOrdersOrderCreateTest
• 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:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderCreate,OrderUnit --stop-on-failure

  PASS TestsUnitOrderTest
  ✓ 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 TestsFeatureOrdersOrderCreateTest
  ✓ 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: 5.89s
```

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.

```
1  ⚡ 🐱  @extends('layouts.layout')
2
3  ↗ @section('content')
4      <h1 class="mt-5">Orders</h1>
5
6      ↗ @if ($errors->any())
7          ↗ <div class="alert alert-danger">
8              ↗ <ul>
9                  ↗ @foreach ($errors->all() as $error)
10                     ↗ <li>{{ $error }}</li>
11                 ↗ @endforeach
12             ↗ </ul>
13         ↗ </div>
14     ↗ @endif
15
16     ↗ <nav class="nav">
17         ↗ <ul class="nav nav-tabs">
18             ↗ <li class="nav-item">
19                 ↗ <a class="nav-link" href="{{ route('orders.index') }}">Index</a>
20             ↗ </li>
21             ↗ <li class="nav-item">
22                 ↗ <a class="nav-link active" href="{{ route('orders.create') }}">Create</a>
23             ↗ </li>
24         ↗ </ul>
25     ↗ </nav>
```

```
27     ↗ <form method="POST" action="{{ route('orders.store') }}">
28         ↗ @csrf
29         ↗ <div class="form-group">
30             ↗ <label for="name">Productnaam</label>
31             ↗ <input type="datetime-local" name="orderdate" class="form-control" id="orderdate" aria-describedby="orderdateHelp"
32                 ↗ placeholder="Kies Order datum" value="{{ old('orderdate') }}"/>
33         ↗ </div>
34         ↗ <div class="form-group">
35             ↗ <label for="user_id">User</label>
36             ↗ <select name="user_id" id="user_id" class="form-control">
37                 ↗ @foreach($users as $user)
38                     ↗ <option value="{{ $user->id }}>
39                         ↗ @if( old('user_id') == $user->id )
40                             ↗ selected
41                         ↗ @endif
42                     >{{ $user->name }}</option>
43                 ↗ @endforeach
44             ↗ </select>
45         ↗ </div>
46         ↗ </form>
47     ↗ @endsection
```

Het resultaat is dat we een net formulier hebben.

The screenshot shows a web browser window with the URL `localhost/admin/orders/create`. The page has a dark header bar with the text "Fixed navbar" and "Admin". Below the header is a title "Orders". There are two buttons: "Index" and "Create", where "Create" is highlighted with a border. The main form area contains fields for "Productname" (with placeholder "dd-mm-jjjj --:--" and a calendar icon), "User" (with dropdown menu showing "Customer"), and a blue "Submit" button.

Fixed navbar Home Link Disabled Product Test Admin Admin

Orders

Index Create

Productname

dd-mm-jjjj --:--

User

Customer

Submit

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. Hierin zit natuurlijk een beforeEach en gaan we alle gebruikers af of ze er bij kunnen komen.

```
3  beforeEach(function () {
4      $this->seed('RoleAndPermissionSeeder');
5      $this->seed('TestingUserSeeder');
6      $this->order = factory( class: App\Order::class)->create();
7  });
8
9  test( description: 'guest can not create an order in the admin', function () {
10     $this->postJson(route( name: 'orders.store'))
11         ->assertStatus(401);
12 })->group( ...groups: 'OrderStore');
13
14 test( description: 'customer can not create an order in the admin', function () {
15     $customer = App\User::find(1);
16     actingAs($customer)
17         ->postJson(route( name: 'orders.store'))
18         ->assertForbidden();
19 })->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 () {
    $this->withoutExceptionHandling();
    $admin = App\User::find(3);
    actingAs($admin);
    $order = make( class: 'App\Order', ['user_id' => 1, 'orderdate' => \Carbon\Carbon::now()->toDateTimeString()]);
    $this->postJson(route( name: 'orders.store'), $order->toArray())
        ->assertRedirect(route( name: 'orders.index'));

    $this->assertDatabaseHas('orders', [
        'user_id' => 1,
        'orderdate' => $order->orderdate
    ]);
})->group( ...groups: 'OrderStore');

test( description: 'sales can create an order in the admin', function () {
    $this->withoutExceptionHandling();
    $sales = App\User::find(2);
    actingAs($sales);
    $order = make( class: 'App\Order', ['user_id' => 1, 'orderdate' => \Carbon\Carbon::now()->toDateTimeString()]);
    $response = $this->postJson(route( name: 'orders.store'), $order->toArray())
        ->assertRedirect(route( name: 'orders.index'));

    $this->assertDatabaseHas('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.

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

```
3  use App\User;  
4  
5  beforeEach(function () {  
6      $this->seed('RoleAndPermissionSeeder');  
7      $this->seed('TestingUserSeeder');  
8      $this->order = factory( class: App\Order::class)->create();  
9  });  
10  
11  test( description: 'an_order_requires_a_user', function () {  
12      $admin = User::find(3);  
13      $this->actingAs($admin);  
14      $product = make( class: 'App\Order', [  
15          'user_id' => null  
16      ]);  
17      $this->postJson(route( name: 'orders.store'), $product->toArray())->assertStatus(422);  
18  })->group( ...groups: 'OrderStoreCheck');  
19  
20  test( description: 'an_user_id_must_be_an_integer', function () {  
21      $admin = User::find(3);  
22      $this->actingAs($admin);  
23      $product = make( class: 'App\Order', [  
24          'user_id' => 'bla'  
25      ]);  
26      $this->postJson(route( name: 'orders.store'), $product->toArray())->assertStatus(422);  
27  })->group( ...groups: 'OrderStoreCheck');  
28  
29  test( description: 'an_order_belongs_to_an_user', function () {  
30      $admin = User::find(3);  
31      $this->actingAs($admin);  
32      $order = make( class: 'App\Order');  
33      $this->assertInstanceOf('App\User', $order->user);  
34  })->group( ...groups: 'OrderStoreCheck');
```

Als we nu beginnen met testen krijgen we als eerst dit

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderStore,OrderStoreCheck,OrderUnit --stop-on-failure

  PASS | TestsUnitOrderTest
    ✓ 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 | TestsFeatureOrdersOrderStoreCheckTest
    • 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:\laragon\www\workshop (master)
λ php artisan make:request OrderStoreRequest
Request created successfully.
```

En vullen dan nu alleen in dat de user_id verplicht is.

```
3   namespace App\Http\Requests;
4
5   use Illuminate\Foundation\Http\FormRequest;
6
7   class OrderStoreRequest extends FormRequest
8   {
9       /**
10      * Determine if the user is authorized to make this request. ...
11     */
12     public function authorize()
13     {
14         return true;
15     }
16
17     /**
18      * Get the validation rules that apply to the request. ...
19     */
20     public function rules()
21     {
22         return [
23             // ...
24             'user_id' => 'required'
25         ];
26     }
27 }
28
29 }
```

Als we nu gaan testen gaat de order requires a user goed.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderStore,OrderStoreCheck,OrderUnit --stop-on-failure

  PASS TestsUnitOrderTest
    ✓ 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 TestsFeatureOrdersOrderStoreCheckTest
    ✓ 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()
{
    //dd(\Request::get('user_id'));

    return [
        //
        'user_id' => 'required|integer'
    ];
}
```

Als we nu testen komen we er wel doorheen. De volgende fout zit bij de create van een order.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderStore,OrderStoreCheck,OrderUnit --stop-on-failure

  PASS TestsUnitOrderTest
    ✓ 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 TestsFeatureOrdersOrderStoreCheckTest
    ✓ an order requires a user
    ✓ an user id must be an integer
    ✓ an order belongs to an user

  FAIL TestsFeatureOrdersOrderStoreTest
    ✓ 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 '<!doctype html>\n<html lang="en" class="h-100">\n
```

We testen natuurlijk of deze in de database gaat 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 wat het werd nog niet opgeslagen

```
55     public function store(OrderStoreRequest $request)
56     {
57         //
58         $order = new Order();
59         $order->user_id = $request->user_id;
60         $order->orderdate = $request->orderdate;
61         $order->save();
62     }
```

Zodra we de methode dan zo maken dat het wordt opgeslagen. Als we testen merken we dat we de redirect zijn vergeten

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderStore --stop-on-failure

FAIL TestsFeatureOrdersOrderStoreTest
✓ 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.
```

We voegen dus nog even de redirect toe aan de methode.

```
55     public function store(OrderStoreRequest $request)
56     {
57         //
58         $order = new Order();
59         $order->user_id = $request->user_id;
60         $order->orderdate = $request->orderdate;
61         $order->save();

63         return redirect()->route('orders.index')->with('message', 'Order toegevoegd');
64     }
```

Als we nu testen zien we dat alles werkt.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderStore,OrderStoreCheck,OrderUnit --stop-on-failure

  PASS  TestsUnitOrderTest
    ✓ 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  TestsFeatureOrdersOrderStoreCheckTest
    ✓ an order requires a user
    ✓ an user id must be an integer
    ✓ an order belongs to an user

  PASS  TestsFeatureOrdersOrderStoreTest
    ✓ 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: 8.14s
```

De gehele store werkt dus zoals we het willen.

Show

De show methode gaan we nu nog niet doen omdat er erg weinig in zal komen. Bij de show wil je straks natuurlijk zien wat er werkelijk in de order zit. Hiervoor hebben we orderrow nodig.

In de index staat dus op dit moment alles wat we nodig hebben.

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, met de volgende tests.

```
3  beforeEach(function () {
4      $this->seed('RoleAndPermissionSeeder');
5      $this->seed('TestingUserSeeder');
6      $this->order = factory( class: App\Order::class)->create();
7  });
8
9  test( description: 'guest can not see the order edit page', function(){
10     $this->get(route( name: 'orders.edit', ['order' => $this->order->id]))
11         ->assertRedirect(route( name: 'login'));
12 })->group( ...groups: 'OrderEdit');
13
14 test( description: 'customer can not see the order edit page', function(){
15     $customer = App\User::find(1);
16
17     actingAs($customer)
18         ->get(route( name: 'orders.edit', ['order' => $this->order->id]))
19         ->assertForbidden();
20 })->group( ...groups: 'OrderEdit');
21
22 test( description: 'sales can see the order edit page', function(){
23     $sales = App\User::find(2);
24     $this->withoutExceptionHandling();
25
26     actingAs($sales)
27         ->get(route( name: 'orders.edit', ['order' => $this->order->id]))
28         ->assertViewIs( value: 'admin.orders.edit')
29         ->assertSee($this->order->user->name)
30         ->assertSee($this->order->orderdate);
31 })->group( ...groups: 'OrderEdit');
32
33 test( description: 'admin can see the order edit page', function(){
34     $admin = App\User::find(3);
35     $this->withoutExceptionHandling();
36
37     actingAs($admin)
38         ->get(route( name: 'orders.edit', ['order' => $this->order->id]))
39         ->assertViewIs( value: 'admin.orders.edit')
40         ->assertSee($this->order->user->name)
41         ->assertSee($this->order->orderdate);
42 })->group( ...groups: 'OrderEdit');
```

Als we nu beginnen met testen zien we dit

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderEdit,OrderUnit --stop-on-failure

  PASS TestsUnitOrderTest
    ✓ 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 TestsFeatureOrdersOrderEditTest
    ✓ 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

```
83     public function edit(Order $order)
84     {
85         //
86         return view( view: 'admin.orders.edit', compact( varname: 'order'));
87     }
```

De volgende test

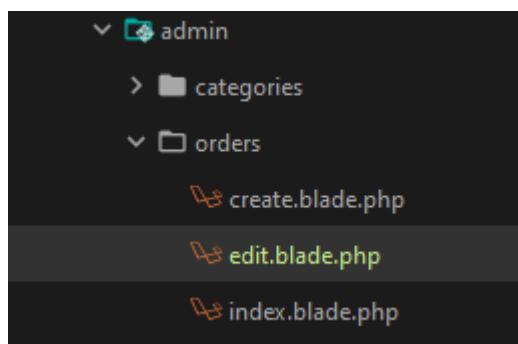
```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderEdit --stop-on-failure

  FAIL TestsFeatureOrdersOrderEditTest
    ✓ 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

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderEdit --stop-on-failure

 FAIL TestsFeatureOrdersOrderEditTest
✓ 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 "Sales".
```

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

```
1  * @extends('layouts.layout')
2
3  * @section('content')
4      <h1 class="mt-5">Orders</h1>
5
6      @if ($errors->any())
7          <div class="alert alert-danger">
8              <ul>
9                  @foreach ($errors->all() as $error)
10                     <li>{{ $error }}</li>
11                 @endforeach
12             </ul>
13         </div>
14     @endif
15
16     <nav class="nav">
17         <ul class="nav nav-tabs">
18             <li class="nav-item">
19                 <a class="nav-link" href="{{ route('orders.index') }}">Index</a>
20             </li>
21             <li class="nav-item">
22                 <a class="nav-link" href="{{ route('orders.create') }}">Create</a>
23             </li>
24             <li class="nav-item">
25                 <a class="nav-link active" href="{{ route('orders.edit', [
26                     'order' => $order->id]) }}">Edit Order</a>
27             </li>
28         </ul>
29     </nav>
```

```

31      <form method="POST" action="{{ route('orders.update', ['order' => $order->id]) }}>
32          @method('PUT')
33          @csrf
34          <div class="form-group">
35              <label for="name">Orderdate</label>
36              <input type="datetime-local" name="orderdate" class="form-control" id="orderdate"
37                  aria-describedby="orderdateHelp" placeholder="Kies Order datum"
38                  value="{{ old('orderdate', $order->orderdate) }}"
39          </div>
40          <div class="form-group">
41              <label for="user_id">User</label>
42              <select name="user_id" id="user_id" class="form-control">
43                  @foreach($users as $user)
44                      <option value="{{ $user->id }}>
45                          @if( old('user_id', $order->user_id) == $user->id )
46                              selected
47                          @endif
48                      >{{ $user->name }}</option>
49                  @endforeach
50              </select>
51          </div>
52          <button type="Submit" class="btn btn-primary">Submit</button>
53      </form>
54  @endsection

```

Als we opnieuw gaan testen zien we dit

```

D:\laragon\www\workshop (master)
\ vendor\bin\pest --group=OrderEdit --stop-on-failure

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:\laragon\www\workshop\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 meeesturen naar de view.

```

80      /** Show the form for editing the specified resource. ...*/
81
82      public function edit(Order $order)
83      {
84          $users = User::all();
85
86          return view('admin.orders.edit', compact('order', '_users'));
87      }

```

Als we nu kijken naar de test

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderEdit --stop-on-failure

  PASS  TestsFeatureOrdersOrderEditTest
    ✓ 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:  4 passed
  Time:   3.15s
```

De test is geslaagd. Maar.... Als we op de pagina gaan kijken is de datum niet ingevuld.

The screenshot shows a web page titled 'Orders'. At the top, there are three buttons: 'Index', 'Create', and 'Edit Order', with 'Edit Order' being the active one. Below these are two input fields. The first field is labeled 'Orderdate' and contains the placeholder 'dd-mm-jjjj --:--'. The second field is labeled 'User' and contains the value 'Customer'. At the bottom of the form is a blue 'Submit' button.

Als we naar de bron kijken zien we dit

```
<input type="datetime-local" name="orderdate" class="form-control" id="orderdate"
aria-describedby="orderdateHelp" placeholder="Kies Order datum"
value="2020-06-21 16:52:00">
```

Dit betekent dus, dat de test wel lukt omdat de waarde er netjes staat. Helaas is dit niet de format die een formulier wilt hebben. Het formulier zou er zo uit moeten zien in code:

```
<input type="datetime-local" name="orderdate" class="form-control" id="orderdate"
aria-describedby="orderdateHelp" placeholder="Kies Order datum"
value="2020-06-21T16:52">
```

Die T ertussen maakt het grote verschil uit binnen html.

Wat we doen is, we veranderen onze test zodat de goede format wordt verwacht. Hiervoor gebruiken we Carbon voor, dus dit zetten we bovenin de test.

```
1 <?php
2
3 use Carbon\Carbon;
```

De test zelf wordt dit. (let op dat je beide aanpast)

```
24 test( description: 'sales can see the order edit page', function(){
25     $sales = App\User::find(2);
26     $this->withoutExceptionHandling();
27
28     actingAs($sales)
29         ->get(route( name: 'orders.edit', ['order' => $this->order->id]))
30         ->assertViewIs( value: 'admin.orders.edit')
31         ->assertSee($this->order->user->name)
32         ->assertSee(Carbon::createFromFormat( format: 'Y-m-d H:i:s', $this->order->orderdate)
33             ->format( format: 'Y-m-d\TH:i'));
34     })->group( ...groups: 'OrderEdit');
35
36 test( description: 'admin can see the order edit page', function(){
37     $admin = App\User::find(3);
38     $this->withoutExceptionHandling();
39
40     actingAs($admin)
41         ->get(route( name: 'orders.edit', ['order' => $this->order->id]))
42         ->assertViewIs( value: 'admin.orders.edit')
43         ->assertSee($this->order->user->name)
44         ->assertSee(Carbon::createFromFormat( format: 'Y-m-d H:i:s', $this->order->orderdate)
45             ->format( format: 'Y-m-d\TH:i'));
46     })->group( ...groups: 'OrderEdit');
```

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 krijgen we een foutmelding

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderEdit --stop-on-failure

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'
```

Met daarbij de verwachte goede format, die we nog niet hebben in ons formulier.

```
\n  ' contains "2020-06-20T17:48".
```

Om dit te krijgen gaan we de controller veranderen.

```
79     /** Show the form for editing the specified resource. ...*/
80
81     public function edit(Order $order)
82     {
83
84         $users = User::all();
85
86         $order->orderdate = Carbon::createFromFormat( format: 'Y-m-d H:i:s', $order->orderdate)
87             ->format( format: 'Y-m-d\TH:i');
88
89         return view( view: 'admin.orders.edit', compact( varname: 'order', _: 'users'));
90
91     }
92 }
```

Hier doe ik dus eigenlijk hetzelfde als in de test. Wat wel nu is, is dat we een goed resultaat krijgen

Als we de test uitvoeren komt hij door alles heen.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderEdit --stop-on-failure

[PASS] TestsFeatureOrdersOrderEditTest
✓ 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: 4 passed
Time: 3.14s
```

Als ik dan op de site ga kijken wanneer ik een order wil aanpassen, zie ik dit

Orders

[Index](#) [Create](#) [Edit Order](#)

Orderdate

21-06-2020 16:52

User

Customer

Submit

De datum en tijd zijn nu ingevuld zoals we hadden verwacht. Eindelijk werkt nu echt alles van de edit.

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.

In de OrderUpdateTest komt dan dit.

```
3      use Carbon\Carbon;
4
5      use App\User;
6
7      use App\Order;
8
9
10     beforeEach(function () {
11         $this->seed('RoleAndPermissionSeeder');
12         $this->seed('TestingUserSeeder');
13         $this->order = factory(\App\Order::class)->create();
14     });
15
16     test('guest can not update an order in the admin', function () {
17         $this->patchJson(route('orders.update', ['order' => $this->order->id]))
18             ->assertStatus(401);
19     })->group(...groups: 'OrderUpdate');
20
21     test('customer can not update an order in the admin', function () {
22         $customer = User::find(1);
23         actingAs($customer)
24             ->patchJson(route('orders.update', ['order' => $this->order->id]))
25             ->assertForbidden();
26     })->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.

```
25     test( description: 'admin can update an order in the admin', function () {
26         $this->withoutExceptionHandling();
27         $admin = User::find(3);
28         actingAs($admin);
29         $order = make( class: 'App\Order', ['user_id' => 1, 'orderdate' => Carbon::now()->toDateTimeString()]);
30         $newdate = Carbon::now()->addDays( value: 10)->toDateTimeString();
31         $this->patchJson(route( name: 'orders.update', ['order' => $this->order->id]),
32                           ['user_id' => 2, 'orderdate' => $newdate])->assertRedirect(route( name: 'orders.index'));
33
34         $this->assertDatabaseHas('orders', ['user_id' => 2, 'orderdate' => $newdate]);
35     })->group( ...groups: 'OrderUpdate');
36
37     test( description: 'sales can update an order in the admin', function () {
38         $this->withoutExceptionHandling();
39         $sales = User::find(2);
40         actingAs($sales);
41         $order = make( class: 'App\Order', ['user_id' => 1, 'orderdate' => Carbon::now()->toDateTimeString()]);
42         $newdate = Carbon::now()->addDays( value: 10)->toDateTimeString();
43         $this->patchJson(route( name: 'orders.update', ['order' => $this->order->id]),
44                           ['user_id' => 2, 'orderdate' => $newdate])->assertRedirect(route( name: 'orders.index'));
45
46         $this->assertDatabaseHas('orders', ['user_id' => 2, 'orderdate' => $newdate]);
47     })->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.

Als we gaan testen krijgen we dit

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderUpdate,OrderUpdateCheck --stop-on-failure

FAIL Tests\Feature\Orders\OrderUpdateCheckTest
• an order update requires a 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.

```
100     public function update(OrderStoreRequest $request, Order $order)
101     {
102         //
103     }
```

Als we dan weer testen zien we dit

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderUpdate,OrderUpdateCheck --stop-on-failure

  PASS TestsFeatureOrdersOrderUpdateCheckTest
    ✓ an order update requires a user
    ✓ an user id must be an integer

  FAIL TestsFeatureOrdersOrderUpdateTest
    ✓ 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.

```
100     public function update(OrderStoreRequest $request, Order $order)
101     {
102         //
103
104         return redirect()->route('orders.index')->with('message', 'Order gewijzigd');
105     }
```

Bij de volgende test zien we dit

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderUpdate,OrderUpdateCheck --stop-on-failure

  PASS TestsFeatureOrdersOrderUpdateCheckTest
    ✓ an order update requires a user
    ✓ an user id must be an integer

  FAIL TestsFeatureOrdersOrderUpdateTest
    ✓ 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": "2020-07-02 09:59:13"
  }.

  Found: [
    {
      "id": 1,
      "orderdate": "2020-06-22 09:59:13",
      "user_id": 1,
      "created_at": "2020-06-22 09:59:13",
      "updated_at": "2020-06-22 09:59:13"
    }
  ].
```

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.

```
100     public function update(OrderStoreRequest $request, Order $order)
101     {
102         //
103         $order->user_id = $request->user_id;
104         $order->orderdate = $request->orderdate;
105         $order->save();
106         return redirect()->route( route: 'orders.index')->with('message', 'Order gewijzigd');
107     }
```

We zijn dan ook meteen klaar met de update, want alle testen gaan nu goed

```
D:\laragon\www\workshop (master)
\ .\vendor\bin\pest --group=OrderUpdate,OrderUpdateCheck --stop-on-failure

  PASS  TestsFeatureOrdersOrderUpdateCheckTest
✓ an order update requires a user
✓ an user id must be an integer

  PASS  TestsFeatureOrdersOrderUpdateTest
✓ 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:  6 passed
Time:  4.26s
```

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. De OrderDeleteTest ziet er dan zo uit.

```
3   use App\User;
4   use App\Order;
5   use Carbon\Carbon;
6
7   beforeEach(function () {
8       $this->seed('RoleAndPermissionSeeder');
9       $this->seed('TestingUserSeeder');
10      $this->order = factory( class: Order::class)->create();
11  });
12
13  test( description: 'guest can not see the order delete page', function(){
14      $this->get(route( name: 'orders.delete', ['order' => $this->order->id]))
15          ->assertRedirect(route( name: 'login'));
16  })->group( ...groups: 'OrderDelete');
```

```

18  test( description: 'customer can not see the order delete page', function(){
19      $customer = User::find(1);
20
21      actingAs($customer)
22          ->get(route( name: 'orders.delete', ['order' => $this->order->id]))
23          ->assertForbidden();
24  })->group( ...groups: 'OrderDelete');
25
26  test( description: 'sales can not see the order delete page', function(){
27      $sales = User::find(2);
28
29      actingAs($sales)
30          ->get(route( name: 'orders.delete', ['order' => $this->order->id]))
31          ->assertForbidden();
32  })->group( ...groups: 'OrderDelete');
33
34  test( description: 'admin can see the order delete page', function(){
35      $admin = User::find(3);
36      $this->withoutExceptionHandling();
37
38      actingAs($admin)
39          ->get(route( name: 'orders.delete', ['order' => $this->order->id]))
40          ->assertViewIs( value: 'admin.orders.delete')
41          ->assertSee($this->order->user->name)
42          ->assertSee(Carbon::createFromFormat( format: 'Y-m-d H:i:s', $this->order->orderdate)
43              |      ->format( format: 'Y-m-d\TH:i')));
44  })->group( ...groups: 'OrderDelete');

```

Dus de sales en customer krijgen dezelfde foutmelding. De admin ziet de delete pagina. Als we de test nu gaan uitvoeren krijgen we dit

```

D:\laragon\www\workshop (master)
\ vendor\bin\pest --group=OrderDelete --stop-on-failure

FAIL TestsFeatureOrdersOrderDeleteTest
✓ 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

---
• TestsFeatureOrdersOrderDeleteTest > 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.

```
109      /**
110      * Show the form for deleting the specified resource.
111      *
112      * @param \App\Order $order
113      * @return \Illuminate\Http\Response
114      */
115     public function delete(Order $order)
116     {
117     }
118 }
```

Als we testen krijgen we dit

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderDelete --stop-on-failure

FAIL TestsFeatureOrdersOrderDeleteTest
✓ 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

---
• TestsFeatureOrdersOrderDeleteTest > 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.

```
115     public function delete(Order $order)
116     {
117         return view('admin.orders.delete');
118     }
```

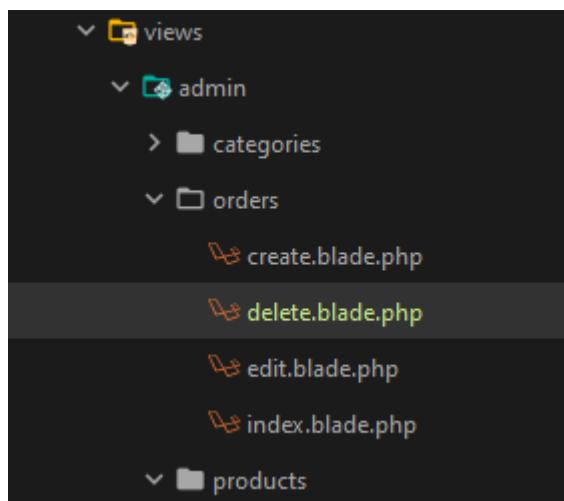
Daarna testen we weer.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderDelete --stop-on-failure

FAIL TestsFeatureOrdersOrderDeleteTest
✓ 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

---
• TestsFeatureOrdersOrderDeleteTest > 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:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderDelete --stop-on-failure

FAIL TestsFeatureOrdersOrderDeleteTest
✓ 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

---
• TestsFeatureOrdersOrderDeleteTest > admin can see the order delete page
  Failed asserting that '' contains "Sales".
```

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.

Als eerst het bovenstuk van de delete pagina

```
1  ⚡ @extends('layouts.layout')
2
3  @section('content')
4      <h1 class="mt-5">Orders</h1>
5
6  @if ($errors->any())
7      <div class="alert alert-danger">
8          <ul>
9              @foreach ($errors->all() as $error)
10                  <li>{{ $error }}</li>
11          @endforeach
12      </ul>
13  </div>
14 @endif
15
16 <nav class="nav">
17     <ul class="nav nav-tabs">
18         <li class="nav-item">
19             <a class="nav-link" href="{{ route('orders.index') }}">Index</a>
20         </li>
21         <li class="nav-item">
22             <a class="nav-link" href="{{ route('orders.create') }}">Create</a>
23         </li>
24         <li class="nav-item">
25             <a class="nav-link active" href="{{ route('orders.delete', [
26                 'order' => $order->id]) }}">Delete Order</a>
27         </li>
28     </ul>
29 </nav>
```

Daarna het formulier zelf

```
31      <form method="POST" action="{{ route('orders.destroy', ['order' => $order->id]) }}">
32          @method('DELETE')
33          @csrf
34          <div class="form-group">
35              <label for="name">Orderdate</label>
36              <input type="datetime-local" name="orderdate" class="form-control" id="orderdate"
37                  aria-describedby="orderdateHelp" placeholder="Kies Order datum"
38                  value="{{old('orderdate', $order->orderdate)}}"
39                  disabled="disabled">
40          </div>
41          <div class="form-group">
42              <label for="user_id">User</label>
43              <select name="user_id" id="user_id" class="form-control" disabled="disabled">
44                  @foreach($users as $user)
45                      <option value="{{ $user->id }}"
46                          @if( old('user_id', $order->user_id) == $user->id)
47                          selected
48                      @endif
49                      >{{ $user->name }}</option>
50                  @endforeach
51          </select>
52          <button type="Submit" class="btn btn-primary">Submit</button>
53      </form>
54  @endsection
```

Let op, dat je de route goed zet en dat de inputs disabled zijn. Als we nu testen krijgen we dit.

```
D:\laragon\www\workshop (master)
\ 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:\laragon\www\workshop\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

```
115     public function delete(Order $order)
116     {
117         $users = User::all();
118         $order->orderdate = Carbon::createFromFormat( format: 'Y-m-d H:i:s', $order->orderdate)
119             ->format( format: 'Y-m-d\TH:i');
120         return view( view: 'admin.orders.delete', compact( varname: 'order', _: 'users'));
121     }
```

Nu staat alles zoals we willen. De gehele test werkt nu.

```
D:\laragon\www\workshop (master)
\ 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: 2.90s
```

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.

```
3   use App\User;
4   use App\Order;
5   use Carbon\Carbon;
6
7   beforeEach(function () {
8       $this->seed('RoleAndPermissionSeeder');
9       $this->seed('TestingUserSeeder');
10      $this->order = factory( class: Order::class)->create();
11  });
12
13  test( description: 'guest can not delete an order', function(){
14      $this->json('DELETE', route( name: 'orders.destroy', ['order' => $this->order->id]))
15          ->assertStatus(401);
16  })->group( ...groups: 'OrderDestroy');
```

We verwachten een 401 bij de guest.

De customer en sales mogen niks verwijderen en krijgen een forbidden. De admin mag de order wel verwijderen. We controleren dan ook of de order niet meer in de database staat.

```
18  ⌂ test( description: 'customer can not delete an order', function(){
19      $customer = User::find(1);
20      actingAs($customer);
21      $this->json('DELETE', route( name: 'orders.destroy', ['order' => $this->order->id]))
22          ->assertForbidden();
23  })->group( ...groups: 'OrderDestroy');
24
25  ⌂ test( description: 'sales can not delete an order', function(){
26      $sales = User::find(2);
27      actingAs($sales);
28      $this->json('DELETE', route( name: 'orders.destroy', ['order' => $this->order->id]))
29          ->assertForbidden();
30  })->group( ...groups: 'OrderDestroy');
31
32  ⌂ test( description: 'admin can delete an order', function(){
33      $admin = User::find(3);
34      actingAs($admin);
35      $this->json('DELETE', route( name: 'orders.destroy', ['order' => $this->order->id]));
36      $this->assertDatabaseMissing('orders', ['id' => $this->order->id]);
37  })->group( ...groups: 'OrderDestroy');
```

Als we gaan testen werken de testen bijna. Alleen de admin werkt nog niet.

```
D:\laragon\www\workshop (master)
\ vendor\bin\pest --group=OrderDestroy --stop-on-failure

 FAIL TestsFeatureOrdersOrderDestroyTest
✓ guest can not delete an order
✓ customer can not delete an order
✓ sales can not delete an order
• admin can delete an order

---
• TestsFeatureOrdersOrderDestroyTest > admin can delete an order
Failed asserting that a row in the table [orders] does not match the attributes {
    "id": 1
}.

Found similar results: [
    {
        "id": 1,
        "orderdate": "2020-06-22 10:42:45",
        "user_id": 3,
        "created_at": "2020-06-22 10:42:45",
        "updated_at": "2020-06-22 10:42:45"
    }
].
```

We hebben ook nog niks in de destroy staan, dus de order wordt niet verwijderd.

```
129     public function destroy(Order $order)
130     {
131         //
132         $order->delete();
133         return redirect()->route('orders.index')->with('message', 'Order deleted');
134     }
```

Nu werkt de test al geheel

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --group=OrderDestroy --stop-on-failure

  PASS  Tests\Feature\Orders\OrderDestroyTest
    ✓ guest can not delete an order
    ✓ customer can not delete an order
    ✓ sales can not delete an order
    ✓ admin can delete an order

Tests:  4 passed
Time:  2.91s
```

Als we voor de eindcontrole nog is alle testen uitvoeren zien we dat alles nu goed door de test komt.

```
D:\laragon\www\workshop (master)
λ .\vendor\bin\pest --stop-on-failure

  PASS  Tests\Feature\ProductStoreCheckTest
    ✓ guest can not check product
    ✓ customer can not check product
    ✓ sales can not check product
    ✓ admin can check product

Tests:  88 passed
Time:  62.55s

D:\laragon\www\workshop (master)
```

De Order is nu tot zover klaar en dus ook het stuk TDD in deze lessenserie.

Checklist

Omgeving

- Laragon installeren
- Nieuwe PHP-versie downloaden (<https://windows.php.net/download#php-7.4>)
- Gedownloade php zip uitpakken en in “laragon/bin/php” zetten
- In Laragon naar menu->tools->quickadd->phpMyAdmin, zodat phpMyAdmin actief is.
- PhpMyAdmin nieuwe versie (4.9.5) downloaden (<https://www.phpmyadmin.net/downloads/>)
- Gedownloade PhpMyAdmin uitpakken en in “laragon/etc/aps/phpMyAdmin” zetten

Installatie

- In Laragon, ga naar menu->quickapp->laravel (vul je projectnaam in, wat dan ook de map wordt)
- In Laragon, naar menu->switch document root : kies de public map van het project. (Dit zodat je straks localhost/ als website kan gebruiken)
- In PhpStorm, check de .env bestand in de root directory. Verander de database gegevens indien nodig. (Als je meerdere projecten met Laravel hebt, is een andere databasenaam handig)

Permissions

- Verkrijgen van permissions: composer require spatie/laravel-permission
- In config/app.php in provider zetten: Spatie\Permission\PermissionServiceProvider::class
- Publiceren van permissions: php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider"
- In model User: zet “use HasRoles;”
- Zet spatie middleware in kernel : <https://docs.spatie.be/laravel-permission/v3/basic-usage/middleware/>
- Maak de permission seeder aan: php artisan make:seeder RolesAndPermissionsTableSeeder
- Maak rollen en permissies: <https://docs.spatie.be/laravel-permission/v3/advanced-usage/seeding/>
- Voeg testusers toe en geef ze permissies: <https://docs.spatie.be/laravel-permission/v3/basic-usage/new-app/>
- Maak de user seeder aan: php artisan make:seeder UsersTableSeeder
- Maak random users aan in UsersTableSeeder: <https://laravel.com/docs/7.x/seeding#using-model-factories>
- Voeg de UsersTableSeeder en RolesAndPermissionsTableSeeder toe aan de DatabaseSeeder
Let op volgorde: eerst roles, dan users, dan de rest. (Bij de rest ook letten op dat de primary keys al gemaakt zijn als ze gebruikt worden als foreign key bij een andere tabel)
- Tabellen werkelijk maken met: php artisan migrate --seed (Als je al een migrate hebt gedaan, kan je dit gebruiken: php artisan migrate:refresh --seed)

Data (voorbeeld met een basis webshop)

Categorie

- Aanmaken van model, migration, factory en resource controller voor categories: php artisan make:model --all Categorie
- Invullen van categorie migration. Alle attributen van de tabel neerzetten tussen \$table->id(); en \$table->timestamps();
<https://laravel.com/docs/7.x/migrations#migration-structure>
- Invullen van categorie factory.
<https://laravel.com/docs/7.x/database-testing#writing-factories>
Er wordt gebruik gemaakt van Faker om data te genereren. (<https://github.com/fzaninotto/Faker>)
- Invullen van categorie seeder.
- In DatabaseSeeder de categorie seeder toevoegen

Product

- Aanmaken van model, migration, factory en resource controller voor products: php artisan make:model --all Product
- Invullen van product migration. De foreign key kolom als van **categorie** niet vergeten, met de onDelete en onUpdate: <https://laravel.com/docs/7.x/migrations#foreign-key-constraints>
- Invullen van de product factory. Niet de categorie invullen, gaat straks via seeder.
- Aanmaken van relaties in model. Let op dat een relatie 2 kanten op gaat, dus beide models updaten.
in Categorie: <https://laravel.com/docs/7.x/eloquent-relationships#one-to-many>
In Product: <https://laravel.com/docs/7.x/eloquent-relationships#one-to-many-inverse>
- Aanvullen van de categorie Seeder. Voor iedere categorie nu bijv 30 producten.
->each(function(\$categorie){
 {
 \$categorie->product()->saveMany(factory(Product::class, 30)
 ->create(['categorie_id' => \$categorie->id]));
 });

Price

- Aanmaken van model, migration, factory en resource controller voor prices: php artisan make:model --all Price
- Invullen van price migration. De foreign key kolom als van **product** niet vergeten, met de onDelete en onUpdate: <https://laravel.com/docs/7.x/migrations#foreign-key-constraints>
- Invullen van de price factory. De product_id gaan we invullen met een random product.
'product_id' => Product::all()->random()->id
Vergeet niet de model bij use te zetten: use App\Product
- Aanmaken van relaties in model.
in Product: <https://laravel.com/docs/7.x/eloquent-relationships#one-to-many>
In Price: <https://laravel.com/docs/7.x/eloquent-relationships#one-to-many-inverse>
- Aanvullen van de price Seeder.
factory(App\Price::class, 500)->create();
Maak genoeg prices aan voor de producten
- In DatabaseSeeder de price seeder toevoegen

Order

- Aanmaken van model, migration, factory en resource controller voor orders: php artisan make:model --all Order
- Invullen van order migration. De foreign key kolom als van **user** niet vergeten, met de onDelete en onUpdate: <https://laravel.com/docs/7.x/migrations#foreign-key-constraints>
- Invullen van de order factory. De user_id gaan we invullen met een random user.
'user_id' => User::all()->random()->id
Vergeet niet de model bij use te zetten: use App\User
- Aanmaken van relaties in model.
in User: <https://laravel.com/docs/7.x/eloquent-relationships#one-to-many>
In Order: <https://laravel.com/docs/7.x/eloquent-relationships#one-to-many-inverse>
- Aanvullen van de Order Seeder.
factory(App\Order::class, 50)->create();
- In DatabaseSeeder de order seeder toevoegen

Orderrow

- Aanmaken van model, migration, factory en resource controller voor orders: php artisan make:model --all Orderrow
- Invullen van order migration. De foreign key kolommen als van **order en product** niet vergeten, met de onDelete en onUpdate: <https://laravel.com/docs/7.x/migrations#foreign-key-constraints>
- De OrderrowFactory moet er wel zijn, maar mag je verder leeg laten. Via de seeder gaan we deze straks regelen
- Aanmaken van relaties in model.
in Order en Product: <https://laravel.com/docs/7.x/eloquent-relationships#one-to-many>
In Orderrow: <https://laravel.com/docs/7.x/eloquent-relationships#one-to-many-inverse>
- Aanvullen van de Order Seeder. (we gebruiken dus niet een orderrowseeder!)

```
factory(App\Order::class, 50)->create()
->each(function($order)
{
    $order->orderrow()->saveMany(factory(Orderrow::class, 5)
        ->create(['order_id' => $order->id,
        'product_id' => Product::all()->random()->id]));
});
```

Bij de order maken we dus eerst een order aan, we maken per order 5 orderrows aan en vullen de foreign keys.
Let op dat je de models erbij zet: use App\Product; use App\Orderrow;

Reviews

- Aanmaken van model, migration, factory en resource controller voor reviews: php artisan make:model --all Review
- Invullen van review migration. De foreign key kolommen als van **user en product** niet vergeten, met de onDelete en onUpdate: <https://laravel.com/docs/7.x/migrations#foreign-key-constraints>
- De ReviewFactory invullen. Je kan de foreign keys product_id en user_id gewoon random invullen.
'product_id' => App\Product::all()->random()->id,
'user_id' => App\User::all()->random()->id
- Aanmaken van relaties in model.
in User en Product: <https://laravel.com/docs/7.x/eloquent-relationships#one-to-many>
In Review: <https://laravel.com/docs/7.x/eloquent-relationships#one-to-many-inverse>
- Aanvullen van de review Seeder.
factory(App\Review::class, 150)->create()
- In DatabaseSeeder de review seeder toevoegen

Tips & Tricks

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.

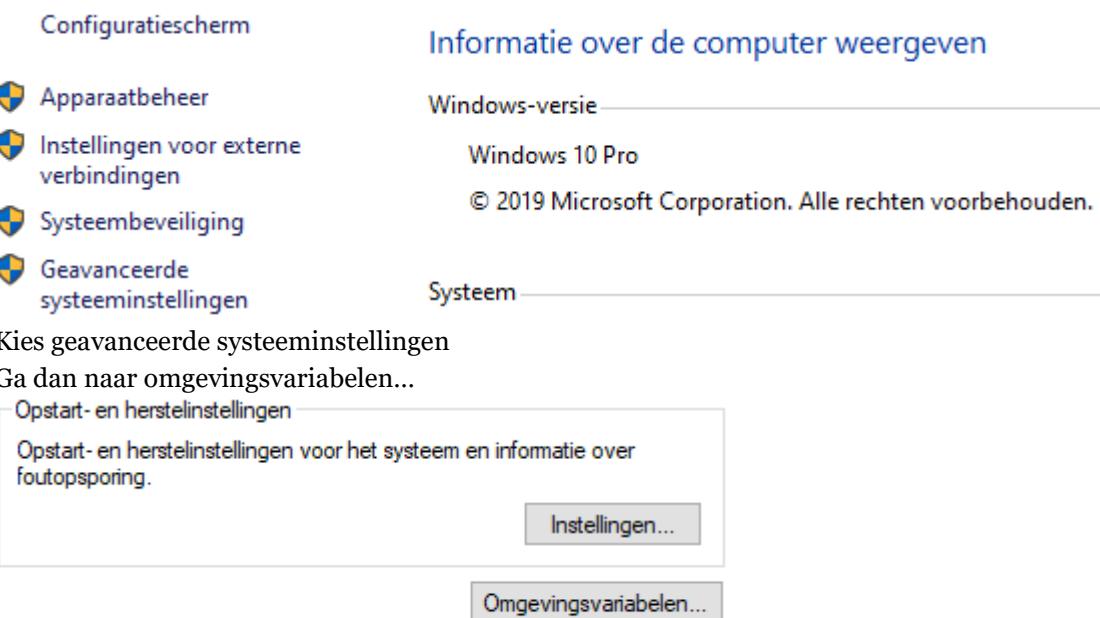
```
D:\laragon\www\workshop (master)
λ php -v
PHP 7.4.5 (cli) (built: Apr 14 2020 16:17:34) ( ZTS Visual C++ 2017 x64 )
Copyright (c) The PHP Group
Zend Engine v3.4.0, Copyright (c) Zend Technologies
```

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:

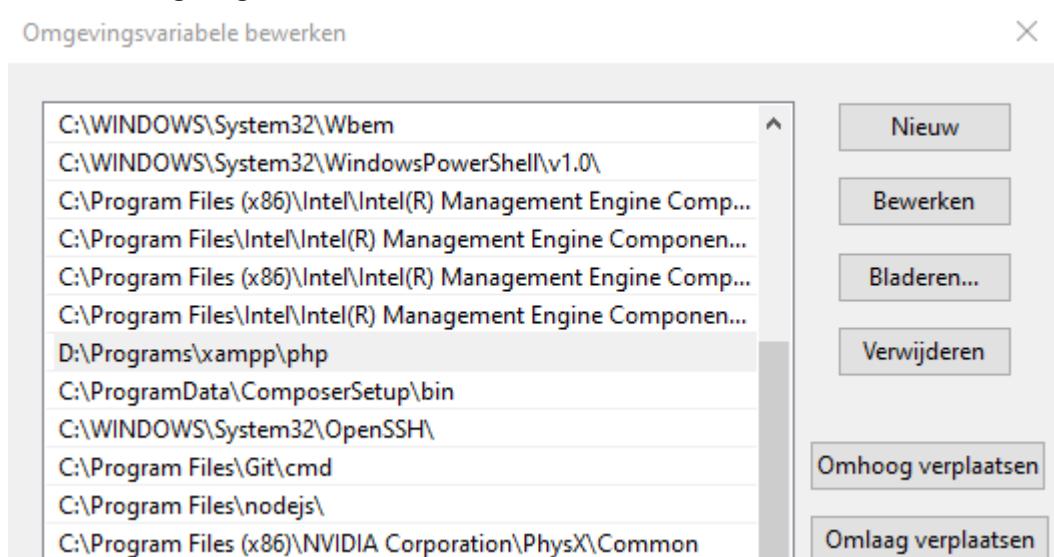


- **Let op, hier moet je voorzichtig zijn. Als je het verkeerd doet kan je je Windows om zeep helpen.**

- 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 |

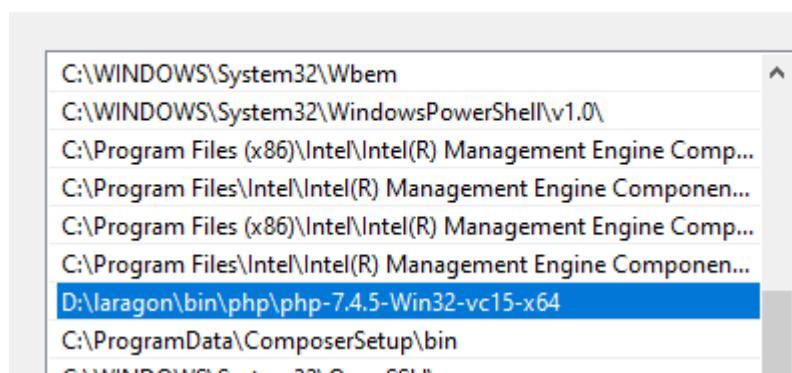
- Dubbelklik op Path
- Je ziet nu: omgevingsvariabele bewerken



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

Uitbreidingen

Per validatie een aparte message

```
// In \App\Http\Requests\SomeRequest

/**
 * Get the validation rules that apply to the request.
 *
 * @return array
 */
public function rules()
{
    return [
        'post_code' => 'required|valid_postcode',
    ];
}

public function messages()
{
    return [
        'valid_postcode' => 'Invalid postcode',
    ];
}
```

Prefix voor resource controller

23

```
Route::namespace('Admin')->prefix('admin')->name('admin.')->group(function () {
    Route::resource('users', 'UserController');
});
```



These options will result in the following for the *Resource*:

- `namespace()` sets Controller namespace to `\Admin\UserController`
- `prefix()` sets request URI to `/admin/users`
- `name()` sets route name accessor to `route('admin.users.index')`

In `name()` the DOT is intended, it is not a typo.

```
Route::resource('faq', 'ProductFaqController', [
    'as' => 'prefix'
]);
```

This will give you routes such as `prefix.faq.index`, `prefix.faq.store`, etc.

