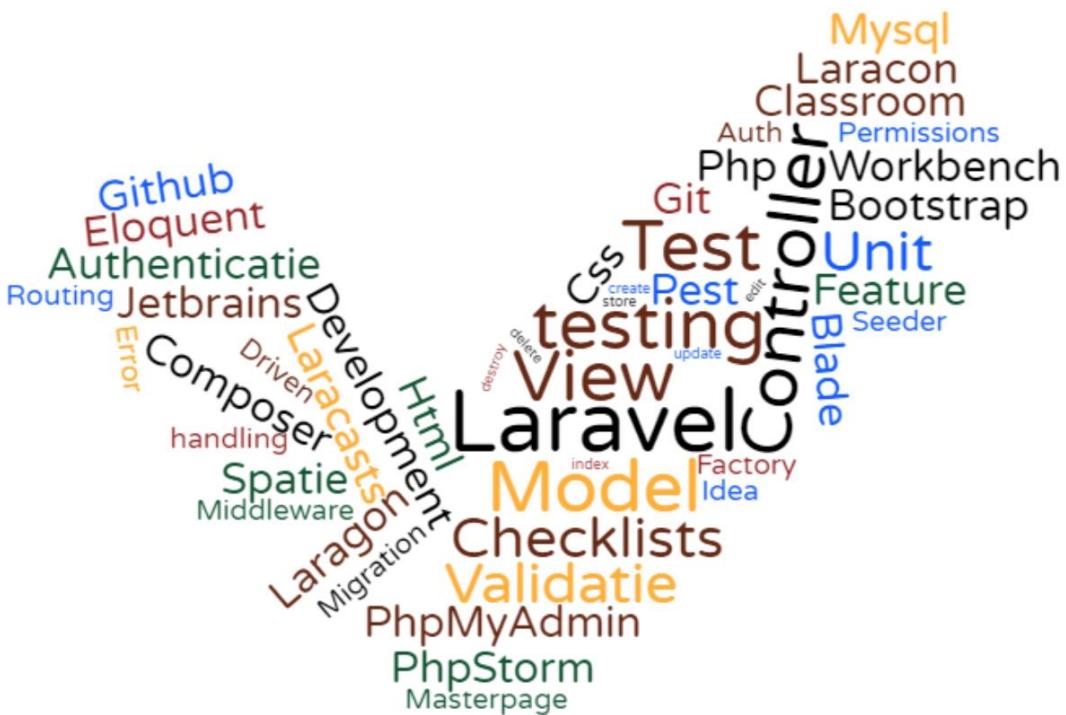


Laravel 9 Level 2



Laravel



Marcel Koningstein

Techniek College Rotterdam locatie
Spijkenisse

Inhoud

1.	Inleiding	4
1.1.	Versiebeheer	5
1.2.	Omgeving & installatie	6
1.3.	Casus beschrijving	7
1.4.	Opdrachten	8
2.	Rollen en Permissies.....	10
2.1.	Role and Permission Seeder.....	13
2.2.	User Seeder	16
2.3.	Opdracht 10: Roles & Users	20
2.4.	Middleware gebruiken in Routes.....	21
2.5.	Authenticatie & layout	24
2.6.	Permissies in de Controller	34
2.7.	Opdracht 11: Permissions	37
3.	Product & Price.....	38
3.1.	Models	39
3.2.	Migrations	42
3.3.	Relaties in models	44
3.4.	Opdracht 12: Models en Migrations	47
3.5.	Factory	48
3.6.	Seed.....	50
3.7.	Opdracht 13: Factory	55
3.8.	Opdracht 14: Seeder	56
3.9.	Product Controller.....	57
3.9.1.	Index	57
3.9.2.	Uitbreiden van de index	61
3.9.3.	Opdracht 15: Index van Tasks	66
3.9.4.	Permissies voor product controller	67
3.9.5.	Opdracht 16: Permissies voor tasks	68
3.9.6.	Create	69
3.9.7.	Store	73
3.9.8.	Opdracht 17: Create & store voor tasks met validatie.....	78
3.9.9.	Show	79
3.9.10.	Opdracht 18: Show voor task	81

3.9.11. Edit.....	82
3.9.12. Update	84
3.9.13. Opdracht 19: Edit & update voor tasks	88
3.9.15. Delete	89
3.9.16. Destroy	91
3.9.17. Opdracht 20: Delete & destroy voor tasks	92
3.9.18. Category Test.....	93

1. Inleiding

Dit is een lessenserie om stap voor stap de basis van MVC te leren door middel van het framework Laravel. Bij het schrijven van deze lessenserie is de huidige versie van Laravel 9. In de lessenserie leer je hoe je de omgeving kan opzetten, de basisvaardigheden van Laravel en hoe je een werkelijk project kan maken. De gehele serie bestaat uit meerdere levels, zodat het overzichtelijk blijft.

De levels zijn:

- Level 0: Installatie & basis uitleg van Laravel
- Level 1: Een crud zonder relatie & simpele publieke pagina
- Level 2: Authenticatie, permissions en een crud met relatie
- Level 3: Geavanceerde publieke pagina's
- Level 4: Testen met Phpunit / Pest / Dusk & Test driven development
- Level 5: Werken met API-calls in combinatie met React / Vue

Dit document is voor **level 2**

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

- De student weet wat een MVC is
- De student kan views aanpassen
- De student kan werken met Migrations, Seeds en Factory
- De student kan werken met Controllers
- De student kan werken met Models
- De student kan een database aanspreken via Laravel
- De student weet wat hidden files zijn
- De student kan werken met validatie
- De student kan een simpele website bouwen in Laravel.

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

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

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

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

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

1.1. Versiebeheer

Versie	Beschrijving	Datum	Aanpassing
0.9	Initieel document	2016 - 2022	Verschillende wijzigingen t/m Laravel 8
1.0	Aanpassing	10-2-2022	Aanpassing Laravel 8 naar Laravel 9

1.2. Omgeving & installatie

Omdat je op zoveel verschillende manieren je omgeving kan hebben, heb ik dit nu niet hier staan. In het document van level 0 kan je allerlei verschillende manieren vinden hoe je Laravel kan krijgen op verschillende omgevingen.

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

Editor:

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

Versiebeheer:

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

Andere tools:

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

Zelf heb ik gekozen voor de omgeving met:

- Wamp.net
 - Php 8.1
 - Nginx 1.21.3
 - Mariadb 10.5.9
 - PhpMyAdmin 5.1.1
- PhpStorm
 - Material Theme UI
 - Material Icons
 - Laravel-idea (<https://laravel-idea.com/>)

1.3. Casus beschrijving

Als we naar de lessenserie gaan kijken, zijn we bij level 1 begonnen met een casus. Dan is het handig om deze casus/opdracht ook in dit document te hebben. Bij deze de beschrijving.

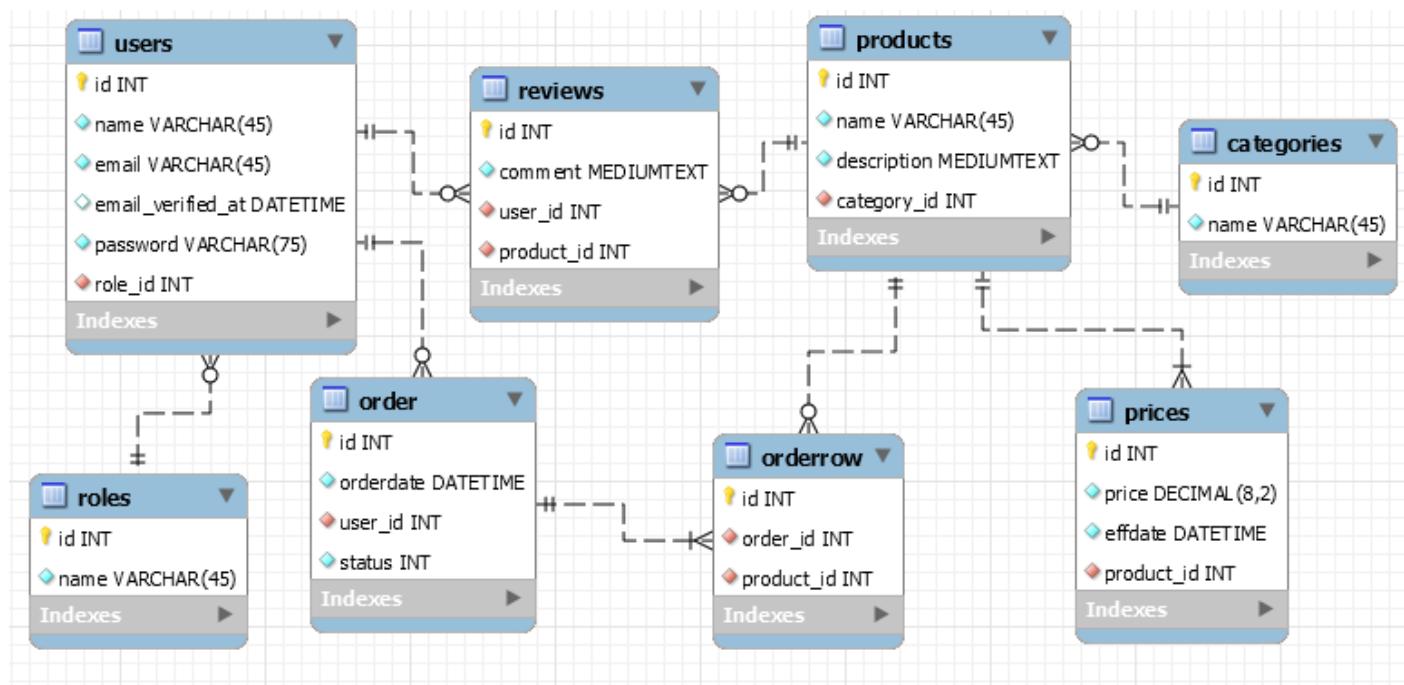
We gaan een simpele webshop maken voor spellen. Hierbij kunnen gebruikers:

- 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 gaan regelen.

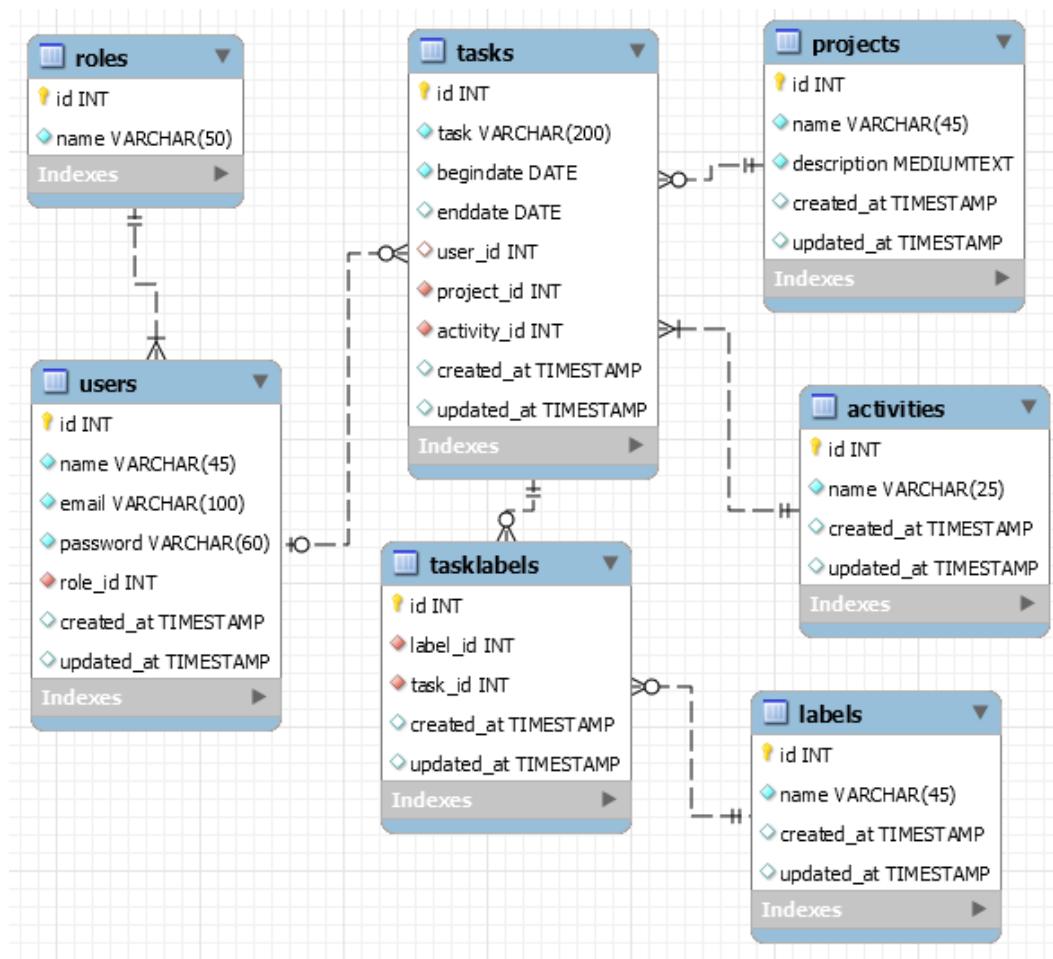
1.4. Opdrachten

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

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

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

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



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

Dit heb je als het goed is al gedaan bij het document van level 1. Mocht dit niet het geval zijn kan je vinden op:

- <https://gitlab.com/koningstein/laravel9-opdrachten>

Voor nu nog geen opdracht verder, want je zal eerst wat moeten leren over Laravel voordat je een opdracht uit kan voeren. Als je wilt beginnen met de opdrachten, check dan het hoofdstuk uit waarin uitgelegd wordt hoe je de repository kan clonen en configureren. Dit kan je vinden in het documenten van level 0.

Hierin staan de volgende stappen beschreven:

- Clone het project in een directory
- Maak in wamp.net een project aan en zet de document root op de public map. Gebruik minimaal php versie 8
- Zorg dat je bij de phpMyAdmin kan komen
- Maak 2 databases aan, 1 voor de website en 1 voor testen
 - Database 1: opdrachten
 - Database 2: opdrachtentest (Let op, gebruik je een andere naam, wijzig dit in phpunit.xml)
- Maak een .env file aan, en kopier de .env.example daar naartoe
- In de .env file, check de database naam (opdrachten) en de username (root)
- Ga naar je root directory van je project in de terminal, en voer daar uit: composer install
- Voer dan in de terminal uit: php artisan key:generate

Gebruik je een andere omgeving, dan zal je soortgelijke stappen moeten nemen om de opdrachten klaar te zetten.

2. Rollen en Permissies

De eerste crud is gemaakt in het document level. Nu zullen we wat aan de beveiliging moeten gaan doen. Op een website is het natuurlijk niet de bedoeling dat iedereen overal bij kan. Hiervoor gaan aan de slag met rollen en permissies.

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

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

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

Documentatie: <https://spatie.be/docs/laravel-permission/v5/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/developers>.

packagist v5.5.0 Run Tests passing downloads 17M

Documentation, Installation, and Usage Instructions

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

What It Does

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

Once installed you can do stuff like this:

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

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

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

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

Composer require spatie/laravel-permission

```
PS D:\wamp\sites\laravel9v1> composer require spatie/laravel-permission
Using version ^5.5 for spatie/laravel-permission
./composer.json has been updated
Running composer update spatie/laravel-permission
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
- Locking spatie/laravel-permission (5.5.0)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
- Installing spatie/laravel-permission (5.5.0): Extracting archive
Generating optimized autoload files
```

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

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

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

```
PS D:\wamp\sites\laravel9v1> 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\2022_03_01_152547_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;*

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

Als het goed is komt regel 10 er vanzelf bij, anders handmatig de regel toevoegen:

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

2.1. Role and Permission Seeder

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

```
PS D:\wamp\sites\laravel9v1> php artisan make:seeder RoleAndPermissionSeeder
Seeder created successfully.
```

Hierdoor staat de seeder klaar.

```
1  <?php
2
3  namespace Database\Seeders;
4
5  use Illuminate\Database\Console\Seeds\WithoutModelEvents;
6  use Illuminate\Database\Seeder;
7
8  class RoleAndPermissionSeeder extends Seeder
9  {
10     /**
11      * Run the database seeds. ...
12     */
13     public function run()
14     {
15         //
16     }
17 }
18 }
19 }
```

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

```
use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use Spatie\Permission\Models\Role;
use Spatie\Permission\Models\Permission;

class RoleAndPermissionSeeder extends Seeder
{
    /**
     * Run the database seeds. ...
     */
    public function run()
    {
        app()[\Spatie\Permission\PermissionRegistrar::class]→forgetCachedPermissions();
    }
}
```

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

Nu gaan we hieronder de permissies toevoegen. Dit zijn allemaal mogelijke permissies voor onze functionaliteit. We hebben op dit moment de crud voor categorie die we gemaakt hebben in het document van level 1. Zodra we meer functionaliteit maken gaan we hiervoor ook de permissies toevoegen.

```
class RoleAndPermissionSeeder extends Seeder
{
    /**
     * Run the database seeds. ...
     */
    public function run()
    {
        app()[\Spatie\Permission\PermissionRegistrar::class]→forgetCachedPermissions();

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

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. Als je goed kijkt zie je geen permissie voor bijvoorbeeld store, update of destroy, terwijl dit wel methodes zijn in een crud. Bij de create hoort natuurlijk de store, bij de edit hoort de update en bij de delete hoort de destroy.

Om straks dit niet voor elke persoon te hoeven doen, gaan we Rollen aanmaken. De rollen krijgen dan rechten over bepaalde permissies.

```
class RoleAndPermissionSeeder extends Seeder
{
    /**
     * Run the database seeds. ...
     */
    public function run()
    {
        app()[\Spatie\Permission\PermissionRegistrar::class]→forgetCachedPermissions();

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

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

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

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

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

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

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

De rollen en permissies zijn nu klaar.

2.2. User Seeder

We gaan aan de slag met de users. Doordat we bij level 1 al Laravel Breeze hebben geïnstalleerd, hebben we echt van alles al voor authenticatie. Onder andere een model, migration, factory en verschillende controllers. Wat we nog niet hebben is een UserSeeder. Om ervoor te zorgen dat een user een rol gaat krijgen, zullen we de [UserSeeder](#) aan gaan maken

```
PS D:\wamp\sites\laravel9v1> php artisan make:seeder UserSeeder
Seeder created successfully.
```

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

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

```
3  namespace Database\Seeders;
4
5  use App\Models\User;
6  use Illuminate\Database\Console\Seeds\WithoutModelEvents;
7  use Illuminate\Database\Seeder;
8  use Illuminate\Support\Facades\Hash;
9
10 class UserSeeder extends Seeder
11 {
```

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

```
public function definition()
{
    return [
        'name' => $this->faker->name,
        'email' => $this->faker->unique()->safeEmail,
        'email_verified_at' => now(),
        'password' => '$2y$10$92IXUNpkj00r0Q5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi', // password
        'remember_token' => Str::random( length: 10),
    ];
}
```

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

De UserSeeder wordt dan dit:

```
10 class UserSeeder extends Seeder
11 {
12     /**
13      * Run the database seeds. ...
14     */
15     public function run()
16     {
17         // create customer
18         $customer = User::factory()->create([
19             'name' => 'Customer',
20             'email' => 'customer@tcrmbo.nl',
21             'password' => Hash::make( value: 'test1234')
22         ]);
23         $customer->assignRole('customer');
24
25         // create sales account
26         $sales = User::factory()->create([
27             'name' => 'Sales',
28             'email' => 'sales@tcrmbo.nl',
29             'password' => Hash::make( value: 'test1234')
30         ]);
31         $sales->assignRole('sales');
32
33         // create admin
34         $admin = User::factory()->create([
35             'name' => 'Admin',
36             'email' => 'admin@tcrmbo.nl',
37             'password' => Hash::make( value: 'test1234')
38         ]);
39         $admin->assignRole('admin');
40
41     }
42 }
43 }
```

Je ziet hier, dat ik voor elke user de standaard factory overschreven wordt voor de velden name, email en password. Deze zijn herkenbaar voor de rollen die ze krijgen, waardoor het straks goed uit te testen is.

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()
    {
        // \App\Models\User::factory(10)→create();
        $this→call([
            RoleAndPermissionSeeder::class,
            UserSeeder::class,
            CategorySeeder::class
        ]);
    }
}
```

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

Om ervoor te zorgen dat we weer een schone database krijgen doe ik eerst een `migrate:fresh --seed`.

Ook als je niet naar een vorige situatie hoeft, maar gewoon een schone database is het handig. We willen nu een schone database, waardoor we dit commando gaan gebruiken: `php artisan migrate:fresh --seed`

```
PS D:\wamp\sites\laravel9v1> 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 (36.01ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (32.42ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (24.83ms)
Migrating: 2019_12_14_000001_create_personal_access_tokens_table
Migrated: 2019_12_14_000001_create_personal_access_tokens_table (29.98ms)
Migrating: 2022_02_11_082343_create_categories_table
Migrated: 2022_02_11_082343_create_categories_table (9.96ms)
Migrating: 2022_02_11_093836_update_name_length_categories_table
Migrated: 2022_02_11_093836_update_name_length_categories_table (123.98ms)
Migrating: 2022_03_01_152547_create_permission_tables
Migrated: 2022_03_01_152547_create_permission_tables (407.10ms)
Seeding: Database\Seeders\RoleAndPermissionSeeder
Seeded: Database\Seeders\RoleAndPermissionSeeder (170.85ms)
Seeding: Database\Seeders\UserSeeder
Seeded: Database\Seeders\UserSeeder (389.71ms)
Seeding: Database\Seeders\CategorySeeder
Seeded: Database\Seeders\CategorySeeder (51.68ms)
Database seeding completed successfully.
```

In de database hebben we nu wel veel meer tabellen gekregen

Tabel	Actie	Rijen
categories	Verkennen Structuur Zoeken Invoegen Legen Verwijderen	25
failed_jobs	Verkennen Structuur Zoeken Invoegen Legen Verwijderen	0
migrations	Verkennen Structuur Zoeken Invoegen Legen Verwijderen	7
model_has_permissions	Verkennen Structuur Zoeken Invoegen Legen Verwijderen	0
model_has_roles	Verkennen Structuur Zoeken Invoegen Legen Verwijderen	3
password_resets	Verkennen Structuur Zoeken Invoegen Legen Verwijderen	0
permissions	Verkennen Structuur Zoeken Invoegen Legen Verwijderen	5
personal_access_tokens	Verkennen Structuur Zoeken Invoegen Legen Verwijderen	0
roles	Verkennen Structuur Zoeken Invoegen Legen Verwijderen	3
role_has_permissions	Verkennen Structuur Zoeken Invoegen Legen Verwijderen	9
users	Verkennen Structuur Zoeken Invoegen Legen Verwijderen	3
11 tabellen	Som	55

De RoleAndPermission maakt 5 tabellen: model_has_permissions, model_has_roles, permissions, roles, role_has_permissions. Als we naar de rollen kijken staan deze netjes in de tabel.

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

Ook de 3 gebruikers staan netjes in de database.

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

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

2.3. Opdracht 10: Roles & Users

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

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

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

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

id	name	email	Password
1	student	student@tcrmbo.nl	student
2	teacher	teacher@tcrmbo.nl	teacher
3	admin	admin@tcrmbo.nl	admin

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

Opdracht controle

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

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

2.4. Middleware gebruiken in Routes

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

In de documentatie is dit de introductie van middleware:

Middleware provide a convenient mechanism for inspecting and filtering HTTP requests entering your application. For example, Laravel includes a middleware that verifies the user of your application is authenticated. If the user is not authenticated, the middleware will redirect the user to your application's login screen. However, if the user is authenticated, the middleware will allow the request to proceed further into the application.

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 role & permission aan onze kernel moeten toevoegen. De kernel vinden we in `app/Http/Kernel.php`

We gaan hier de volgende onderdelen toevoegen:

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

We voegen deze toe aan protected \$routeMiddleware

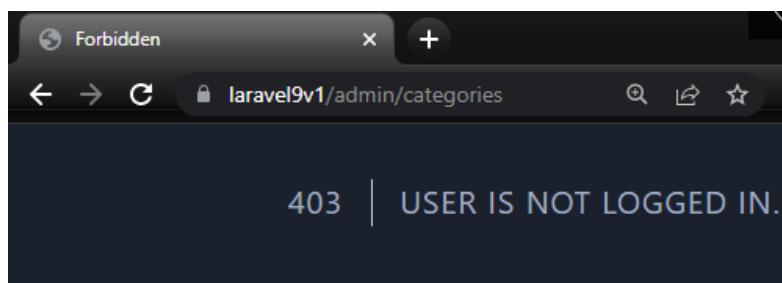
```
/** The application's route middleware. ... */  
protected $routeMiddleware = [  
    'auth' => \App\Http\Middleware\Authenticate::class,  
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,  
    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,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

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

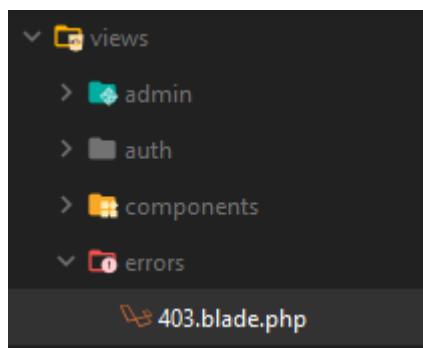
```
Route::group(['middleware' => ['role:sales|admin']], function () {
    Route::get('admin/categories/{category}/delete', [Admin\CategoryController::class, 'delete'])
        ->name('categories.delete');
    Route::resource('admin/categories', Admin\CategoryController::class);
});
```

Dit zorgt ervoor dat als we in de browser naar admin/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 sales of admin moet hebben om deze routes te zien.

Als we dit uitproberen zullen we als we ingelogd zijn als sales of admin 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 lay-out 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. De `$exception` is binnen Laravel beschikbaar om fouten te laten zien.

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

@section('content')
    <div class="card mt-6">
        
        <div class="card-header flex flex-row justify-between">
            <h1 class="h6">Error</h1>
        </div>
        

        
        <div class="card-body grid grid-cols-1 gap-6 lg:grid-cols-1">
            <div class="p-4">
                <div class="alert alert-error">
                    {{ $exception→getMessage() }}
                </div>
            </div>
        </div>
        
    </div>
@endsection
```

Als we nu opnieuw gaan kijken naar de categories, zien we de foutmelding binnen de eigen lay-out.

The screenshot shows a web browser window with the URL `laravel9v1/admin/categories`. The page has a dark header with navigation icons. The main content area has a sidebar on the left with sections for **ADMIN** (containing [Link1](#) and [Link2](#)) and **PUBLIC** (containing [Link1](#) and [Link2](#)). The main content area displays an **Error** message in a white box, with a pink callout box below it stating **User Is Not Logged In.**.

Voorlopig houden we even deze standaard foutmeldingen. Later kunnen we nog kijken om eigen foutmeldingen te schrijven.

2.5. Authenticatie & lay-out

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 'laravel9v1/login'. The page displays a login form. At the top is a logo consisting of three interlocking 3D cubes. Below the logo are two input fields: 'Email' and 'Password', each with a small circular icon to its right. Underneath these fields is a checkbox labeled 'Remember me'. At the bottom of the form are two buttons: a blue link labeled 'Forgot your password?' and a dark blue button labeled 'LOG IN'.

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

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

The screenshot shows a browser window with the URL 'laravel9v1/admin/categories'. The page title is 'Cleopatra'. On the left, there's a sidebar with sections 'ADMIN' and 'PUBLIC', each containing 'Link1' and 'Link2'. The main content area has a large 'Error' heading. Below it is a red rectangular box containing the text 'User Is Not Logged In.'

Wat wel mooi is, is dat rechts 'M.Koningstein' staat. In de lay-out die ik heb gebruikt stond namelijk een plek voor de gebruikersnaam. We gaan zorgen dat daar de naam komt als je bent ingelogd. Verder zie je in de drop down

onderaan een logout staan. Deze gaat nog nergens naar toe, maar kunnen we wel mooi gebruiken. Je ziet ook dat sommige plaatjes niet goed staan. Dit komt omdat de plek naar de plaatjes statisch in de lay-out zijn gezet, terwijl we niet meer in de root map ('/') zitten maar in '/ admin/categories'.

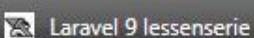
We gaan dus eerst de master lay-out veranderen, waar we alle onderdelen even gaan bekijken die nu nog niet goed staan, of waar we iets met authenticatie kunnen doen. We beginnen al bij de head. Hier zien we dat een fav.png in de map img moet staan. Dit is een statisch gedaan, want als we in /admin/categories zitten, dan zou de fav.png in /admin/categories/img/ moeten staan. Doordat we dus de masterpage gebruiken met allerlei URL's moeten we deze verwijzingen dynamisch moeten maken.

```
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="shortcut icon" href="./img/fav.png" type="image/x-icon">
  <link rel="stylesheet" href="https://kit-pro.fontawesome.com/releases/v5.12.1/css/pro.min.css">
  <link href="{{ asset('css/style.css') }}" rel="stylesheet" type="text/css">
```

Nu is het zo dat asset() naar de public map verwijst. Als daar dus een map img staat, kunnen we makkelijk alle plaatjes dynamisch gebruiken.

```
<link rel="shortcut icon" href="{{ asset('img/fav.png') }}" type="image/x-icon">
```

Als er dus een fav.png in die map zit, wat bij mij het geval is, zou je dus meteen wat moeten zien als je de pagina refreshed. Dit is dan ook het geval:



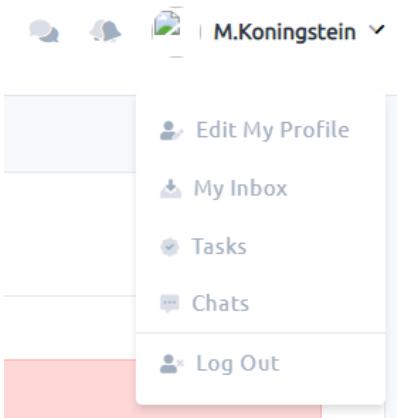
Het volgende stuk is weer een plaatje:

```
<!-- logo -->
<div class="flex-none w-56 flex flex-row items-center">
  
  <strong class="capitalize ml-1 flex-1">cleopatra</strong>
```

Wat natuurlijk op dezelfde manier wordt veranderd:

```
<!-- logo -->
<div class="flex-none w-56 flex flex-row items-center">
  
  <strong class="capitalize ml-1 flex-1">cleopatra</strong>
```

Je weet nu hoe je plaatjes moet veranderen, dus die gaan we verder niet meer benoemen. We kijken vooral naar het stuk authenticatie. Hier komen we dan een stuk tegen rechtsboven in de site.



Dit gedeelte gaat natuurlijk over een ingelogde gebruiker die dan allerlei opties heeft. Dit moet er alleen komen als je dan ook werkelijk ingelogd bent. In code begint dit dan bij de user drop down. Het plaatje is alvast veranderd.

```
52  ←— user —→
53  <div class="dropdown relative md:static">
54
55      <button class="menu-btn focus:outline-none focus:shadow-outline flex flex-wrap items-center">
56          <div class="w-8 h-8 overflow-hidden rounded-full">
57              
58          </div>
59
60          <div class="ml-2 capitalize flex ">
61              <h1 class="text-sm text-gray-800 font-semibold m-0 p-0 leading-none">M.Koningstein</h1>
62              <i class="fad fa-chevron-down ml-2 text-xs leading-none"></i>
63          </div>
64      </button>
```

Om dit alleen zichtbaar te hebben als je bent ingelogd kunnen we @guest gebruiken. In @guest zetten we dan iets wat we als niet ingelogde gebruiker willen zien, zoals guest.

```
<div class="flex flex-row-reverse items-center">
    @guest
        <div class="w-8 h-8 overflow-hidden rounded-full">
            
        </div>
        <div class="ml-2 capitalize flex pr-3">
            <h1 class="text-sm text-gray-800 font-semibold m-0 p-0 leading-none">Guest</h1>
        </div>
    @else
        @endguest
```

In de @else betekent het dat je dus geen guest bent, maar ingelogd. Dan heb je een user account en kunnen we Auth gebruiken. Met Auth kunnen we bijvoorbeeld de naam of id van de gebruiker opvragen.

```
@else

<div class="dropdown relative md:static">

    <button class="menu-btn focus:outline-none focus:shadow-outline flex flex-wrap items-center">
        <div class="w-8 h-8 overflow-hidden rounded-full">
            
        </div>

        <div class="ml-2 capitalize flex">
            <h1 class="text-sm text-gray-800 font-semibold m-0 p-0 leading-none">{{ Auth::user()→name }}</h1>
            <i class="fad fa-chevron-down ml-2 text-xs leading-none"></i>
        </div>
    </button>
```

De drop down zit nog in de @else. Nu zie je nog niet de gehele afronding, want we moeten @guest en @else nog eindigen met @endguest

Onderaan de drop down zit de log out. De log out is even wat anders dan de rest. We gaan expres het eerst even verkeerd doen, zodat je ziet wat er gebeurt.

```
<button class="hidden fixed top-0 left-0 z-10 w-full h-full menu-overflow"></button>

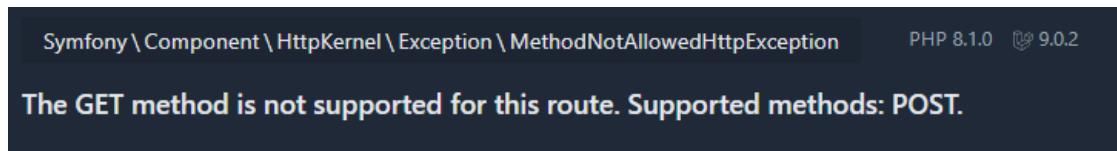
<div class="text-gray-500 menu hidden md:mt-10 md:w-full rounded bg-white shadow-md absolute z-20 right-0 top-0">
    <a class="px-4 py-2 block capitalize font-medium text-sm tracking-wide bg-white hover:bg-gray-200">
        <a class="px-4 py-2 block capitalize font-medium text-sm tracking-wide bg-white hover:bg-gray-200">
        <a class="px-4 py-2 block capitalize font-medium text-sm tracking-wide bg-white hover:bg-gray-200">
        <a class="px-4 py-2 block capitalize font-medium text-sm tracking-wide bg-white hover:bg-gray-200">
        <hr>
        <a class="px-4 py-2 block capitalize font-medium text-sm tracking-wide bg-white hover:bg-gray-200">
            <i class="fad fa-user-times text-xs mr-1"></i>
            log out
        </a>
    </div>
</div>

@endguest
```

Nu staat de log out nog in een <a href>, dus normaal zou je denken dat je dit kan gebruiken op de volgende manier:

```
<a href="{{ route('logout') }}" class="px-4 py-2 block capitalize font-medium text-sm tracking-wide bg-white hover:bg-gray-200">
    <i class="fad fa-user-times text-xs mr-1"></i>
    log out
</a>
```

Als we dit gaan gebruiken en testen krijgen we een foutmelding:



Een <a href> zorgt standaard voor een URL, dus een GET methode, terwijl we met een log out een POST nodig hebben. Nu kunnen we dit wel faken, zodat als je op het linkje klikt een formulier gepost wordt met javascript.

```
<hr>

    
    log out

<form id="logout-form" action="{{ route('logout') }}" method="POST" style="display: none;">
    @csrf
</form>
</div>
```

Met de onclick zorgen we ervoor dat de default niet uitgevoerd wordt, maar logout-form. Het formulier eronder heeft de id logout-form dus deze wordt uitgevoerd. Het formulier is zelf niet zichtbaar door display: none. Natuurlijk moeten we bij een formulier niet de @csrf vergeten.

Voor zover even de lay-out, laten we eerst even controleren of we niets zijn vergeten. Als we naar de site gaan zien we dit:

The screenshot shows a web browser window with a dark theme. The address bar says "laravel9v1/". The header includes a back/forward button, a search icon, a refresh icon, a star icon, a gear icon, a puzzle icon, and a user icon. The user is logged in as "Cleopatra". On the right, there are notification icons and a "Guest" status. The sidebar on the left has two sections: "ADMIN" which contains "Link1" and "Link2", and "PUBLIC" which also contains "Link1" and "Link2".

Een nette pagina. Je ziet dat je een guest bent. Ok, er zijn nog notificatie plaatjes aan de rechterkant, maar dat maakt op zich niet uit. Wat wel een ding is, is dat we wel moeten kunnen inloggen en registeren, zeker bij een winkel is dit handig. Deze links gaan we in het menu aan de linkerkant maken.

Deze links staan op dit moment zo in de code:

```
<p class="uppercase text-xs text-gray-600 mb-4 tracking-wider">Admin</p>

<a href=". /index.html" class="mb-3 capitalize font-medium text-sm hover:text-teal-600 transition ease-in-out duration-500">Link1</a>
<a href=". /index-1.html" class="mb-3 capitalize font-medium text-sm hover:text-teal-600 transition ease-in-out duration-500">Link2</a>


<p class="uppercase text-xs text-gray-600 mb-4 mt-4 tracking-wider">Public</p>

<a href=". /email.html" class="mb-3 capitalize font-medium text-sm hover:text-teal-600 transition ease-in-out duration-500">Link1</a>
<a href="#" class="mb-3 capitalize font-medium text-sm hover:text-teal-600 transition ease-in-out duration-500">Link2</a>

<p class="uppercase text-xs text-gray-600 mb-4 tracking-wider">Guest</p>
<a href="{{ route('login') }}" class="mb-3 capitalize font-medium text-sm hover:text-teal-600
    transition ease-in-out duration-500">{{ __('Login') }}</a>
@if(Route::has('register'))
    <a href="{{ route('register') }}" class="mb-3 capitalize font-medium text-sm hover:text-teal-600
        transition ease-in-out duration-500">{{ __('Register') }}</a>
@endif
@else
```

Hier zie je de @guest staan, wat betekent dat dit voor de niet ingelogde gebruikers is. We hebben routes naar de login en register. Deze routes bestaan al namelijk door deze regel in de web.php

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

Die geeft aan routes/auth.php ook ingeladen moet worden. In dat bestand staan alle routes voor de onderdelen van de authenticatie.

Naast de @guest zie je ook Route::has('register'). Je kan in Laravel de registratie uitzetten zodat deze optie niet gebruikt kan worden. Zo zorg je er dus voor dat de link weg is als de registratie uit staat.

Het volgende stuk wat we gaan bekijken is als je dus geen gast bent. Je bent dan dus wel ingelogd.

```
@else
@hasanyrole('sales|admin')
    ←!— links for sales & admins —→
    <p class="uppercase text-xs text-gray-600 mb-4 tracking-wider">Admin</p>
    <a href="{{ route('categories.index') }}" class="mb-3 capitalize font-medium text-sm hover:text-teal-600
        transition ease-in-out duration-500">Category Admin</a>
    <a href="#" class="mb-3 capitalize font-medium text-sm hover:text-teal-600 transition ease-in-out
        duration-500">Link2</a>
@endhasanyrole

    ←!— links for logedin users —→
@endguest
```

Omdat de admin alleen voor de rollen sales en admin gebruikt mag worden, kunnen we deze afschermen voor de andere ingelogde gebruikers met @hasanyrole. Als je maar 1 rol wilt gebruiken kan je bijvoorbeeld ook @role of @hasrole gebruiken. Dit staat allemaal in de documentatie van Laravel Permission:

<https://spatie.be/docs/laravel-permission/v5/basic-usage/blade-directives>

Er wordt in de documentatie wel verteld dat je het beste het kan afschermen met 'permission directives'. We gaan deze nog wel ergens anders gebruiken, maar omdat we deze 2 rollen hebben die erbij mogen kunnen we nu toch het best de rollen gebruiken.

Na al deze mogelijkheden hebben we ook nog links voor alle gebruikers, zoals de publieke pagina van categorie die is gemaakt. Deze kunnen we als laatst nog in het menu zetten:

```
@endguest

    ←!— links for every user —→
    <p class="uppercase text-xs text-gray-600 mb-4 mt-4 tracking-wider">Public</p>
    <a href="{{ route('open.categories.index') }}" class="mb-3 capitalize font-medium text-sm
        hover:text-teal-600 transition ease-in-out duration-500">Categories</a>
    <a href="#" class="mb-3 capitalize font-medium text-sm hover:text-teal-600 transition
        ease-in-out duration-500">Link2</a>
```

Als we naar de pagina nu kijken als gast zien we netjes de login en register link, maar ook de link naar Categories.

The screenshot shows a web browser window with the URL 'laravel9v1/'. The page has a dark header with icons for search, refresh, and other navigation. Below the header, there's a logo for 'Cleopatra' and a sidebar menu. The sidebar includes sections for 'GUEST' (Login, Register) and 'PUBLIC' (Categories, Link2). On the right side of the screen, there's a large empty area, likely a placeholder for content.

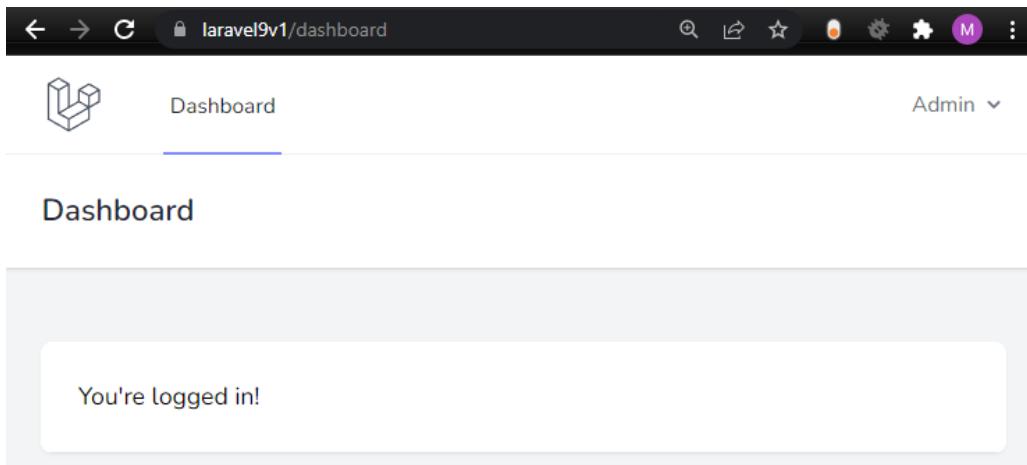
Als we naar de login gaan zien we dit:

The screenshot shows the 'laravel9v1/login' page. It features a large 'Laravel' logo at the top. Below it is a form with fields for 'Email' and 'Password', each accompanied by a file upload icon. There's also a 'Remember me' checkbox. At the bottom of the form are links for 'Forgot your password?' and a dark 'LOG IN' button.

We kunnen nu uittesten of de admin wel de juiste dingen ziet. Dus ik log netjes in met het admin account. In de UserSeeder staat dat de mail admin@tcrmbo.nl is en het wachtwoord test1234.

This screenshot shows the same login page as above, but with the 'Email' field filled with 'admin@tcrmbo.nl' and the 'Password' field filled with 'test1234'. Both fields have a light blue border, indicating they are valid inputs.

Zodra we ingelogd zijn komen we op een dashboard.



The screenshot shows a web browser window with the URL 'laravel9v1/dashboard'. The page title is 'Dashboard'. In the top right corner, there is a user menu with the text 'Admin'. Below the title, there is a message box containing the text 'You're logged in!'. The overall layout is a standard Breeze admin dashboard.

Dit is een standaard lay-out vanuit Breeze, en komen nu dus niet op onze admin uit. Er zijn 2 mogelijkheden om ervoor te zorgen dat het wel goed gaat. De eerste is, de standaard pagina na de login veranderen. Dit staat in de RouteServiceProvider. Deze kan je vinden in app/Providers/.

```
11 class RouteServiceProvider extends ServiceProvider
12 {
13     /**
14      * The path to the "home" route for your application.
15      *
16      * This is used by Laravel authentication to redirect users after login.
17      *
18      * @var string
19     */
20     public const HOME = '/dashboard';
```

Hier zie je dat de HOME /dashboard is. Dit zouden we kunnen veranderen. Nu kan je ook als klant straks inloggen, dus we moeten niet een admin pagina daar zetten. Maar het menu past zich nu aan op basis van de rol die de gebruiker heeft, dus we kunnen de beginpagina hiervoor gebruiken.

```
11 class RouteServiceProvider extends ServiceProvider
12 {
13     /**
14      * The path to the "home" route for your application.
15      *
16      * This is used by Laravel authentication to redirect users after login.
17      *
18      * @var string
19     */
20     public const HOME = '/';
```

Als we nu uitloggen en opnieuw inloggen zal je merken dat we nu onze eigen pagina gewoon zien na de login, met daarbij de links voor de admin.

The screenshot shows a web browser window with the URL 'laravel9v1/'. The page has a header with a logo, the name 'Cleopatra', and a navigation bar with icons for mail, messages, calendar, and user profile. On the right, there's a dropdown menu for 'Admin'. The main content area is divided into two sections: 'ADMIN' on the left and 'PUBLIC' on the right. Under 'ADMIN', there are links for 'Category Admin' and 'Link2'. Under 'PUBLIC', there are links for 'Categories' and 'Link2'.

Als we inloggen met de customer zien we de admin niet. Als we proberen via de URL er te komen krijgen we ook netjes een foutmelding:

The screenshot shows a web browser window with the URL 'laravel9v1/admin/categories'. The page has a header with a logo, the name 'Cleopatra', and a navigation bar with icons for mail, messages, calendar, and user profile. On the right, there's a dropdown menu for 'Customer'. The main content area shows an 'Error' message in a box: 'User Does Not Have The Right Roles.' This indicates that the customer user does not have the necessary permissions to access the admin categories.

Tot zover werkt de authenticatie dus goed.

2.6. Permissies in de Controller

Als we verder gaan kijken met permissies, kunnen we kijken of de rest ook werkt zoals we willen. Hiervoor gaan we inloggen met de sales@tcrmbo.nl account. Als we naar de category admin gaan zien we netjes het overzicht.

NAME	DETAILS	EDIT	DELETE
Zane Kunde	Details		
Velda Collins	Details		

Alleen, het sales account zou niet een categorie moeten kunnen verwijderen. Helaas lukt dit wel. Dit komt omdat we wel permissies hebben aangemaakt, maar nog niet gekoppeld met de functionaliteiten.

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

```
public function run()
{
    app()[\Spatie\Permission\PermissionRegistrar::class]→forgetCachedPermissions();

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

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

De construct gaan we dan voor de koppeling gebruiken.

```
class CategoryController extends Controller
{
    /**
     * Set permissions on methods
     */
    public function __construct()
    {
        $this->middleware( middleware: 'auth');
        $this->middleware( middleware: 'permission:index category', ['only' => ['index']]);
        $this->middleware( middleware: 'permission:show category', ['only' => ['show']]);
        $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']]);
    }
}
```

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

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

Als we nu als Sales proberen een categorie te verwijderen lukt dit niet meer.

The screenshot shows a web browser window with the URL `laravel9v1/admin/categories/2/delete`. The page has a dark header with navigation icons. On the left, there's a sidebar with menu items: 'ADMIN' (Category Admin, Link2), 'PUBLIC' (Categories, Link2). The main content area has a title 'Error' and a red box containing the message 'User Does Not Have The Right Permissions.' At the top right of the page, there's a user profile for 'Sales'.

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

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

Dit doen we bij de header van de tabel:

```
<th class="px-4 py-3">Edit</th>
@can('delete category')
    <th class="px-4 py-3">Delete</th>
@endcan
</tr>
```

En in de foreach bij de kolom voor de link van delete

```
@can('delete category')
<td>
    <div class="flex items-center space-x-4 text-sm">
        <a href="{{ route('categories.delete', ['category' => $category->id]) }}>
            <button class="flex items-center justify-between px-2 py-2 text-sm font-medium leading-5 text-purple-600 rounded-lg focus:outline-none focus:shadow-outline-gray" aria-label="Delete">
                <svg class="w-5 h-5" aria-hidden="true" fill="currentColor" viewBox="0 0 20 20">
                    <path fill-rule="evenodd" d="M9 2a1 1 0 00-.894.553L7.382 4H4a1 1 0 000 2v10a2 2 0 002 2h8a2 2 0 001 1v6a1 1 0 100-2h-3.382l-.724-1.447A1 1 0 0011 2H9zM7 8a1 1 0 012 0v6a1 1 0 11-2 0V8zm5-1a1 1 0 00-1 1v6a1 1 0 102 0V8a1 1 0 00-1-1z" clip-rule="evenodd"></path>
                </svg></button></a>
    </div>
</td>
@endcan
```

Als je dan kijkt wat Sales kan zien bij het overzicht zie je alleen de details en edit nog staan.

The screenshot shows a web application interface. On the left, there's a sidebar with two sections: 'ADMIN' and 'PUBLIC'. Under 'ADMIN', there are links for 'Category Admin' and 'Link2'. Under 'PUBLIC', there are links for 'Categories' and 'Link2'. The main content area has a title 'Overzicht Category' and a 'Category Toevoegen' button. Below that, there's a section titled 'Category Admin' with a table. The table has columns 'NAME', 'DETAILS', and 'EDIT'. It contains two rows: one for 'Zane Kunde' with 'Details' and 'Edit' buttons, and another for 'Velda Collins' with 'Details' and 'Edit' buttons. At the top right of the main content area, there are user profile icons and a dropdown menu for 'Sales'.

NAME	DETAILS	EDIT
Zane Kunde	Details	
Velda Collins	Details	

2.7. Opdracht 11: Permissions

Maak de volgende permissies aan.

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

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

Opdracht controle

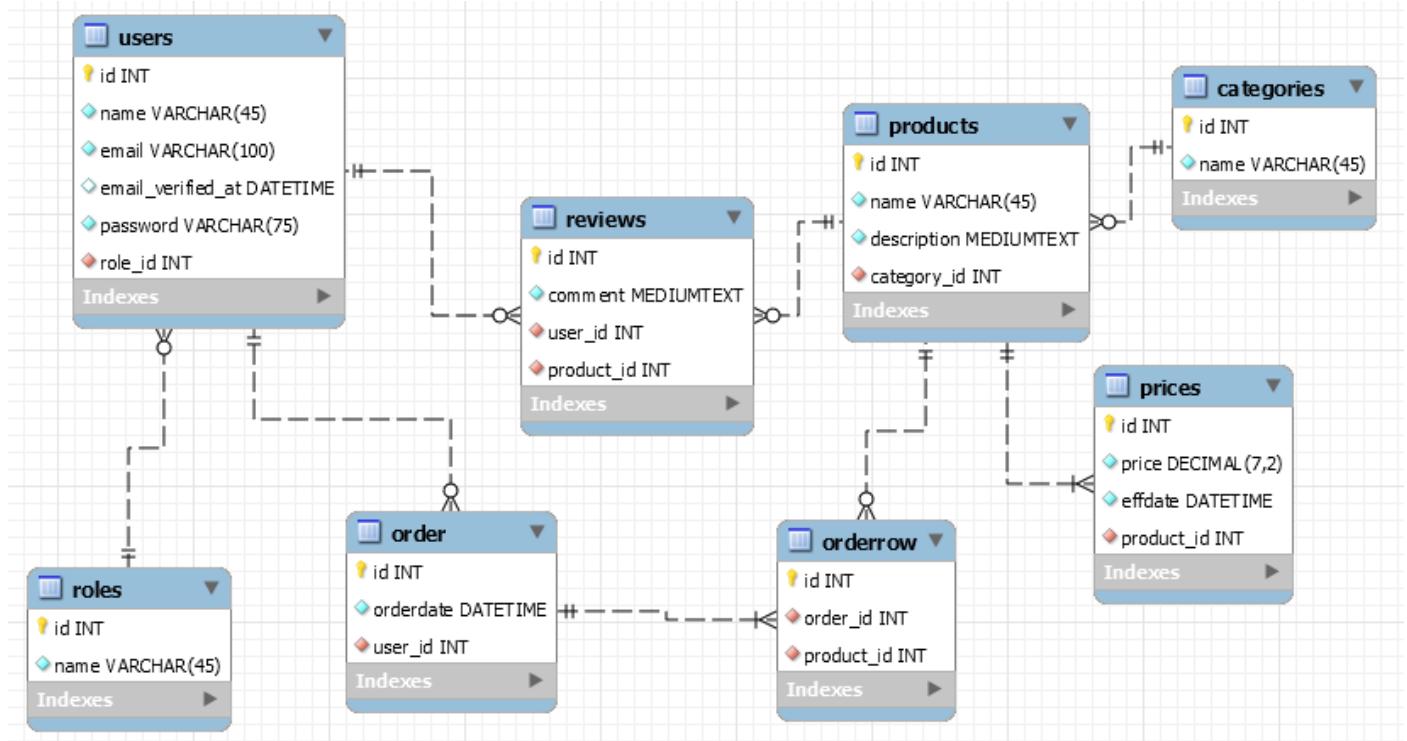
Als je de opdracht af hebt en gebruik hebt gemaakt van de opdrachten repository, kan je het volgende commando gebruiken om te controleren of alles goed werkt:

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

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

3. 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. Verder ga je ook niet los een prijs aanpassen, maar wil je de prijs van een product aanpassen, dus voor prijs hebben we sowieso product nodig. 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.

3.1. Models

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

```
PS D:\wamp\sites\laravel9v1> php artisan make:model --all Product
Model created successfully.
Factory created successfully.
Created Migration: 2022_03_07_121708_create_products_table
Seeder created successfully.
Request created successfully.
Request created successfully.
Controller created successfully.
Policy created successfully.
```

Zoals je ziet zijn er allerlei bestanden aangemaakt:

- Model: Product
- Factory: ProductFactory
- Migration: create_products_table
- Seeder: ProductSeeder
- 2 Requests: StoreProductRequest en UpdateProductRequest
- Controller: ProductController
- Policy: ProductPolicy

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

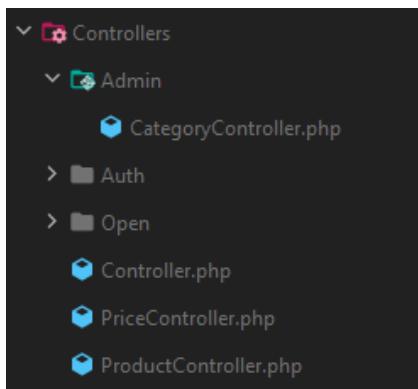
```
PS D:\wamp\sites\laravel9v1> php artisan make:model --all Price
Model created successfully.
Factory created successfully.
Created Migration: 2022_03_07_124918_create_prices_table
Seeder created successfully.
Request created successfully.
Request created successfully.
Controller created successfully.
Policy created successfully.
```

Ook hiervoor zijn nu de bestanden aangemaakt:

- Model: Price
- Factory: PriceFactory
- Migration: create_prices_table
- Seeder: PriceSeeder
- 2 Requests: StorePriceRequest en UpdatePriceRequest
- Controller: PriceController
- Policy: PricePolicy

Zoals je ziet is deze optie een stuk makkelijker dan voor elk bestand een `php artisan` commando te moeten typen. Er zit wel 1 nadeel aan deze manier. We krijgen onze controller nu niet in een admin map. Verder zijn de requests iets anders genoemd. Ok, dit is misschien de standaard, maar als je 25 requests hebt staan in dezelfde map is het toch makkelijk kijken waar `ProductStoreRequest` bijvoorbeeld staat, omdat het gewoon alfabetisch staat. We hebben ook Policies gekregen. Deze zijn om te kijken of een bepaalde actie gedaan mag worden met de huidige gebruiker.

Voorlopig gaan we dit nog niet gebruiken, maar afhandelen met de roles en permissies die we net hebben gebruikt. Later kunnen we naar de Policies kijken, want het kan natuurlijk veel uitgebreider dan wat we nu doen. Voor nu gaan we de onderdelen die zijn aangemaakt even zo neerzetten zodat het binnen ons project ook klopt. Hiervoor kijken we als eerst naar de controllers.



We zien dus de ProductController en PriceController in de hoofdmap van controllers zitten, terwijl we deze in de Admin map willen hebben. Deze kunnen we verplaatsen, maar het is goed om nog even te kijken wat aangepast moet gaan worden. Om even goed naar de verschillen te kijken van CategoryController en ProductController, zetten we deze naast elkaar:

```
namespace App\Http\Controllers\Admin;

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

class CategoryController extends Controller
```

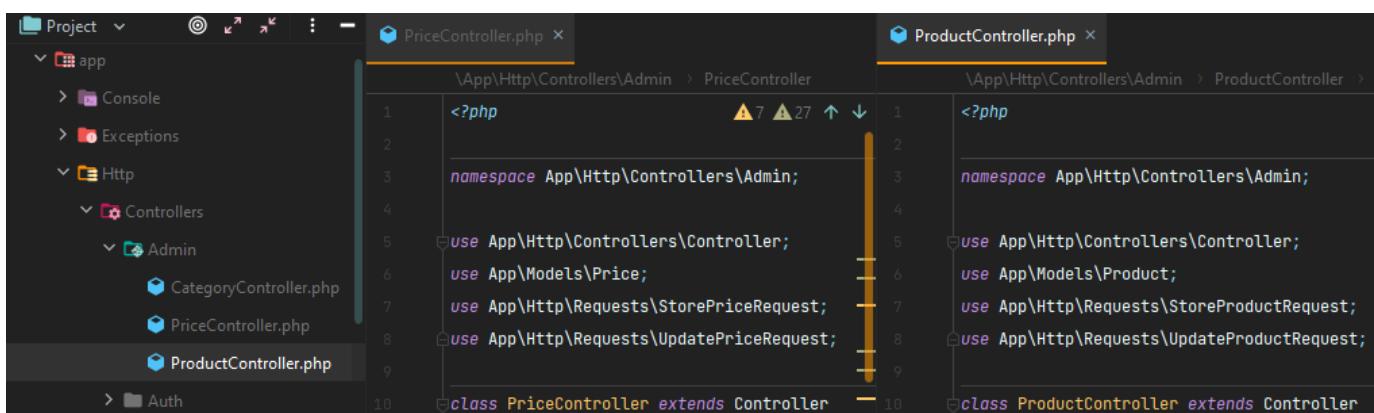


```
namespace App\Http\Controllers;

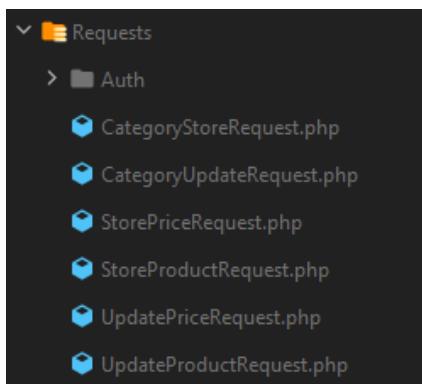
use App\Models\Product;
use App\Http\Requests\StoreProductRequest;
use App\Http\Requests\UpdateProductRequest;

class ProductController extends Controller
```

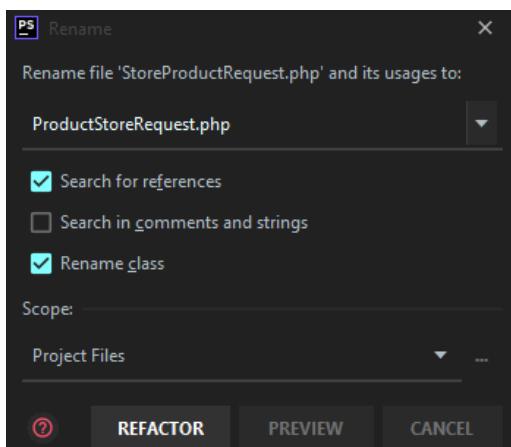
Hier zien we dat de namespace anders is. Bij de CategoryController staan er nog \Admin achter. Tevens omdat de namespace dan anders is, moet ook nog use App\Http\Controllers\Controller; gebruikt worden, zodat de basis controller gebruikt kan worden. We gaan dit ook doorvoeren voor Product en Price. We verplaatsen de bestanden naar de Admin map waar de CategoryController ook staat. De namespace veranderen we naar de Admin map en de basis controller wordt ingeladen.



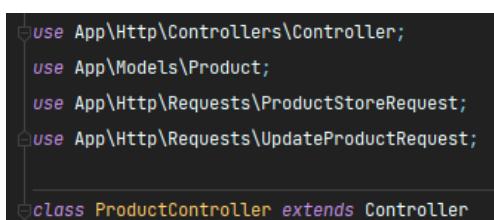
Naast deze wijziging gaan we ook even de requests hernoemen. In een project moeten de bestanden wel hetzelfde format hebben. De bestanden die er nu staan zijn dit:



Als je een bestand hernoemd in PhpStorm, kan je de refactor gebruiken. Je kan de class naam veranderen, maar ook kijken wanneer de class wordt gebruikt en het daar ook meteen laten veranderen. Voor deze optie kies ik dan ook meteen.



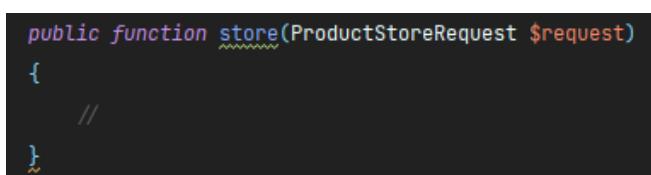
Je kan meteen zien dat de naam is veranderd. Verder kan je ook in de ProductController zien dat ook de hernoemde class gebruikt wordt:



```
use App\Http\Controllers\Controller;
use App\Models\Product;
use App\Http\Requests\ProductStoreRequest;
use App\Http\Requests\UpdateProductRequest;

class ProductController extends Controller
```

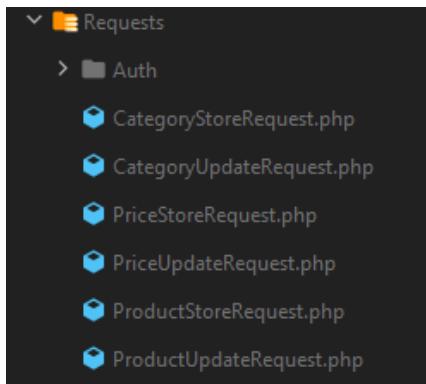
Zelfs bij de store methode in de ProductController is het al veranderd:



```
public function store(ProductStoreRequest $request)
```

Deze actie doen we voor alle product en price requests, zodat ze allemaal dezelfde standaard naamgeving hebben.

Uiteindelijk staan de requests dan van elke controller bij elkaar. Het ziet er dan zo uit:



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

```
PS D:\wamp\sites\laravel9v1> composer dump-autoload
Generating optimized autoload files
```

3.2. Migrations

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

Als eerst de products migration.

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

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

Als eerste dus up methode. De string en tekst zijn denk ik wel al duidelijk, maar het lastige stuk is de foreignId.

(<https://laravel.com/docs/9.x/migrations#foreign-key-constraints>)

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

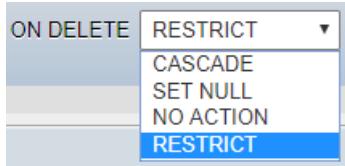
The `foreignId` method creates an `UNSIGNED BIGINT` equivalent column, while the `constrained` method will use conventions to determine the table and column name being referenced. If your table name does not match Laravel's conventions, you may specify the table name by passing it as an argument to the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

Je geeft dus aan met de naam van de foreign key naar welke tabel het moet en dat met constrained het wordt vastgelegd. We gebruiken de juiste Laravel conventie met category_id, zodat het een verwijzing wordt naar de tabel categories. Zou je de naamgeving niet volgen, kan je binnenen constrained aangeven naar welke tabel er wordt gerefereerd door daar de tabelnaam aan te geven. Dit hoeft bij ons nu niet.

```
$table→foreignId( column: 'category_id')→constrained( table: 'categories')
```

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

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



- No Action:

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

- Set NULL:

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

- Cascade:

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

- Restrict:

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

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

Dan de Price migration

```
return new class extends Migration
{
    /**
     * Run the migrations. ...
     */
    public function up()
    {
        Schema::create('prices', function (Blueprint $table) {
            $table->id();
            $table->decimal('price', 8, 2);
            $table->dateTime('effdate');
            $table->foreignId('product_id')->constrained()
                ->onUpdate('cascade')->onDelete('cascade');
            $table->timestamps();
        });
    }
}
```

Je ziet hier een effdate. Dit is een effective date, wat veel in databases gebruikt wordt. Het is bij dit voorbeeld de ingangsdatum wanneer de 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 ook dat we nu niet restrict gebruiken, 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.

3.3. Relaties in models

Nu de database is aangemaakt en de relaties tussen de tabellen duidelijk zijn, zullen we aan de slag moeten met de relaties van de models. Laravel ziet namelijk niet de relaties in de database, waardoor we deze relaties niet zo kunnen gebruiken. Het is daarom handig om de relaties tussen data op te gaan schrijven zodat we binnen het framework daar wat mee kunnen. Nu zijn models degene die alles voor de data regelen, dus het zal binnen de models beschreven moeten worden. 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. Relaties leggen in de model lijkt heel erg op de relaties in een ERD. Bij het bepalen van de relaties in een ERD gebruiken we ERDish om de relaties te beschrijven:

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

Dit waarbij de eerste keuze de optionaliteit beschrijft (may/must) en de tweede keuze de kardinaliteit (1 and only 1/1 or more). Dit moet je in beide richtingen uitschrijven om de relatie te kunnen tekenen. Vanaf de categories naar products zou het zijn:

- *Each category may have 1 or more products*

De andere richting, van products naar categories:

- *Each product must belong to 1 and only 1 category*

In de models van Laravel wordt eigenlijk dezelfde manier gebruikt, wat dus erg handig is voor ons.

In Category geven we aan dat deze een relatie heeft met Product door middel van een methode. Het is van belang om de methode zo duidelijk mogelijk te maken. Met bijv. product() is duidelijk dat dit met een product te maken heeft, maar als het te maken heeft met meerdere producten, dan zou het products() worden. Het is dus van belang dat het duidelijk is waarom de relatie er is, en door middel van enkelvoud/ meervoud weten we of we 1 of meerdere rijen terugkrijgen.

Hierbij dan de model van Category:

```
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Category extends Model
{
    use HasFactory;

    protected $guarded = ['name'];

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

Je ziet dus dat in de model we hetzelfde doen. De naam van de methode is naar welk model we toe verwijzen. Omdat we eenhasMany hebben, zorgen we ervoor dat de methode meervoud is, dus products(). 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.

Van product naar category is het:

Each product must belong to 1 and only 1 category

Van product naar price is het:

Each product must have 1 or more prices

Hierdoor hebben we dus voor product naar category de methode category() nodig. Voor product naar price verwachten we meerdere prijzen, dus wordt het meervoud: prices()

```
class Product extends Model
{
    use HasFactory;

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

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

Bij category gebruiken we een belongsTo. Dit is altijd bij een 1 op veel relatie zo, aan de ene kant een hasMany en aan de andere kant een belongsTo. Dit klopt ook, want een product heeft maar 1 category. De bevestiging kan je ook zien in de ERDish hiervan. Bij de methode prices() geven we dan de relatie van Product met Price. Een Product kan meerdere prijzen hebben, waardoor we de hasMany gebruiken.

En dan de Price model.

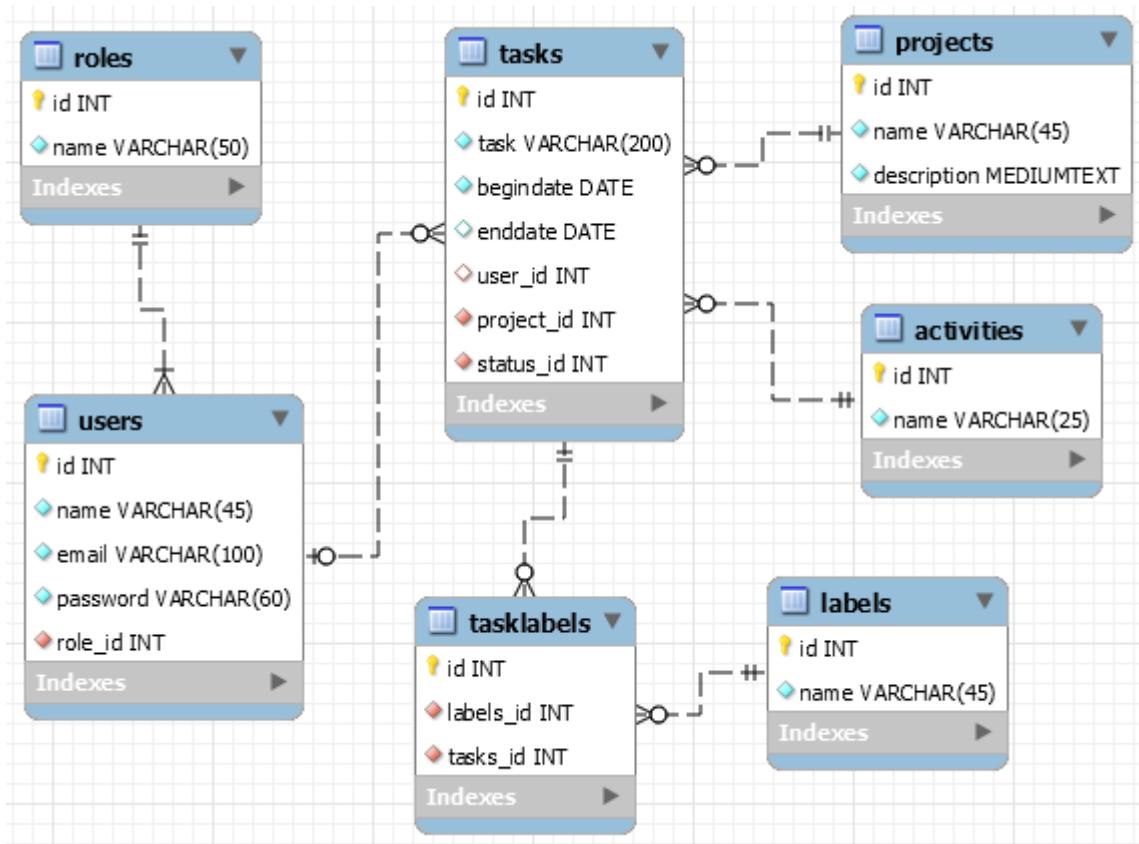
```
class Price extends Model
{
    use HasFactory;

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

Een Price behoort bij een product. Hierdoor ook in Price een methode product(). Nu hebben we binnen Laravel aangegeven hoe de relaties zijn waardoor we de relaties die in de database staan ook kunnen gebruiken.

3.4. Opdracht 12: Models en Migrations

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



Maak dus de models aan voor activities en tasks.

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

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

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

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

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

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

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

3.5. Factory

Om ervoor te zorgen dat we straks data hebben voor de products en prices gaan we aan de slag met factories. 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.

```
3  namespace Database\Factories;
4
5  use App\Models\Category;
6  use Illuminate\Database\Eloquent\Factories\Factory;
7
8  /**
9   * @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\Product>
10  */
11 class ProductFactory extends Factory
12 {
13     /**
14      * Define the model's default state. ...
15     */
16     public function definition()
17     {
18         return [
19             'name' => $this->faker->name,
20             'description' => $this->faker->paragraph( nbSentences: 15),
21             'category_id' => Category::all()->random()->id
22         ];
23     }
24 }
25
26 }
```

Omdat we op regel 23 de model van Category gebruiken zullen we op regel 5 de class wel moeten toevoegen. Voor de category_id, wat een foreign key is die refereert naar de tabel categories, gaan we dus kijken in die categories tabel met de model Category. We halen alle categories op, kiezen er daar 1 van uit, en pakken alleen de id ervan om in de 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.

```
3 namespace Database\Factories;
4
5 use App\Models\Product;
6 use Carbon\Carbon;
7 use Illuminate\Database\Eloquent\Factories\Factory;
8
9 /**
10 * @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\Price>
11 */
12 class PriceFactory extends Factory
13 {
14     /**
15      * Define the model's default state. ...
16     */
17     public function definition()
18     {
19         return [
20             'price' => $this->faker->randomFloat( nbMaxDecimals: 2, min: 2, max: 6),
21             // 'effdate' => $this->faker->dateTime,
22             'effdate' => Carbon::today()->subDays(random_int(0, 365)),
23             'product_id' => Product::all()->random()->id
24         ];
25     }
26 }
27
28 }
```

Voor de datum van effdate, kan je faker gebruiken. Binnen Laravel wordt vaak Carbon gebruikt, wat een uitbreiding is op de DateTime class van php. De documentatie van Carbon kan je hier vinden:

<https://carbon.nesbot.com/docs/>

Carbon zit standaard al in Laravel, dus we hoeven alleen de class in te laden, wat we op regel 6 doen. Omdat we wel steeds een andere datum willen, kunnen we met subDays een aantal dagen eraf halen, waarbij de random_int steeds een andere waarde heeft.

We gebruiken Carbon, omdat later dit goed te gebruiken is voor testen die we kunnen gaan schrijven.

De product_id doen we net als de category_id, dus alle producten ophalen en daar random 1 product uit halen, waarvan we de id pakken.

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

3.6. Seed

Als je aan de slag gaat met de seed, waarbij we te maken krijgen met de relaties, kan je in de problemen komen als je het niet goed doet. Dit komt doordat we met data altijd een bepaald doel hebben. Straks willen we, net zoals bij categorie, een overzicht van producten hebben met daarbij de huidige prijs.

Als we dan de huidige factory gebruiken van Price, pakken we een random product en zetten we de prijs in de database.

```
class PriceFactory extends Factory
{
    /**
     * Define the model's default state. ...
     */
    public function definition()
    {
        return [
            'price' => $this->faker->randomFloat( nbMaxDecimals: 2, min: 2, max: 6),
            // 'effdate' => $this->faker->dateTime,
            'effdate' => Carbon::today()->subDays(random_int(0, 365)),
            'product_id' => Product::all()->random()->id
        ];
    }
}
```

Maar wat als we 5 producten en 5 prijzen seeden. Zou dan elk product een prijs hebben???

Kunnen we dan in een overzicht van producten altijd een prijs verwachten, of zou een product ook geen prijs kunnen hebben?

Natuurlijk zal dat niet het geval zijn. Het zou wel heel toevallig zijn als de random methode toevallig alle 5 de producten zou kiezen. Voor dit probleem is natuurlijk een oplossing, namelijk het gebruik van factory relationships: <https://laravel.com/docs/9.x/database-testing#factory-relationships>

De handigste manier om dit voor elkaar te krijgen is met magic methods. We hebben namelijk de relaties tussen Product en Price beschreven in de models met methods. Deze methods kunnen we nu gebruiken om aan te geven dat bijvoorbeeld een product meerdere prijzen al standaard krijgt.

Using Magic Methods

For convenience, you may use Laravel's magic factory relationship methods to build relationships. For example, the following example will use convention to determine that the related models should be created via a `posts` relationship method on the `User` model:

```
$user = User::factory()
    ->hasPosts(3)
    ->create();
```

Voor de product seed gaan we dus gebruiken maken van deze methode in de Product model:

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

Laten we nu dan naar de ProductSeeder gaan kijken.

```
5  use App\Models\Product;
6  use Illuminate\Database\Console\Seeds\WithoutModelEvents;
7  use Illuminate\Database\Seeder;
8
9  class ProductSeeder extends Seeder
10 {
11     /** Run the database seeds. ...*/
12     public function run()
13     {
14         Product::factory()
15             ->times( count: 100)
16             ->hasPrices(2)
17             ->create();
18     }
19 }
20
21 }
```

In de seeder gaan we de model aanspreken, om dan de factory ervan te gebruiken. Hierdoor moet op regel 5 de model worden ingeladen. Omdat we 25 categorieën hebben, probeer ik redelijk wat producten te maken, dus daarom 100. Omdat onze methode in de model prices() is, moeten we hasPrices() gebruiken om deze relatie te gebruiken in de seeder. Per product gaan we nu 2 prices krijgen.

We zorgen dat de ProductSeeder in de DatabaseSeeder staat:

```
class DatabaseSeeder extends Seeder
{
    /** Seed the application's database. ...*/
    public function run()
    {
        // \App\Models\User::factory(10)->create();
        $this->call([
            RoleAndPermissionSeeder::class,
            UserSeeder::class,
            CategorySeeder::class,
            ProductSeeder::class
        ]);
    }
}
```

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

```
PS D:\wamp\sites\laravel9v1> 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 (37.12ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (31.47ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (31.66ms)
Migrating: 2019_12_14_000001_create_personal_access_tokens_table
Migrated: 2019_12_14_000001_create_personal_access_tokens_table (47.23ms)
Migrating: 2022_02_11_082343_create_categories_table
Migrated: 2022_02_11_082343_create_categories_table (12.74ms)
Migrating: 2022_02_11_093836_update_name_length_categories_table
Migrated: 2022_02_11_093836_update_name_length_categories_table (130.03ms)
Migrating: 2022_03_01_152547_create_permission_tables
Migrated: 2022_03_01_152547_create_permission_tables (458.12ms)
Migrating: 2022_03_07_121708_create_products_table
Migrated: 2022_03_07_121708_create_products_table (62.41ms)
Migrating: 2022_03_07_124918_create_prices_table
Migrated: 2022_03_07_124918_create_prices_table (62.71ms)
Seeding: Database\Seeders\RoleAndPermissionSeeder
Seeded: Database\Seeders\RoleAndPermissionSeeder (149.60ms)
Seeding: Database\Seeders\UserSeeder
Seeded: Database\Seeders\UserSeeder (287.97ms)
Seeding: Database\Seeders\CategorySeeder
Seeded: Database\Seeders\CategorySeeder (50.03ms)
Seeding: Database\Seeders\ProductSeeder
Seeded: Database\Seeders\ProductSeeder (1,323.75ms)
Database seeding completed successfully.
```

Als we de tabel prices nu bekijken in de phpMyAdmin, zien we netjes dat er 200 prices zijn, waarbij steeds 2 rijen aan een product zijn gekoppeld.

✓ Weergave van records 0 - 24 (200 totaal, Query duurde 0,0002 seconden.)

SELECT * FROM `prices`

Profiling [Inline bewerken] [Wijzigen] [Verklaar SQL] [Genereer PHP-code] [Ververs]

1 > >> | Toon alles | Aantal rijen: 25 Filter rijen: Zoek in deze tabel Sorteren op sleutel: ↴

+ Opties

	Wijzigen	Kopiëren	Verwijderen	id	price	effdate	product_id	created_at	updated_at
<input type="checkbox"/>				1	3.70	2021-11-20 00:00:00	1	2022-03-08 12:39:55	2022-03-08 12:39:55
<input type="checkbox"/>				2	4.58	2021-11-08 00:00:00	1	2022-03-08 12:39:55	2022-03-08 12:39:55
<input type="checkbox"/>				3	2.14	2021-06-08 00:00:00	2	2022-03-08 12:39:55	2022-03-08 12:39:55
<input type="checkbox"/>				4	2.63	2021-10-05 00:00:00	2	2022-03-08 12:39:55	2022-03-08 12:39:55
<input type="checkbox"/>				5	4.63	2021-05-01 00:00:00	3	2022-03-08 12:39:55	2022-03-08 12:39:55
<input type="checkbox"/>				6	3.47	2021-10-10 00:00:00	3	2022-03-08 12:39:55	2022-03-08 12:39:55

En ja, er staan ook gewoon producten in de tabel products

✓ Weergave van records 0 - 24 (100 totaal, Query duurde 0,0003 seconden.)

SELECT * FROM `products`

Profiling [Inline bewerken] [Wijzigen] [Verklaar SQL] [Genereer PHP-code] [Ververs]

1 > >> Toon alles Aantal rijen: 25 Filter rijen: Zoek in deze tabel Sorteren op sleutel: Geen

+ Opties

<input type="checkbox"/>		Kopiëren	<input type="checkbox"/> Verwijderen	1	Stephany Lueilwitz DVM	Itaque perspiciatis ut et mollitia soluta. Archite...	13	2022-03-08 12:39:55	2022-03-08 12:39:55
<input type="checkbox"/>		Kopiëren	<input type="checkbox"/> Verwijderen	2	Dr. Tiana Carter MD	Iusto culpa explicabo enim odit. Aliquam et illum ...	21	2022-03-08 12:39:55	2022-03-08 12:39:55
<input type="checkbox"/>		Kopiëren	<input type="checkbox"/> Verwijderen	3	Mr. Adriel Murray IV	Reiciendis repellat ipsum nihil quia impedit sunt....	23	2022-03-08 12:39:55	2022-03-08 12:39:55
<input type="checkbox"/>		Kopiëren	<input type="checkbox"/> Verwijderen	4	Daren Schroeder	Quos dicta vel dignissimos laudantium vel eum fugi...	20	2022-03-08 12:39:55	2022-03-08 12:39:55
<input type="checkbox"/>		Kopiëren	<input type="checkbox"/> Verwijderen	5	Leopoldo Emmerich	Enim repudiandae quasi facere dolores nemo. Sint d...	12	2022-03-08 12:39:55	2022-03-08 12:39:55
<input type="checkbox"/>		Kopiëren	<input type="checkbox"/> Verwijderen	6	Prof. Anne Barrows	Nihil itaque dignissimos voluptas sit eos repellen...	21	2022-03-08 12:39:55	2022-03-08 12:39:55

We kunnen nu nog wel een PriceSeeder gebruiken als we willen, maar hoeft niet per se. De testdata hebben we nu wel. Toch gaan we het gewoon nog even doen, dus dit is de PriceSeeder:

```
5      use App\Models\Price;
6
7      use Illuminate\Database\Console\Seeds\WithoutModelEvents;
8
9      use Illuminate\Database\Seeder;
10
11     class PriceSeeder extends Seeder
12     {
13
14         /**
15          * Run the database seeds. ...
16         */
17         public function run()
18         {
19             Price::factory()
20                 ->times( count: 200 )
21                 ->create();
22         }
23     }
24 }
```

De DatabaseSeeder vullen we dan weer aan.

```
1 class DatabaseSeeder extends Seeder
2 {
3
4     /**
5      * Seed the application's database. ...
6      */
7     public function run()
8     {
9
10         // \App\Models\User::factory(10)->create();
11
12         $this->call([
13
14             RoleAndPermissionSeeder::class,
15
16             UserSeeder::class,
17
18             CategorySeeder::class,
19
20             ProductSeeder::class,
21
22             PriceSeeder::class
23         ]);
24     }
25 }
```

Dan seeden we opnieuw de database.

```
PS D:\wamp\sites\laravel9v1> 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 (32.59ms)
```

Als we dan in de database kijken, zien we dat er nu 400 prices zijn.

✓ Weergave van records 0 - 24 (400 totaal, Query duurde 0,0002 seconden.)

SELECT * FROM `prices`

Profiling [Inline bewerken] [Wijzigen] [Verklaar SQL] [Genereer PHP-code] [Ververs]

1 < > >> | Toon alles | Aantal rijen: 25 Filter rijen: Zoek in deze tabel Sorteren op sleutel:

+ Opties

	Wijzigen	Kopiëren	Verwijderen	id	price	effdate	product_id	created_at	updated_at
<input type="checkbox"/>				1	2.42	2021-11-12 00:00:00	1	2022-03-08 12:47:34	2022-03-08 12:47:34
<input type="checkbox"/>				2	3.23	2022-01-02 00:00:00	1	2022-03-08 12:47:34	2022-03-08 12:47:34
<input type="checkbox"/>				3	2.26	2021-12-25 00:00:00	2	2022-03-08 12:47:34	2022-03-08 12:47:34
<input type="checkbox"/>				4	2.28	2021-04-12 00:00:00	2	2022-03-08 12:47:34	2022-03-08 12:47:34

Nu zie je bij de eerste 200 dat ze steeds per 2 aan een product zijn gekoppeld. Ga je boven de id=200 kijken, zie je dat het geheel random is:

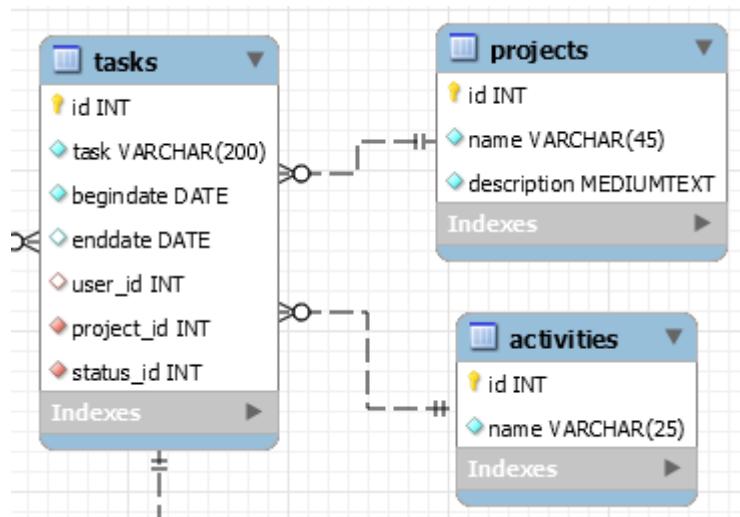
← T →

	Wijzigen	Kopiëren	Verwijderen	id	price	effdate	product_id	created_at	updated_at
<input type="checkbox"/>				226	5.03	2021-11-02 00:00:00	16	2022-03-08 12:47:36	2022-03-08 12:47:36
<input type="checkbox"/>				227	4.08	2021-07-30 00:00:00	67	2022-03-08 12:47:36	2022-03-08 12:47:36
<input type="checkbox"/>				228	4.51	2021-04-19 00:00:00	29	2022-03-08 12:47:36	2022-03-08 12:47:36
<input type="checkbox"/>				229	5.22	2021-07-02 00:00:00	75	2022-03-08 12:47:36	2022-03-08 12:47:36
<input type="checkbox"/>				230	2.10	2021-08-09 00:00:00	67	2022-03-08 12:47:36	2022-03-08 12:47:36

Deze zijn dan ook toegevoegd door de PriceSeeder. De seed is nu klaar, want alle testdata staat nu in de database.

3.7. Opdracht 13: Factory

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



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

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

Opdracht controle

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

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

3.8. Opdracht 14: Seeder

Bij opdracht 12 en 13 zijn de migration en factories gemaakt voor activities en tasks. Nu wordt het tijd om de seeder te gaan maken, waarbij je netjes de koppeling maakt met activities, projects, users en tasks.

De users zijn al aangemaakt, net als de projects. Verder stond bij de hoofdopdracht het volgende:

- Een taak heeft een activity, bijvoorbeeld: Todo, Doing, Testing, Verify, Done

Zorg dat deze allemaal in de tabel activities komen door middel met de [ActivitySeeder](#), als het volgende:

id	name
1	Todo
2	Doing
3	Testing
4	Verify
5	Done

De volgende seeder zal je moeten aanpassen/ aanmaken:

- ProjectSeeder: Project moet minimaal 2 tasks hebben. Er moeten minimaal 5 projecten aangemaakt worden.
- TaskSeeder: Er moeten minimaal 10 random tasks aangemaakt worden in de TaskSeeder

Opdracht controle

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

```
.\vendor\bin\pest --group=Opdracht14
```

3.9. Product Controller

3.9.1. Index

Bij de index gaan we meteen al aan de slag met relaties. Voor elk product wil ik namelijk laten zien in welke Category deze zit. Ook 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.

```
class ProductController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return View
     */
    public function index(): View
    {
        $products = Product::all();
        return view('admin.products.index', compact('products'));
    }
}
```

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

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

```
@extends('layouts.layout')
@section('topmenu')
<nav class="card">
    <div class="max-w-7xl mx-auto px-2 sm:px-6 lg:px-8">
        <div class="relative flex items-center justify-between h-16">
            <div class="flex-1 flex items-center justify-center sm:items-stretch sm:justify-start">
                <div class="sm:block sm:ml-6">
                    <div class="flex space-x-4">
                        <!-- Current: "bg-gray-900 text-white", Default: "text-gray-300 hover:bg-gra
                        <a href="{{ route('products.index') }}" class="text-gray-800 px-3 py-2 rounde
                        <a href="{{ route('products.create') }}" class="text-gray-800 hover:text-teal-600 px-3 py-2 rounded-l-lg border border-gray-300 transition duration-150 ease-in-out">
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </nav>
@endsection
```

De titel is aangepast naar Products.

```
@section('content')


<div class="card-header flex flex-row justify-between">
        <h1 class="h6">Product Admin</h1>
    </div>
    
    @if(session('status'))
        <div class="card-body">
            <div class="bg-green-400 text-green-800 rounded-lg shadow-md p-6 pr-10 mb-8" style="...>{{ session('status') }}</div>
        </div>
    @endif
    @if(session('status-wrong'))
        <div class="card-body">
            <div class="bg-red-400 text-red-800 rounded-lg shadow-md p-6 pr-10 mb-8" style="...>{{ session('status-wrong') }}</div>
        </div>
    @endif


```

De routes van de linkjes zijn ook naar de product routes. In de foreach verwachten we een \$products array, die we doorlopen met \$product

```
<tbody class="bg-white divide-y">
@foreach($products as $product)
    <tr class="text-gray-700">
        <td class="px-4 py-3 text-sm">{{ $product->name }}</td>
        <td class="px-4 py-3 text-sm"><a href="{{ route('products.show', ['product' => $product->id]) }}">Details</a></td>
        <td class="px-4 py-3">
            <div class="flex items-center space-x-4 text-sm">
                <a href="{{ route('products.edit', ['product' => $product->id]) }}"...>
            </div>
        </td>
        @can('delete product')
            <td>
                <div class="flex items-center space-x-4 text-sm">
                    <a href="{{ route('products.delete', ['product' => $product->id]) }}"...>
                </div>
            </td>
        @endcan
    </tr>
@endforeach
</tbody>
```

Let even op, dat er ook al in staat, dat alleen met permissie ‘delete product’ we de delete kunnen zien. De tabel zal straks uitgebreid worden met een aantal kolommen, zoals categorie en laatste prijs, maar eerst even een simpel overzicht voordat we het moeilijker maken.

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

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

    Route::resource('admin/categories', Admin\CategoryController::class);

    Route::get('admin/products/{product}/delete', [Admin\ProductController::class, 'delete'])
        ->name('products.delete');

    Route::resource('admin/products', Admin\ProductController::class);
});
```

We zetten hem meteen binnen de middleware, want we zijn al ingelogd. De permissies in de controller zelf laten we nog even weg.

Als laatst nog even zorgen dat we wel vanuit onze site bij de pagina kunnen. Dan doen we in de lay-out (masterpage)

```
@else
@hasanyrole('sales|admin')

<p class="uppercase text-xs text-gray-600 mb-4 tracking-wider">Admin</p>
<a href="{{ route('categories.index') }}" class="mb-3 capitalize font-medium text-sm hover:text-teal-600 transition ease-in-out duration-500">Category Admin</a>
<a href="{{ route('products.index') }}" class="mb-3 capitalize font-medium text-sm hover:text-teal-600 transition ease-in-out duration-500">Product Admin</a>
@endhasanyrole
```

Als we nu gaan uitproberen of we op de site kunnen en ook bij de product admin wat krijgen te zien, zien we dit:

The screenshot shows a web browser window with the URL `laravel9v1/admin/products`. The page has a dark header with a search icon, a refresh icon, and a user profile icon labeled "Admin". On the left, there's a sidebar with two sections: "ADMIN" containing "Category Admin" and "Product Admin", and "PUBLIC" containing "Categories" and "Link2". The main content area has a title "Product Admin" and two buttons: "Overzicht Product" and "Product Toevoegen". Below this is a table with three rows of product data:

NAME	DETAILS	EDIT	DELETE
Nathan O'Reilly	Details		
Miss Nellie Sauer II	Details		
Moses Macejkovic	Details		

Alles werkt netjes. Je ziet dat we geen delete zien, omdat daar wel permissie voor nodig is en die hebben we nog niet geregeld. Verder staan er 100 producten in de database, dus het wordt scrollen, zeker als je meer producten hebt. Een zoekbalk zou mooi zijn, maar dat werkt het makkelijkst met een stuk javascript erbij, dus dat gaan we nu even niet maken. We kunnen wel paginate() gebruiken om in ieder geval het overzichtelijker te maken. Verder gaan we natuurlijk aan de slag met de uitbreiding van het overzicht. De categorie en laatste prijs moet in dit overzicht komen te staan.

Voor het zover is, kijken we even met de debugbar naar de pagina, en dan vooral naar de query's. (mocht je de debugbar niet hebben, kijk even in het level 0 document voor de installatie)

NAME	DETAILS	EDIT	DELETE
Nathan O'Reilly	Details		
Miss Nellie Sauer II	Details		
Moses Macejkovic	Details		

```

= 3 AND `model_has_roles`.`model_type` = 'App\Models\User'
Select * From `Products`
Select `Permissions`.* , `Model_has_permissions`.`Model_id` As
`Pivot_model_id` , `Model_has_permissions`.`Permission_id` As
`Pivot_permission_id` , `Model_has_permissions`.`Model_type` As
`Pivot_model_type` From `Permissions` Inner Join `Model_has_permissions`
On `Permissions`.`Id` = `Model_has_permissions`.`Permission_id` Where
`Model_has_permissions`.`Model_id` = 1 And

```

Nu nog niks aan de hand, maar dit even als voorbereiding op de aanvullingen die we gaan maken. Op dit moment worden er 4 query's uitgevoerd, waarvan 3 voor de users & permissies en 1 voor het overzicht van de producten.

3.9.2. Uitbreiden van de index

De eerste uitbreiding van de index zal de categorie worden. Dit is eigenlijk heel erg makkelijk binnen Laravel. De product heeft namelijk een relatie met category, waardoor we deze relatie kunnen gebruiken om bij de naam van de category te komen.

In de model hebben we de relatie al beschreven:

```
class Product extends Model
{
    use HasFactory;

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

Hierdoor kunnen we nu categorie gebruiken in de index bij product. In de lay-out kunnen we dus de relatie van de model gebruiken.

```
<tbody class="bg-white divide-y">
@foreach($products as $product)
    <tr class="text-gray-700">
        <td class="px-4 py-3 text-sm">{{ $product->name }}</td>
        <td class="px-4 py-3 text-sm">{{ $product->category->name }}</td>
        <td class="px-4 py-3 text-sm"><a href="{{ route('products.show', ['product' =
```

We zien hier dus \$product->category->name

Dit betekent, van dit product, met de relatie category, willen we de naam verkrijgen.

Hoe makkelijk een join is met een model en dat we dit overal kunnen gebruiken is dus van grote waarde om makkelijk informatie op het scherm te krijgen.

Als we dan nu naar onze index van products gaan kijken in de browser zien we netjes de categorie erbij staan.

The screenshot shows a Laravel application's admin panel. On the left, there's a sidebar with 'ADMIN' and 'PUBLIC' sections. Under 'ADMIN', 'Category Admin' and 'Product Admin' are listed. Under 'PUBLIC', 'Categories' and 'Link2' are listed. The main content area has tabs 'Overzicht Product' and 'Product Toevoegen'. Below that, a section titled 'Product Admin' lists three products:

NAME	CATEGORY	DETAILS	EDIT	DELETE
Nathan O'Reilly	Liliana Turcotte	Details		
Miss Nellie Sauer II	Randi Wisoky	Details		
Moses Macejkovic	Ms. Karina Dietrich	Details		

At the bottom, a debug bar shows several database queries and their execution times. The queries are:

- Select * From `Categories` Where `Categories`.`Id` = 4 Limit 1
- Select * From `Categories` Where `Categories`.`Id` = 12 Limit 1
- Select * From `Categories` Where `Categories`.`Id` = 9 Limit 1
- Select * From `Categories` Where `Categories`.`Id` = 19 Limit 1
- Select * From `Categories` Where `Categories`.`Id` = 2 Limit 1

Execution times range from 240µs to 270µs. The total response time is 339ms.

Ik heb expres nu de debugbar open staan, want kijk is naar de hoeveelheid query's die uitgevoerd worden. Voordat we de categories hadden toegevoegd werden er 4 query's uitgevoerd en nu 104. Dit komt omdat per productregel een query wordt uitgevoerd om te kijken welke categorie erbij hoort. Bij Laravel is dit bekend als het N+1 probleem. Gelukkig hebben ze hier al wat op bedacht en dat heet eager loading. (<https://laravel.com/docs/9.x/eloquent-relationships#eager-loading>)

We veranderen de code naar dit:

```
public function index(): View
{
    $products = Product::with('category')->get();
    return view('admin.products.index', compact('products'));
}
```

Wat we hier aangeven is, haal alle producten op en maak gebruik van de methode category om ook meteen die gegevens op te halen.

Het resultaat op de pagina is precies hetzelfde

The screenshot shows a Laravel application interface. At the top, there's a header with a back arrow, forward arrow, refresh button, and a URL bar showing 'laravel9v1/admin/products'. To the right of the URL are standard browser controls like search, refresh, and a menu. Below the header is a navigation bar with icons for email, messages, and other admin functions. On the far right of the navigation bar is a user profile icon labeled 'Admin'.

The main content area has a sidebar on the left with sections for 'ADMIN' (Category Admin, Product Admin) and 'PUBLIC' (Categories, Link2). The main content area is titled 'Product Admin' and displays a table with three rows of data:

NAME	CATEGORY	DETAILS	EDIT	DELETE
Nathan O'Reilly	Liliana Turcotte	Details		
Miss Nellie Sauer II	Randi Wisoky	Details		
Moses Macejkovic	Ms. Karina Dietrich	Details		

At the bottom of the page, there's a footer with various icons and a detailed performance timeline showing database queries and their execution times.

Maar als je kijkt hoeveel query's worden uitgevoerd, is het nog maar 5. Het maakt nu niet uit hoeveel rijen we ophalen, want het getal zal niet groter worden, wat eerst wel het geval was omdat het voor elke rij een query uitvoerde. In de lay-out blijft de code gewoon hetzelfde, hierdoor werkt het ook gewoon meteen.

Voor de huidige prijs van een product wordt het lastiger. Dit omdat we met de relatie die we hebben beschreven dit niet kunnen doen.

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

Bij de methode `prices()` 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`

De Product model breiden we dus uit met de latest_price methode.

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

// get latest price of a product
public function latest_price()
{
    return $this->hasOne( related: Price::class)->orderBy( column: 'effdate', direction: 'desc');
}
```

Nu zie je bij latest_price dat we geenhasMany hebben, maar hasOne. Dit komt omdat we maar 1 prijs terug willen krijgen. Om deze laatste prijs te krijgen sorteren we de prijzen van het product op effdate, en met de laatste datum willen we eerst. Deze wordt namelijk op dat moment alleen teruggegeven. Binnen de tabel in de index voegen we dit ook toe en gebruiken `$product->latest_price->price` om de info te krijgen

```
<tbody class="bg-white divide-y">
@foreach($products as $product)
    <tr class="text-gray-700">
        <td class="px-4 py-3 text-sm">{{ $product->name }}</td>
        <td class="px-4 py-3 text-sm">{{ $product->category->name }}</td>
        <td class="px-4 py-3 text-sm">{{ $product->latest_price->price }}</td>
        <td class="px-4 py-3 text-sm"><a href="{{ route('products.show', ['product'
    <td class="px-4 py-3">
```

Binnen de ProductController gaan we gebruiken maken van Eager Loading voor latest_price

```
public function index(): View
{
    $products = Product::with( relations: 'category', 'latest_price')->get();
    return view( view: 'admin.products.index', compact( var_name: 'products'));
}
```

In ons overzicht op de pagina zie je dan nu ook de laatste prijs staan.

The screenshot shows a Laravel admin interface titled 'Cleopatra'. On the left, there's a sidebar with 'ADMIN' and 'PUBLIC' sections. Under 'ADMIN', there are links for 'Category Admin' and 'Product Admin'. Under 'PUBLIC', there are links for 'Categories' and 'Link2'. The main content area is titled 'Product Admin' and shows a table of products:

NAME	CATEGORY	PRICE	DETAILS	EDIT	DELETE
Nathan O'Reilly	Liliana Turcotte	2.39	Details		
Miss Nellie Sauer II	Randi Wisoky	2.26	Details		
Moses Macejkovic	Ms. Karina Dietrich	5.50	Details		

At the bottom, a debug bar shows the following SQL queries:

```
Select * From `Products`  
Select * From `Categories` Where `Categories`.`Id` In (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25)  
Select * From `Prices` Where `Prices`.`Product_id` In (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52)
```

Je ziet in de debugbar ook dat we door de eager loading maar 6 query's uitvoeren. Als laatst nog het scrollen wegwerken. Dit kunnen we doen door middel van paginate.

In de controller wijzigen we de get voor paginate:

```
public function index(): View  
{  
    $products = Product::with( relations: 'category', 'latest_price')→paginate( perPage: 25);  
    return view( view: 'admin.products.index', compact( var_name: 'products'));  
}
```

En in de index view zetten we de links erbij zodra de tabel is gesloten:

```
        </tbody>  
    @endforeach  
  </table>  
  {{ $products→links() }}  
</div>
```

Nu krijgen we een net resultaat, met 25 producten per pagina, met aan de onderkant netjes de links voor de pagina's:

Lonny Hane	Prof. Vilma Runte	4.30	Details	
Ozella Lebsack	Catharine Mayer	4.78	Details	
« Previous			Next »	

3.9.3. Opdracht 15: Index van Tasks

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

- Task id
- De taak
- Startdatum
- Einddatum
- Naam van gebruiker
- Projectnaam
- Naam van de activity

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

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

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

Opdracht controle

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

```
.\vendor\bin\pest --group=Opdracht15
```

3.9.4. Permissies voor product controller

Om ervoor te zorgen dat de permissies voor products op orde zijn, zullen we als eerst de permissies moeten aanmaken. Dit doen we in de RoleAndPermissionSeeder.

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

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

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

Het koppelen van deze permissies zal eronder gaan staan.

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

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

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

De customer heeft nog steeds nergens recht op in de admin, de sales mag alleen niet verwijderen en de admin mag sowieso al alles. In de ProductController zullen we dan de permissies moeten koppelen aan de methodes.

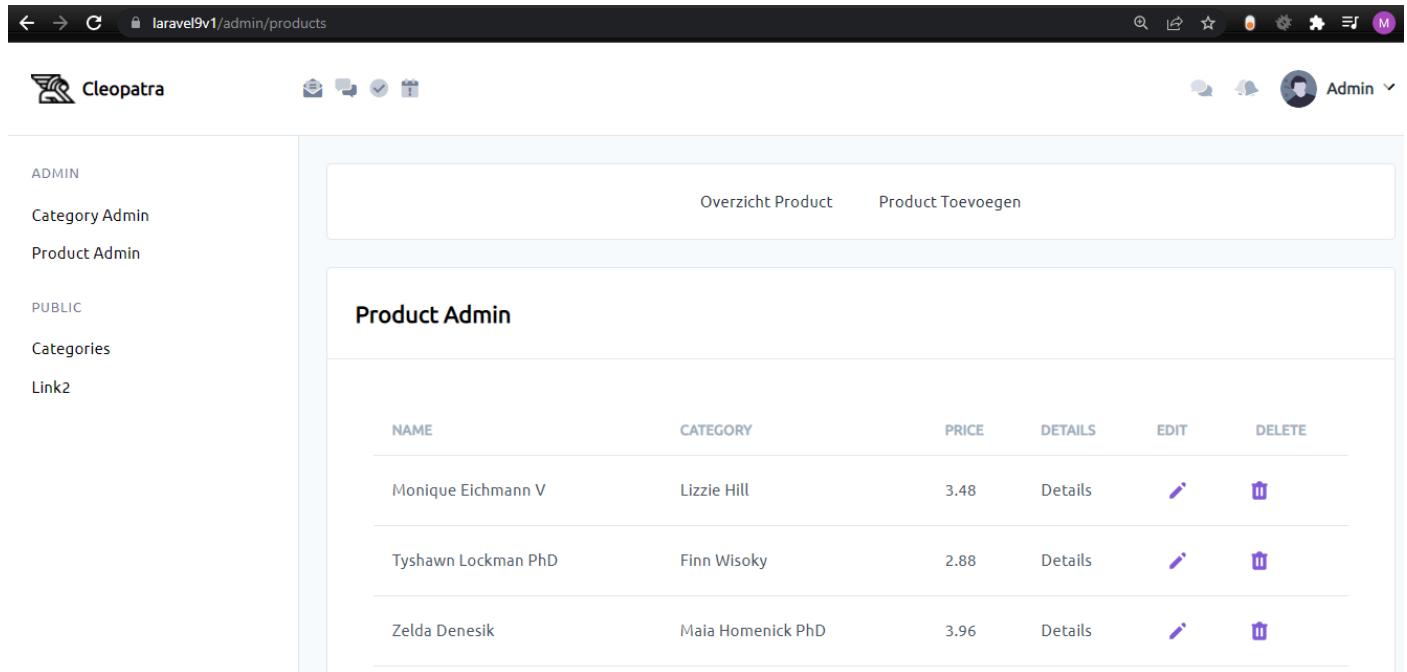
```
class ProductController extends Controller
{
    /**
     * Set permissions on methods
     */
    public function __construct()
    {
        $this->middleware('auth');

        $this->middleware('permission:index product', ['only' => ['index']]);
        $this->middleware('permission:show product', ['only' => ['show']]);
        $this->middleware('permission:create product', ['only' => ['create', 'store']]);
        $this->middleware('permission:edit product', ['only' => ['edit', 'update']]);
        $this->middleware('permission:delete product', ['only' => ['delete', 'destroy']]);
    }
}
```

Als we nu de seed opnieuw uitvoeren zijn de permissies geregeld.

```
PS D:\wamp\sites\laravel9v1> 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 (32.75ms)
```

En als admin zien we dan ook de delete vanzelf verschijnen bij de index van producten.



De index van producten is nu eindelijk af. Op naar de volgende onderdelen van de product CRUD!

3.9.5. Opdracht 16: Permissies voor tasks

Maak de volgende permissies aan.

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

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

Opdracht controle

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

```
.\vendor\bin\pest --group=Opdracht16
```

Er wordt nu alleen gecontroleerd of de rechten bij de index goed staan, de rest van de functionaliteiten worden later getest, waarbij de permissie ook meegenomen zal worden.

3.9.6. 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 drop down maken, waar je dan je categorie kan selecteren. Een drop down is leuk voor een aantal opties, maar zou je veel meer opties hebben zou een zoekfunctie makkelijk zijn. We kiezen nu voor een simpele drop down omdat we eerst met de basis bezig zijn en de zoekfunctie met bijv. javascript beter gemaakt zou kunnen worden. Voor de drop down hebben we wel alle categorieën nodig. Deze moeten we meesturen naar de view.

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

Let even op, dat de Category model dan wel wordt gebruikt, waardoor dit wel aangeroepen moet worden.

```
namespace App\Http\Controllers\Admin;  
  
use App\Http\Controllers\Controller;  
use App\Models\Category;  
use App\Models\Product;  
use App\Http\Requests\ProductStoreRequest;  
use App\Http\Requests\ProductUpdateRequest;  
use Illuminate\View\View;
```

Net zoals bij de index, kunnen we nu de create van category kopiëren en deze aanpassen.

De titel moet wel weer aangepast worden, en de routes van de linkjes in het menu. Dit zou je ondertussen zelf wel moeten kunnen.

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

```
!-- body -->


<form id="form" class="shadow-md rounded-lg px-8 pt-6 pb-8 mb-4"
      action="{{ route('products.store') }}" method="POST">
  @csrf
  <label class="block text-sm">
    <span class="text-gray-700">Name</span>
    <input class="bg-gray-200 block rounded w-full p-2 mt-1 focus:border-purple-400
      focus:outline-none focus:shadow-outline-purple form-input
      @error('name') border-red-600 focus:border-red-400 focus:shadow-outline-red @enderror"
      name="name" value="{{ old('name') }}" type="text" required>
  </label>
  <label class="block text-sm">
    <span class="text-gray-700">Description</span>
    <textarea
      class="bg-gray-200 shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
      focus:outline-none focus:shadow-outline @error('description') border-red-500 @enderror" name="description"
      id="description">{{ old('description') }}</textarea>
  </label>
  <label class="block text-sm">
    <span class="text-gray-700">Price</span>
    <input class="bg-gray-200 block rounded w-full p-2 mt-1 focus:border-purple-400
      focus:outline-none focus:shadow-outline-purple form-input
      @error('price') border-red-600 focus:border-red-400 focus:shadow-outline-red @enderror"
      name="price" value="{{ old('price') }}" type="text" required>
  </label>
  <label class="block text-sm">
    <span class="text-gray-700">Category</span>
    <select
      class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
      focus:outline-none focus:shadow-outline" name="category_id" id="category_id">
      @foreach($categories as $category)
        <option value="{{ $category->id }}" @selected($category->id == old('category_id'))>{{ $category->name }}</option>
      @endforeach
    </select>
  </label>

  <button class="mt-2 px-4 py-2 text-sm font-medium leading-5 text-white transition-colors duration-150
    bg-purple-600 border border-transparent rounded-lg active:bg-purple-600 hover:bg-purple-700
    focus:outline-none focus:shadow-outline-purple">Toevoegen</button>
</form>


```

Laten we per stuk van het formulier even bekijken.

De action aangepast naar products.store, want daar gaan we het formulier opvangen.

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

De productname, description en price kunnen gewone velden zijn.

```
<label class="block text-sm">
  <span class="text-gray-700">Name</span>
  <input class="bg-gray-200 block rounded w-full p-2 mt-1 focus:border-purple-400
  focus:outline-none focus:shadow-outline-purple form-input
  @error('name') border-red-600 focus:border-red-400 focus:shadow-outline-red @enderror"
        name="name" value="{{ old('name') }}" type="text" required>
</label>
<label class="block text-sm">
  <span class="text-gray-700">Description</span>
  <textarea
    class="bg-gray-200 shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
    focus:outline-none focus:shadow-outline @error('description') border-red-500 @enderror" name="description"
    id="description">{{ old('description') }}</textarea>
</label>
<label class="block text-sm">
  <span class="text-gray-700">Price</span>
  <input class="bg-gray-200 block rounded w-full p-2 mt-1 focus:border-purple-400
  focus:outline-none focus:shadow-outline-purple form-input
  @error('price') border-red-600 focus:border-red-400 focus:shadow-outline-red @enderror"
        name="price" value="{{ old('price') }}" type="text" required>
</label>
```

Bij de drop down willen we de id hebben van de categorie, omdat we deze straks als foreign key gaan opslaan. Als we valideren en iets is niet goed willen we wel de ingevulde waarde, net zoals de text inputs. In de vorige versies van Laravel moesten we het zo doen:

```
@foreach($categories as $category)
  <option value="{{ $category->id }}"
  @if( old('category_id') == $category->id)
    selected
  @endif
  >{{ $category->name }}</option>
@endforeach
```

In Laravel 9 zit nu een betere optie hiervoor, de @selected

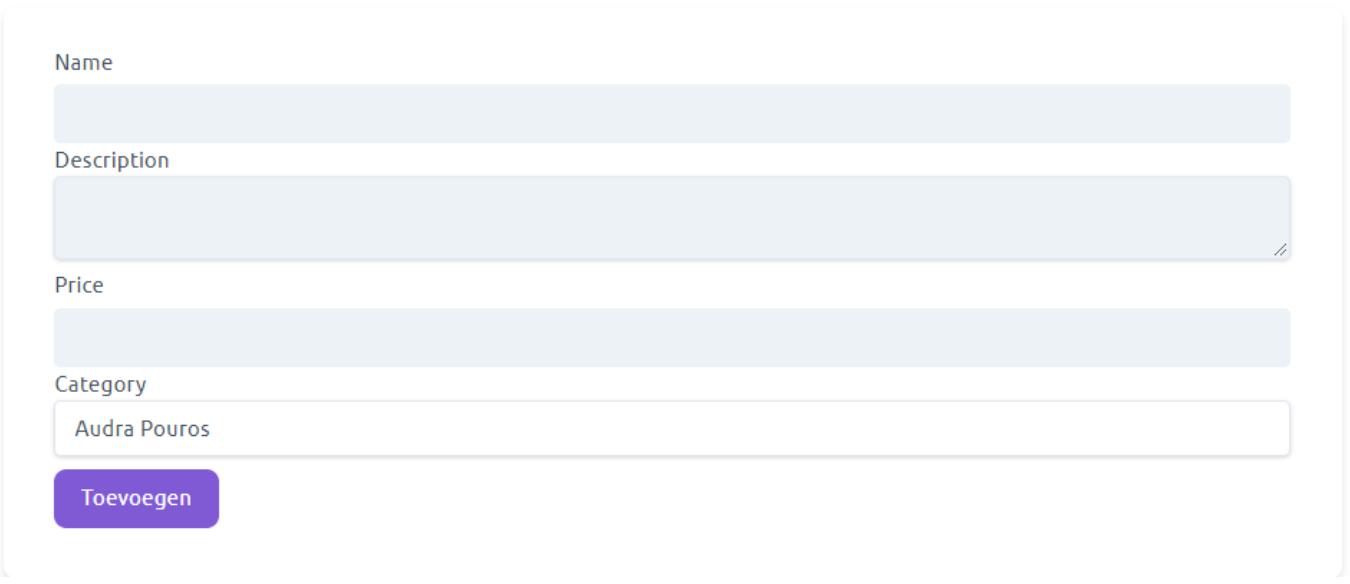
```
<label class="block text-sm">
    <span class="text-gray-700">Category</span>
    <select
        class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
        focus:outline-none focus:shadow-outline" name="category_id" id="category_id">
        @foreach($categories as $category)
            <option value="{{ $category->id }}" @selected($category->id == old('category_id'))>{{ $category->name }}</option>
        @endforeach
    </select>
</label>

<button class="mt-2 px-4 py-2 text-sm font-medium leading-5 text-white transition-colors duration-150
    bg-purple-600 border border-transparent rounded-lg active:bg-purple-600 hover:bg-purple-700
    focus:outline-none focus:shadow-outline-purple">Toevoegen</button>
</form>
```

De @selected is nu een onderdeel van Blade, net zoals bijv. @checked voor een check box. De code ziet er daardoor wat beter uit.

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

Product Admin



The screenshot shows a form for creating a new product. The fields are:

- Name: An input field.
- Description: A text area.
- Price: An input field.
- Category: A dropdown menu containing the option "Audra Pouros".
- A purple button labeled "Toevoegen" (Add).

Bij de category hebben we netjes een drop down gekregen. De create is dan ook hiermee klaar. Of de meldingen netjes komen te staan en de oude waardes erin komen als het niet goed is kunnen we pas testen als de validatie is geregeld.

3.9.7. 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.

```
public function store(ProductStoreRequest $request)
{
    // product toevoegen
    $product = new Product();
    $product->name = $request->name;
    $product->description = $request->description;
    $product->category_id = $request->category_id;
    $product->save();
}
```

Dit is nog niet zo bijzonder. Maar nu moeten we de prijs gaan opslaan. Als we een product aanmaken, gaan we ook meteen een prijs aanmaken, hierdoor wordt hiervoor ook een object aangemaakt.

```
public function store(ProductStoreRequest $request)
{
    // product toevoegen
    $product = new Product();
    $product->name = $request->name;
    $product->description = $request->description;
    $product->category_id = $request->category_id;
    $product->save();

    // prijs toevoegen gekoppeld met het product
    $price = new Price();
    $price->price = $request->price;
    $price->effdate = Carbon::now();
    $price->product_id = $product->id;
    $price->save();
}
```

Je ziet hier dat we met new Price() een object aanmaken. Verder gebruik ik Carbon voor een datum (<https://carbon.nesbot.com/>). Dit hebben we al een keer gebruikt bij de factory van product. Carbon zit al standaard in Laravel, dus ook erg makkelijk om te gebruiken. Zorg er wel voor dat het ook bij de use staat:

```
use App\Http\Controllers\Controller;
use App\Models\Category;
use App\Models\Price;
use App\Models\Product;
use App\Http\Requests\ProductStoreRequest;
use App\Http\Requests\ProductUpdateRequest;
use Carbon\Carbon;
use Illuminate\View\View;

class ProductController extends Controller
```

Voor de product_id gebruik ik natuurlijk de variabel \$product. Hier is een id nu beschikbaar omdat we al save() hebben gedaan.

Natuurlijk hebben we validatie nodig, maar doordat we de alle bestanden al met de model hebben aangemaakt is ook de request voor de store er al, en zelfs bij de store methode staat deze erbij:

```
public function store(ProductStoreRequest $request)
{
    $product = new Product();
```

Alleen als we nu de store zouden uitproberen komen we nooit door de validatie heen, terwijl de validatie leeg is.

Error

This Action Is Unauthorized.

Standaard staat namelijk de authorize() methode op false. Dus deze zetten we op true:

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

Bij de request moeten we altijd goed kijken naar wat er in de database kan. De naam van het product mag maximaal 45 karakters 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. Verder gaan we aangeven dat de description leeg mag zijn, maar als je het invult het wel minimaal 10 karakters moet zijn.

```
public function rules()
{
    return [
        'name' => 'required|max:45|unique:products',
        'description' => 'nullable|min:10',
        'price' => 'required|numeric|max:999999.99'
    ];
}
```

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

```
/*
 * Store a newly created resource in storage.
 *
 * @param ProductStoreRequest $request
 * @return RedirectResponse
 */
public function store(ProductStoreRequest $request): RedirectResponse
{
    // product toevoegen
    $product = new Product();
    $product->name = $request->name;
    $product->description = $request->description;
    $product->category_id = $request->category_id;
    $product->save();

    // prijs toevoegen gekoppeld met het product
    $price = new Price();
    $price->price = $request->price;
    $price->effdate = Carbon::now();
    $price->product_id = $product->id;
    $price->save();

    return to_route(route: 'products.index')->with('status', 'Product toegevoegd');
}
```

De request gebruiken we nu als validatie. Na het opslaan gaan we terug naar de index met een bericht dat het product is toegevoegd. We kunnen nu testen of dit werkt.

Bij de create vul ik dus wat testdata in het formulier.

Product Admin

Name

Testproduct

Description

Een stukje tekst over het product

Price

123.45

Category

Samir Kub

Toevoegen

Als we dit product toevoegen krijgen we ook netjes een bericht dat het product is toegevoegd.

Product Admin

Product Toegevoegd

En als je naar de laatste pagina in het overzicht van producten gaat, zie je het product in de index van producten.

Product Admin

NAME	CATEGORY	PRICE	DETAILS	EDIT	DELETE
Testproduct	Samir Kub	123.45	Details		

« Previous

Next »

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

Product Admin

- The Price Must Not Be Greater Than 999999.99.

Name

test1

Description

test111111111111

Price

1234567890

Category

Mrs. Elise Kreiger I

Toevoegen

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

3.9.8. Opdracht 17: Create & store voor tasks met validatie

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

Op de create zal een formulier beschikbaar moeten zijn die post naar de store methode. In het formulier zijn de volgende inputs:

- Task: text
- Begindate: Datum
- Enddate: Datum
- User_id: select
- Project_id: select
- Activity_id: select

De selects zullen alle drie een drop down moeten worden.

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

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

Als je het formulier hebt verstuurd kom je bij de store methode. Hier zal de data worden opgeslagen in de tabel tasks. Je moet gebruik maken van de named routes die bij een resource controller aangemaakt worden. Bij de validatie wordt gecontroleerd op de volgende voorwaarden:

- De task is ingevuld, minimaal 10 karakters, maximaal 200 karakters
- Begindate moet ingevuld zijn, enddate is niet verplicht.
- De user is niet verplicht. Er moet een project en activity aan gekoppeld staan en die moeten een integer zijn.

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

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

Opdracht controle

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

```
.\vendor\bin\pest --group=Opdracht17
```

3.9.9. Show

De show wordt niet zo moeilijk in de controller. We willen het Product hebben, die al in de variabel \$product staat vanwege de model binding. Hierdoor kan meteen het product naar de view toe.

```
/**  
 * Display the specified resource.  
 *  
 * @param Product $product  
 * @return View  
 */  
  
public function show(Product $product): View  
{  
    return view( view: 'admin.products.show', compact( var_name: 'product'));  
}
```

In de view hebben we de show van categorie gekopieerd en aangepast. Natuurlijk het menu weer naar products veranderd. Dan de card, waarin alles van het product komt. Je kan ervoor kiezen om alleen de huidige prijs te laten zien. Dit hebben we al in de index gebruikt, dus is nu weer te gebruiken.

```
<!-- content -->  


<h2 class="text-lg font-semibold text-gray-800">Product details</h2>  
    <p class="py-2 text-lg text-gray-700">Naam: {{ $product→name }}</p>  
    <p class="py-2 text-lg text-gray-700">Categorie: {{ $product→category→name }}</p>  
    <p class="py-2 text-lg text-gray-700">Beschrijving:<br> {{ $product→description }}</p>  
</div>


```

Maar waarom bewaren we nu de geschiedenis van prijzen dan. Is het dan niet handig om de prijzen van dit product dan hier te zien, zodat je het verloop kan bekijken. Dit gaan we dan ook doen.

Zouden we dan hiervoor alle prijzen die bij het product horen moeten ophalen?? Het antwoord is nee. Dit omdat we de relatie gewoon in onze model hebben, krijgen we een collection terug van prijzen.

```
// relation to Price  
public function prices()  
{  
    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.

```
<table class="w-full whitespace-no-wrap">
  <thead>
    <tr class="text-xs font-semibold tracking-wide text-left text-gray-500 uppercase border-b bg-gray-50">
      <th class="px-4 py-3">Id</th>
      <th class="px-4 py-3">Prijs</th>
      <th class="px-4 py-3">Datum</th>
    </tr>
  </thead>
  <tbody class="bg-white divide-y">
    @foreach($product->prices->sortByDesc('effdate') as $item)
      <tr class="text-gray-700">
        <td class="px-4 py-3 text-sm">{{ $item->id }}</td>
        <td class="px-4 py-3 text-sm">{{ $item->price }}</td>
        <td class="px-4 py-3 text-sm">{{ $item->effdate }}</td>
      </tr>
    @endforeach
  </tbody>
</table>
```

Je ziet in de @foreach dat ik van het product de relatie prices gebruik. De prijzen wil ik nog wel gesorteerd bij de datum wanneer deze ingaat, waarbij dan meeste recente datum bovenaan komt. Hiervoor gebruiken we dan de sortByDesc functie. Als we naar het resultaat gaan kijken ziet het er nu zo uit:

Product Admin

Product Details

Naam: Monique Eichmann V

Categorie: Lizzie Hill

Beschrijving:

Voluptatum Exercitationem Veniam Porro Impedit Eum Placeat Nam Ea.
Enim Ullam Et Dignissimos. Numquam Sed Repudiandae Cumque
Quibusdam. Pariatur Veritatis Possimus Et Tempore Aut Placeat Ab
Officia. Quod Et Aut Laboriosam Velit Cum Optio. Quae Saepe Officia
Dolore Doloremque. Voluptatibus Quis Ea Consequatur In Temporibus
Accusamus Eos. Quas Ut Illo Qui Et Voluptatem Omnis Distinctio Ut.
Dignissimos Vel Vitae Nobis Ratione Corrupti Earum Sit. Quasi Quo Minus
Mollitia. Dolor Consequatur Autem Facilis Delectus. Sunt Dolorum
Quidem Maiores In Illum. Esse Quia Est Ea Pariatur Consequatur. Fugit
Minima Ea Sint Natus. Est Iure Quas Atque Qui Iste Et Ut Laborum. Nisi
Doloribus Quae Accusantium Tenetur Facilis Qui. Nesciunt Maxime Fugit
Repellat Omnis Totam Ut. Dolorem Sed Ut Occaecati Sint.

ID	PRIJS	DATUM
1	3.48	2022-01-02 00:00:00
2	4.05	2021-07-01 00:00:00

Misschien in opmaak kan het beter, maar het resultaat is er. Lay-out kan altijd nog aangepast worden.

3.9.10. Opdracht 18: Show voor task

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

- Task id
- De task
- Begindatum
- Einddatum, indien beschikbaar
- De user met naam, indien beschikbaar
- De projectnaam
- De activity status
- Datum van het aanmaken van de task

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

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

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

Opdracht controle

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

```
.\vendor\bin\pest --group=Opdracht18
```

3.9.11. Edit

Voor de edit kunnen we gaan kijken wat we bij de create hebben gedaan en ook 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 drop down hebben voor categorie. Vanwege de drop down zullen we wel alle categories mee moeten sturen.

De controller gaat er dan zo uitzien.

```
/**  
 * Show the form for editing the specified resource.  
 *  
 * @param Product $product  
 * @return View  
 */  
  
public function edit(Product $product): View  
{  
    $categories = Category::all();  
    return view('admin.products.edit', compact('product', ...$categories));  
}
```

In de edit.blade kopiëren we als eerst wat we in de create hebben staan. Het menu is hetzelfde als bij de create, dus de wijzigingen zullen in het formulier zijn.

Als eerst zorgen we ervoor dat de action wordt veranderd naar products.update, waarbij we de id meegeven. De methode PUT zal er ook bij moeten.

```
<form id="form" class="shadow-md rounded-lg px-8 pt-6 pb-8 mb-4"  
      action="{{ route('products.update', ['product' => $product->id]) }}" method="POST">  
    @method('PUT')  
    @csrf  
    <label class="block text-sm">  
      <span class="text-gray-700">Name</span>  
      <input class="bg-gray-200 block rounded w-full p-2 mt-1 focus:border-purple-400  
            focus:outline-none focus:shadow-outline-purple form-input  
            @error('name') border-red-600 focus:border-red-400 focus:shadow-outline-red @enderror"  
            name="name" value="{{ old('name', $product->name) }}" type="text" required>  
    </label>
```

Je ziet ook bij de input dat ik een kleine wijziging heb gedaan ten opzichte van de create. In de create hebben we met old() gewerkt. We gebruiken dit natuurlijk weer, alleen hebben we nu ook te maken met data uit de database. Bij old() kan je eerst de oude data krijgen en als die er niet is, krijg je de database data. Erg handig voor nu!

In de rest van het formulier eigenlijk hetzelfde gedaan bij alle inputs.

```
<label class="block text-sm">
    <span class="text-gray-700">Description</span>
    <textarea
        class="bg-gray-200 shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
        focus:outline-none focus:shadow-outline @error('description') border-red-500 @enderror" name="description"
        id="description">{{ old('description', $product→description) }}</textarea>
</label>

<label class="block text-sm">
    <span class="text-gray-700">Price</span>
    <input class="bg-gray-200 block rounded w-full p-2 mt-1 focus:border-purple-400
    focus:outline-none focus:shadow-outline-purple form-input
    @error('price') border-red-600 focus:border-red-400 focus:shadow-outline-red @enderror"
    name="price" value="{{ old('price', $product→latest_price→price) }}" type="text" required>
</label>

<label class="block text-sm">
    <span class="text-gray-700">Category</span>
    <select
        class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
        focus:outline-none focus:shadow-outline" name="category_id" id="category_id">
        @foreach($categories as $category)
            <option value="{{ $category→id }}"
                @selected($category→id == old('category_id', $product→category_id))>{{ $category→name }}</option>
        @endforeach
    </select>
</label>

<button class="mt-2 px-4 py-2 text-sm font-medium leading-5 text-white transition-colors duration-150
bg-purple-600 border border-transparent rounded-lg active:bg-purple-600 hover:bg-purple-700
focus:outline-none focus:shadow-outline-purple">Wijzigen</button>
</form>
```

Als we dit uitproberen zal je bij de edit een net ingevuld formulier zien.

Product Admin

Name



Description

Voluptatum exercitationem veniam porro impedit eum placeat nam ea. Enim ullam et dignissimos. Numquam sed repudiandae cumque quibusdam. Pariatur veritatis possimus et tempore aut placeat ab officia. Quid et aut laborescam volit sum optio.

Price

Category

Wijzigen

Een net ingevuld formulier. Dat bijv. de description nu een te klein veld is, is alleen even een design stukje.

3.9.12. 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. Dus als eerst zorgen we ervoor dat alles van de tabel products geüpdate zal worden.

```
public function update(ProductUpdateRequest $request, Product $product)
{
    // er is al een bestaand $product
    $product->name = $request->name;
    $product->description = $request->description;
    $product->category_id = $request->category_id;
    $product->save();
}
```

Hieronder kunnen we dan een controle doen, dat als de opgestuurde prijs niet gelijk is aan de huidige prijs in de database, de opgestuurde prijs in de database wordt opgeslagen.

```
// bij de update gaan we een nieuwe prijs erin zetten (dus niet een oude waarde aanpassen)
// Alleen uitvoeren als er een nieuwe prijs is ingevuld, dus niet met de huidige prijs.

if($product->latest_price->price != $request->price)
{
    $price = new Price();
    $price->price = $request->price;
    $price->effdate = Carbon::now();
    $price->product_id = $product->id;
    $price->save();
}
```

Zoals je ziet, pakken we de latest_price() methode om die te vergelijken met de request. Door alle plugins in PhpStorm zie je niet de !== maar een = met een streep erdoor. We gebruiken natuurlijk !== om ervoor te zorgen dat het datatype ook gelijk moet zijn. Daarnaast updaten we de prijs niet, maar maken een nieuwe aan door gewoon een nieuw object te maken. Hierdoor staat de oude prijs ook nog in de database en hebben we een geschiedenis van de prijs.

```
/**
 * Update the specified resource in storage.
 *
 * @param ProductUpdateRequest $request
 * @param Product $product
 * @return RedirectResponse
 */
public function update(ProductUpdateRequest $request, Product $product): RedirectResponse
{
    // er is al een bestaand $product
    $product->name = $request->name;
    $product->description = $request->description;
    $product->category_id = $request->category_id;
    $product->save();

    // bij de update gaan we een nieuwe prijs erin zetten (dus niet een oude waarde aanpassen)
    // Alleen uitvoeren als er een nieuwe prijs is ingevuld, dus niet met de huidige prijs.

    if($product->latest_price->price == $request->price)
    {
        $price = new Price();
        $price->price = $request->price;
        $price->effdate = Carbon::now();
        $price->product_id = $product->id;
        $price->save();
    }

    return to_route('products.index')->with('status', 'Product gewijzigd');
}
```

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.

Net zoals bij de store, moeten we nu eerst de request gaan regelen voordat we het kunnen uittesten. In de update request zorgen we er dan voor dat hij niet naar de huidige naam van het product kijkt. De rest is hetzelfde als de store request.

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

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

        return [
            'name' => 'required|max:45|unique:products,name,' . $product->id,
            'description' => 'nullable|min:10',
            'price' => 'required|numeric|max:999999.99'
        ];
    }
}
```

Nu kunnen we de edit en update uitproberen. We veranderen alles wat bij id=1 stond, zodat we bovenaan de index meteen zien of ook alles is veranderd, en dat is zo.

Product Admin

The screenshot shows a Laravel application's admin interface for managing products. At the top, a green success message box displays "Product Gewijzigd". Below it is a table with columns: NAME, CATEGORY, PRICE, DETAILS, EDIT, and DELETE. Two rows of data are visible:

NAME	CATEGORY	PRICE	DETAILS	EDIT	DELETE
EditTest	Amya Feil	1.11	Details		
Tyshawn Lockman PhD	Finn Wisoky	2.88	Details		

Als we dan proberen een error te krijgen, door een te grote prijs te gebruiken. Express de product naam van id=1 gekozen om bij product met id = 2 neer te zetten. Verder de prijs groter dan het maximum te zetten.

Het resultaat wat we willen. Een foutmelding met wat er anders moet.

Product Admin

The Name Has Already Been Taken.
The Price Must Not Be Greater Than 999999.99.

Name
Edittest 

Description

Price
123456789

Category
Garrick Yost

Wijzigen

De edit en update zijn nu geheel klaar.

3.9.13. Opdracht 19: Edit & update voor tasks

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

Op de update zal een formulier beschikbaar moeten zijn die post (+ PUT/PATCH) naar de update methode. In het formulier zijn de volgende inputs die al ingevuld moeten zijn:

- Task: text
- Begindate: Datum
- Enddate: Datum
- User_id: select
- Project_id: select
- Activity_id: select

De selects zullen alle drie een drop down moeten worden, waarbij de huidige waarde geselecteerd is.

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

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

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

- De task is ingevuld, minimaal 10 karakters, maximaal 200 karakters
- Begindate moet ingevuld zijn, enddate is niet verplicht.
- De user is niet verplicht. Er moet een project en activity status aan gekoppeld staan en die moeten een integer zijn.

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

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

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

Opdracht controle

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

```
.\vendor\bin\pest --group=Opdracht19
```

3.9.15. Delete

Voor het verwijderen van producten gaan we aan de slag met de delete, net zoals bij categorie. Dit omdat we toch weer een bevestiging willen maken dat we het juiste product verwijderen.

De methode moet dan natuurlijk weer in de controller worden toegevoegd.

```
/**  
 * Show the form for deleting the specified resource.  
 *  
 * @param Product $product  
 * @return View  
 */  
  
public function delete(Product $product): View  
{  
    return view( view: 'admin.products.delete', compact( var_name: 'product'));  
}
```

In de controller kunnen we meteen het product naar het formulier sturen.

Voor de view gaan we de edit.blade kopiëren naar de delete.blade, want we hebben een ingevuld formulier nodig.

Voor het formulier zorgen we dat de velden disabled zijn. Voor de categorie hebben we geen drop down nodig, dus dit zou gewoon in een input kunnen. De old() methode hebben we ook niet nodig, omdat we hier sowieso niks kunnen wijzigen.

Let even op dat je de route en @method nog even veranderd!

```
<form id="form" class="shadow-md rounded-lg px-8 pt-6 pb-8 mb-4"  
      action="{{ route('products.destroy', ['product' => $product->id]) }}" method="POST">  
    @method('DELETE')  
    @csrf  
    <label class="block text-sm">  
      <span class="text-gray-700">Name</span>  
      <input class="bg-gray-200 block rounded w-full p-2 mt-1 focus:border-purple-400  
            focus:outline-none focus:shadow-outline-purple form-input"  
            name="name" value="{{ $product->name }}" type="text" disabled/>  
    </label>  
    <label class="block text-sm">  
      <span class="text-gray-700">Description</span>  
      <textarea  
        class="bg-gray-200 shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight  
        focus:outline-none focus:shadow-outline" name="description"  
        id="description" disabled>{{ $product->description }}</textarea>  
    </label>
```

```

<label class="block text-sm">
    <span class="text-gray-700">Price</span>
    <input class="bg-gray-200 block rounded w-full p-2 mt-1 focus:border-purple-400
    focus:outline-none focus:shadow-outline-purple form-input"
        name="price" value="{{ $product->latest_price->price }}" type="text" disabled/>
</label>

<label class="block text-sm">
    <span class="text-gray-700">Category</span>
    <input class="bg-gray-200 block rounded w-full p-2 mt-1 focus:border-purple-400
    focus:outline-none focus:shadow-outline-purple form-input"
        name="category_id" value="{{ $product->category->name }}" type="text" disabled/>
</label>

<button class="mt-2 px-4 py-2 text-sm font-medium leading-5 text-white transition-colors duration-150
bg-purple-600 border border-transparent rounded-lg active:bg-purple-600 hover:bg-purple-700
focus:outline-none focus:shadow-outline-purple">Verwijderen</button>
</form>

```

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

Product Admin

The screenshot shows a modal dialog box for deleting a product. It contains the following fields:

- Name:** EditTest
- Description:** Dit was product met id=1
- Price:** 1.11
- Category:** Amya Feil

At the bottom of the form is a purple button labeled "Verwijderen" (Delete).

3.9.16. Destroy

Het verwijderen van een product is op zich simpel. We zeggen gewoon, delete().

```
/**  
 * Remove the specified resource from storage.  
 *  
 * @param Product $product  
 * @return RedirectResponse  
 */  
  
public function destroy(Product $product): RedirectResponse  
{  
    $product->delete();  
    return to_route('products.index')->with('status', 'Product verwijderd');  
}
```

Verder zorgen we natuurlijk dat we een bericht krijgen dat het product is verwijderd.

Als we dit dan uittesten, zien we dat het product verwijderd is en we weer op de index zijn. Het product met id = 1 is nu verwijderd, wat ook de bedoeling was.

Product Admin

Product Verwijderd					
Name	Category	Price	Details	Edit	Delete
Tyshawn Lockman PhD	Finn Wisoky	2.88	Details		
Zelda Denesik	Maia Homenick PhD	3.96	Details		

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

Weergave van records 0 - 24 (399 totaal, Query duurde 0,0003 seconden.)
<u>SELECT</u> * FROM `prices`
<input type="checkbox"/> Profiling [Inline bewerken] [Wijzigen] [Verklaar SQL] [Genereer PHP-code] [Ververs]
1 < > >> <input type="checkbox"/> Toon alles Aantal rijen: 25 Filter rijen: <input type="text"/> Zoek in deze tabel Sorteren op sleutel: Ge
+ Opties
id price effdate product_id created_at updated_at
3 3.13 2021-03-16 00:00:00 2 2022-03-09 15:07:49 2022-03-09 15:07:49
4 2.94 2021-10-02 00:00:00 2 2022-03-09 15:07:49 2022-03-09 15:07:49
5 5.48 2021-05-25 00:00:00 3 2022-03-09 15:07:49 2022-03-09 15:07:49
6 3.96 2021-07-06 00:00:00 3 2022-03-09 15:07:49 2022-03-09 15:07:49

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



```
return new class extends Migration
{
    /**
     * Run the migrations. ...
     */
    public function up()
    {
        Schema::create('prices', function (Blueprint $table) {
            $table->id();
            $table->decimal('price', 8, 2);
            $table->dateTime('effdate');
            $table->foreignId('product_id')->constrained()
                ->onDelete('cascade');
            $table->timestamps();
        });
    }
}
```

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

3.9.17. Opdracht 20: Delete & destroy voor tasks

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

Op de delete zal een formulier beschikbaar moeten zijn die post (+ DELETE) naar de destroy methode. In het formulier zijn de volgende inputs die al ingevuld moeten zijn:

- Task: text
- Begindate: Datum
- Enddate: Datum
- User_id: select
- Project_id: select
- Activity_id: select

De selects zullen alle drie een drop down moeten worden, waarbij alleen de huidige waarde een optie is. Alle inputs moeten disabled zijn.

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

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

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

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

Opdracht controle

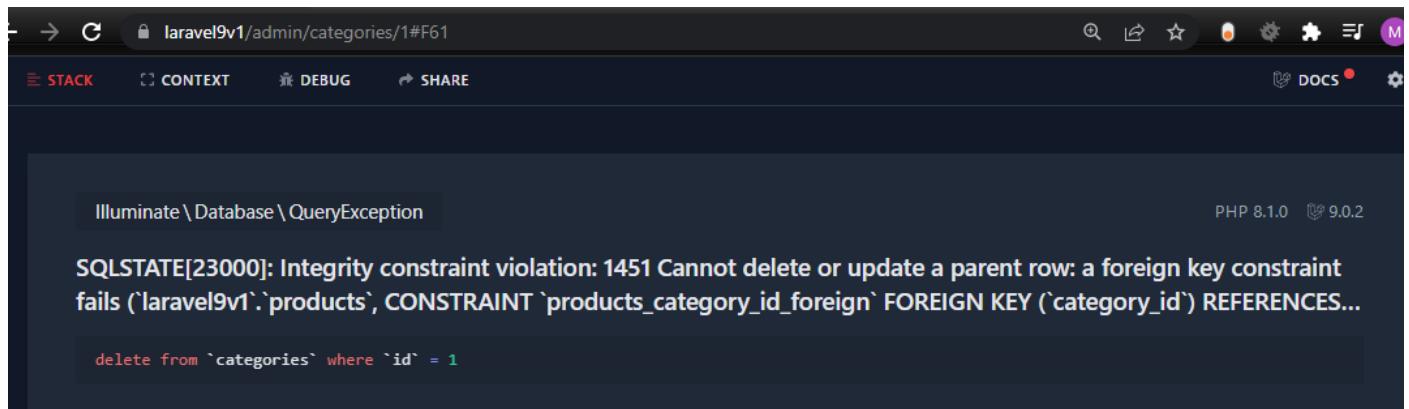
Als je de opdracht af hebt en gebruik hebt gemaakt van de opdrachten repository, kan je het volgende commando gebruiken om te controleren of alles goed werkt:

```
.\vendor\bin\pest --group=Opdracht20
```

3.9.18. 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 erachter dat de delete van een categorie het niet meer doet, zodra er een product aan gekoppeld is. Bij het verwijderen krijgen we deze error.



```
Illuminate\Database\QueryException
SQLSTATE[23000]: Integrity constraint violation: 1451 Cannot delete or update a parent row: a foreign key constraint fails ('laravel9v1`.`products', CONSTRAINT `products_category_id_foreign` FOREIGN KEY (`category_id`) REFERENCES...
delete from `categories` where `id` = 1
```

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

In de product migration staat dit



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

Dit betekent, dat als er nog een product aan een category vast zit, de relatie zegt: restrict.

Je mag op dat moment dus niet de categorie verwijderen, omdat we anders producten krijgen die in een niet bestaande categorie zitten. Dit kan natuurlijk niet.

Wel is het netjes, om in ieder geval een nette foutmelding te krijgen, waarin staat dat er bijvoorbeeld nog producten aan de categorie staan gekoppeld. In de categorie controller gaan we dan ook de destroy aanpassen.

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.

```
/**  
 * Remove the specified resource from storage.  
 *  
 * @param Category $category  
 * @return RedirectResponse  
 */  
  
public function destroy(Category $category): RedirectResponse  
{  
    try {  
        $category->delete();  
    } catch (Throwable $error)  
    {  
        report($error);  
        return to_route('categories.index')->with(['status' => 'Categorie is niet leeg.  
        Er mogen geen producten meer in de categorie staan wanneer deze verwijderd wordt']);  
    }  
    return to_route('categories.index')->with(['status' => 'Categorie verwijderd']);  
}
```

We gebruiken ook een report, om ervoor te zorgen dat de error in een log komt. We gebruiken hiervoor de class Throwable. Let even op dat deze ook bovenin bij de use moet komen.

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

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

Category Admin

Categorie Is Niet Leeg. Er Mogen Geen Producten Meer In De Categorie Staan
Wanneer Deze Verwijderd Wordt

Alles werkt nu weer zoals het de bedoeling is. Tot zover Laravel 9 level 2