Afbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijving

Instructie crud

Gemaakt door Ryan H.

Datum 06/09/2023

Inhoud **(LEES GOED DE !!BELANGRIJK!! KOPJES)**

[**Extensies die je moet hebben op visual studio code (!!BELANGRIJK!!)** 5](#_Toc151858249)

[Git instructies (documentatie) 6](#_Toc151858250)

[Branch maken 6](#_Toc151858251)

[Klonen 6](#_Toc151858252)

[Pushen 6](#_Toc151858253)

[Pullen 7](#_Toc151858254)

[Migraten en seeden 7](#_Toc151858255)

[**Tailwind hulpbron (documentaties)** **(!!BELANGRIJK!!)** 8](#_Toc151858256)

[Breakpoints (als je responsive wil maken) 8](#_Toc151858257)

[Color 8](#_Toc151858258)

[Flex 8](#_Toc151858259)

[Img 8](#_Toc151858260)

[Background img 8](#_Toc151858261)

[Padding 8](#_Toc151858262)

[Scroll padding 8](#_Toc151858263)

[Width & height 8](#_Toc151858264)

[Margin 9](#_Toc151858265)

[Tailwind cheatsheet 9](#_Toc151858266)

[**Laravel hulpbron (one-to-many + many-to-many Documentatie) (!!BELANGRIJK!!)** 10](#_Toc151858267)

[One-to-one 10](#_Toc151858268)

[One-to-many 10](#_Toc151858269)

[Many-to-many 10](#_Toc151858270)

[Datadumps 10](#_Toc151858271)

[Controller 10](#_Toc151858272)

[Blade 10](#_Toc151858273)

[**Model voorbeeld** **(!!BELANGRIJK!!)** 11](#_Toc151858274)

[Uitleg van gebruikte relatie-methoden 11](#_Toc151858275)

[Many-to-Many Model uitleg 12](#_Toc151858276)

[One-to-Many Model uitleg 13](#_Toc151858277)

[Laravel: Wat doet wat? 14](#_Toc151858278)

[Modellen 14](#_Toc151858279)

[Migraties 14](#_Toc151858280)

[Seeders 14](#_Toc151858281)

[Controllers 14](#_Toc151858282)

[web.php (route) 14](#_Toc151858283)

[Resources > css 14](#_Toc151858284)

[Resources > view 15](#_Toc151858285)

[Algemeen, hoe maak je een crud? (Stappenplan) 17](#_Toc151858286)

[Beginsituatie, hoe begin ik? 18](#_Toc151858287)

[Bezoekers kunnen wel website bezoeken zonder inloggen (4.1) 19](#_Toc151858288)

[Bezoekers kunnen niet de website bezoeken zonder inloggen (4.1.1) 24](#_Toc151858289)

[**Maken van one-to-many crud (Beste manier) (!!BELANGRIJK!!)** 27](#_Toc151858290)

[Voordat je verder gaat 27](#_Toc151858291)

[Info wat wij gaan maken 27](#_Toc151858292)

[ERD 27](#_Toc151858293)

[Design 27](#_Toc151858294)

[CRUD Applicatie Structuur Opzetten 28](#_Toc151858295)

[Maken van models 29](#_Toc151858296)

[Het maken van migrations 31](#_Toc151858297)

[Het maken van factories 33](#_Toc151858298)

[Het maken van seeders 35](#_Toc151858299)

[Het maken van crud routes 38](#_Toc151858300)

[Het maken van blades & controllers 41](#_Toc151858301)

[Controller klaargezette voor pagina inzien 41](#_Toc151858302)

[Navigatie aanpassen 42](#_Toc151858303)

[Toestaan van footers in je index 46](#_Toc151858304)

[Alle blades maken 48](#_Toc151858305)

[[AUTHORS] Index maken (blade & controller) 49](#_Toc151858306)

[[AUTHORS] create maken (blade & controller) 51](#_Toc151858307)

[[AUTHORS] edit maken (blade & controller) 53](#_Toc151858308)

[[AUTHORS] destroy maken (blade & controller) 55](#_Toc151858309)

[[AUTHORS] show maken (blade & controller) 56](#_Toc151858310)

[[BOOKS] Index maken (blade & controller) 57](#_Toc151858311)

[[BOOKS] show maken (blade & controller) 59](#_Toc151858312)

[[BOOKS] create maken (blade & controller) 60](#_Toc151858313)

[[BOOKS] edit maken (blade & controller) 63](#_Toc151858314)

[[BOOKS] destroy maken (blade & controller) 65](#_Toc151858315)

[**Maken van many-to-many crud (Beste manier) (!!BELANGRIJK!!)** 67](#_Toc151858316)

[Voordat je verder gaat 67](#_Toc151858317)

[Info wat wij gaan maken 67](#_Toc151858318)

[ERD 67](#_Toc151858319)

[Design 67](#_Toc151858320)

[CRUD Applicatie Structuur Opzetten 68](#_Toc151858321)

[Maken van models 69](#_Toc151858322)

[Het maken van migrations 71](#_Toc151858323)

[Het maken van factories 75](#_Toc151858324)

[Het maken van seeders 77](#_Toc151858325)

[Het maken van crud routes 81](#_Toc151858326)

[Controller klaargezette voor pagina inzien 84](#_Toc151858327)

[Navigatie aanpassen 85](#_Toc151858328)

[Toestaan van footers in je index 89](#_Toc151858329)

[Alle blades maken 91](#_Toc151858330)

[[ARTISTS] Index maken (blade & controller) 92](#_Toc151858331)

[[ARTISTS] create maken (blade & controller) 94](#_Toc151858332)

[[ARTISTS] edit maken (blade & controller) ) 97](#_Toc151858333)

[[ARTISTS] destroy maken (blade & controller) ) 99](#_Toc151858334)

[[ARTISTS] show maken (blade & controller) ) 100](#_Toc151858335)

[[ARTWORKS] Index maken (blade & controller) 101](#_Toc151858336)

[[ARTWORKS] create maken (blade & controller) 103](#_Toc151858337)

[[ARTWORKS] edit maken (blade & controller) 105](#_Toc151858338)

[[ARTWORKS] destroy maken (blade & controller) 107](#_Toc151858339)

[[ARTWORKS] show maken (blade & controller) 108](#_Toc151858340)

[Alternatieve versie van het maken voor seeders/models/seeders **(Suggesteer dit niet)** 109](#_Toc151858341)

[Migration maken 109](#_Toc151858342)

[Het maken van models 112](#_Toc151858343)

[Seeders aanmaken 118](#_Toc151858344)

# **Extensies die je moet hebben op visual studio code (!!BELANGRIJK!!)**

Afbeelding met tekst, schermopname, ontwerp

Automatisch gegenereerde beschrijving

# Git instructies (documentatie)

## Branch maken

Branch aanmaken doe je met:

git branch naam

git checkout naam

Testen ofdat je op je gemaakte branch zit doe je met:

git branch

## Klonen

klonen een kopie van het project van de server naar je computer met de volgende manier:

git clone https://PS195828@dev.azure.com/PS195828/Project\_6/\_git/Project\_6

## Pushen

Druk je wijzigingen naar de server met de volgende manier:

git add --all

git commit

git push (hier gitlink)

Als dit niet werkt, kun je de volgende manier:

git commit

git add --all

git push (hier gitlink)

## Pullen

Download de nieuwste wijzigingen van de server met de volgende manier:

git pull (hier gitlink)

composer install

composer update

npm install

npm run dev -- terminal 1 window

php artisan serve -- terminal 2 window (weet zeker dat localhost:8000 is in .env file in laravel)

Pullen van master en zetten op je branch

Zorg ervoor dat je branch up-to-date is met de master branch met de volgende manier:

git checkout branch2

git pull origin master

## Migraten en seeden

Seeden en migration van laravel doe je met

php artisan migrate:fresh --seed

# **Tailwind hulpbron (documentaties)** **(!!BELANGRIJK!!)**

## Breakpoints (als je responsive wil maken)

<https://tailwindcss.com/docs/responsive-design>

(sm, md, lg, xl, 2xl)

**Je begint altijd klein**

## Color

<https://tailwindcss.com/docs/text-color>

## Flex

Flex-wrap is altijd handig met meerdere items op een pagina te krijgen

<https://tailwindcss.com/docs/flex>

## Img

Object-cover regelt goed dat de afbeelding niet uitgerekt wordt maar wel de afbeelding komt goed uit te zien.

<https://tailwindcss.com/docs/object-fit>

## Background img

<https://tailwindcss.com/docs/background-image>

## Padding

<https://tailwindcss.com/docs/padding>

## Scroll padding

**Gebruik je haast niet, maar kan**

<https://tailwindcss.com/docs/scroll-padding>

## Width & height

Breedte aanpassen:

<https://tailwindcss.com/docs/width>

Height aanpassen:

<https://tailwindcss.com/docs/height>

## Margin

**Gebruik bijna niet, gebruik in plaats daarvan padding**

<https://tailwindcss.com/docs/margin>

## Tailwind cheatsheet

Kan je de begrippen makkelijk inzoeken wel zonder voorbeeld

<https://tailwindcomponents.com/cheatsheet/>

# **Laravel hulpbron (one-to-many + many-to-many Documentatie) (!!BELANGRIJK!!)**

**LEES GOED DOOR**

## One-to-one

<https://laravel.com/docs/10.x/eloquent-relationships#one-to-one>

Hasone

Belongsto

## One-to-many

<https://laravel.com/docs/10.x/eloquent-relationships#one-to-many>

## Many-to-many

<https://laravel.com/docs/10.x/eloquent-relationships#many-to-many>

## Datadumps

### Controller

dd($naam)

### Blade

@dd($naam)

# **Model voorbeeld** **(!!BELANGRIJK!!)**

## Uitleg van gebruikte relatie-methoden

**belongsTo**: Dit wordt gebruikt om een "many-to-one" relatie te definiëren. Bijvoorbeeld, in het "Book" model, is er een "belongsTo" relatie met het "Author" model, wat betekent dat elk boek hoort bij één auteur.

**belongsToMany**: Dit wordt gebruikt om een "many-to-many" relatie te definiëren. Bijvoorbeeld, in het "Student" model en het "Course" model, is er een "belongsToMany" relatie om aan te geven dat studenten meerdere cursussen kunnen volgen, en elke cursus kan door meerdere studenten worden gevolgd.

**hasMany**: Dit wordt gebruikt om een "one-to-many" relatie te definiëren. Bijvoorbeeld, in het "Author" model, is er een "hasMany" relatie met het "Book" model, wat betekent dat een auteur meerdere boeken kan schrijven.

**hasOne**: Dit wordt gebruikt om een "one-to-one" relatie te definiëren, waarbij elk record in het ene model aan slechts één record in het andere model is gekoppeld. Dit werd niet in de voorbeelden gebruikt, maar het is vergelijkbaar met belongsTo in termen van relaties, maar met een beperking dat elk record in het ene model slechts één record in het andere model heeft.

**Lees goed door many-to-many model uitleg en one-to-many model uitleg**

## Many-to-Many Model uitleg

Laten we een "Student" model en een "Course" model maken met een "many-to-many" relatie. Studenten kunnen zich inschrijven voor meerdere cursussen, en elke cursus kan door meerdere studenten worden gevolgd.

In dit voorbeeld kunnen studenten zich inschrijven voor meerdere cursussen, en elke cursus kan door meerdere studenten worden gevolgd.

Student Model:

|  |
| --- |
| namespace App\Models;  use Illuminate\Database\Eloquent\Model;  use Illuminate\Database\Eloquent\Relations\BelongsToMany;  class Student extends Model  {  protected $fillable = [  'name',  'email',  ];  public function courses(): BelongsToMany  {  return $this->belongsToMany(Course::class);  }  } |

Course Model:

|  |
| --- |
| namespace App\Models;  use Illuminate\Database\Eloquent\Model;  use Illuminate\Database\Eloquent\Relations\BelongsToMany;  class Course extends Model  {  protected $fillable = [  'title',  'description',  ];  public function students(): BelongsToMany  {  return $this->belongsToMany(Student::class);  }  } |

## One-to-Many Model uitleg

Laten we nu een "Author" model en een "Book" model maken met een "one-to-many" relatie. Een auteur kan meerdere boeken schrijven, maar een boek heeft slechts één auteur.

In dit voorbeeld kan een auteur meerdere boeken schrijven, maar elk boek behoort tot slechts één auteur.

**Author Model:**

|  |
| --- |
| namespace App\Models;  use Illuminate\Database\Eloquent\Model;  use Illuminate\Database\Eloquent\Relations\HasMany;  class Author extends Model  {  protected $fillable = [  'name',  'email',  ];  public function books(): HasMany  {  return $this->hasMany(Book::class);  }  } |

**Book Model:**

|  |
| --- |
| namespace App\Models;  use Illuminate\Database\Eloquent\Model;  use Illuminate\Database\Eloquent\Relations\BelongsTo;  class Book extends Model  {  protected $fillable = [  'title',  'published\_date',  ];  public function author(): BelongsTo  {  return $this->belongsTo(Author::class);  }  } |

# Laravel: Wat doet wat?

Laravel is een populaire PHP-framework voor het bouwen van webtoepassingen. Het framework biedt een aantal hulpmiddelen en functies om het ontwikkelen van applicaties te vereenvoudigen.

## Modellen

Een model vertegenwoordigt een databasetabel en de bijbehorende operaties voor gegevensbeheer en -interactie. Modellen bieden een objectgeoriënteerde interface om met de database te communiceren.

## Migraties

Migraties worden gebruikt om de databaseschemastructuur te creëren en te wijzigen. Migraties stellen je in staat om de databaseschema te beheren en wijzigingen geleidelijk toe te passen.

## Seeders

Seeders worden gebruikt om de database te vullen met initiële of voorbeeldgegevens. Seeders zijn handig voor testen en het bieden van een startpunt voor de data in je applicatie.

## Controllers

Een controller bevat de logica en functies om HTTP-verzoeken te verwerken en de juiste respons te genereren. Controllers ontvangen de verzoeken, voeren de benodigde acties uit met behulp van de modellen en genereren geschikte antwoorden.

## web.php (route)

Het web.php-bestand definieert URL-routes en hun bijbehorende controlleracties. Dit bestand dient als het startpunt voor het routeren van HTTP-verzoeken in Laravel.

## Resources > css

Het Resources > css-map is waar je CSS-bestanden kunt opslaan. CSS vraagt wel npm. Installeer deze dan ook in je Laravel-applicatie.

## Resources > view

Het Resources > view-map is waar je alle blade-bestanden kunt opslaan. Dit zijn je websitepagina's.

# Algemeen, hoe maak je een crud? (Stappenplan)

1. Definieer je database schema en tabellen: Maak de benodigde migraties aan om de structuur van je database tabellen te definiëren. Migraties stellen je in staat om de database schema te beheren en wijzigingen geleidelijk toe te passen.
2. Maak de modellen aan: Nadat je migraties zijn opgezet, maak je de bijbehorende modellen aan die je database tabellen vertegenwoordigen. Modellen bieden een objectgeoriënteerde interface om met de database te communiceren en relaties tussen tabellen te definiëren.
3. Implementeer de seeders: Nadat je de modellen hebt gemaakt, kun je seeders aanmaken om je database te vullen met initiële of voorbeeldgegevens. Seeders zijn handig voor testen en het bieden van een startpunt voor de data in je applicatie.
4. Maak de controllers aan: Vervolgens maak je controllers aan om de logica van je applicatie te beheren en inkomende HTTP-verzoeken te verwerken. Controllers ontvangen de verzoeken, voeren de benodigde acties uit met behulp van de modellen en genereren geschikte antwoorden.
5. Definieer routes in web.php: Ten slotte definieer je de routes in het web.php-bestand, dat dient als het startpunt voor het routeren van HTTP-verzoeken in Laravel. Wijs de routes toe aan de bijbehorende controlleracties om de verbinding tussen URL's en applicatiefuncties tot stand te brengen.

# Beginsituatie, hoe begin ik?

1. Ga naar een folder en typ in de search **cmd** en dan op **enter**

Afbeelding met tekst, schermopname, lijn, Multimediasoftware

Automatisch gegenereerde beschrijving

1. Over de volgende code uit im een laravel app te maken

|  |
| --- |
| composer create-project laravel/laravel example-app |

1. Na het maken van je laravel app voer de volgende code uit om composer te installeren en project te starten:

|  |
| --- |
| * laravel new example * composer require laravel/breeze --dev * php artisan breeze:install   Blade with Alpine .......................................................................................................................... blade  Would you like dark mode support? (yes/no) [no]  ❯ no  Which testing framework do you prefer? [PHPUnit]  PHPUnit ........................................................................................................................................ 0  Pest ........................................................................................................................................... 1  ❯ 0   * php artisan migrate * composer install * composer update * npm install * php artisan migrate:fresh * npm run dev -- terminal 1 window * php artisan serve -- terminal 2 window (weet zeker dat localhost:8000 is in .env file in laravel) |
| * Ga naar config folder en open database.php * Ga naar mysql en verander engine => null naar InnoDB   **'engine' => 'InnoDB',** |

1. **Nu moet je kiezen ofdat je login only wilt ga dan naar 4.1. Als bezoekers ook jouw website mogen bezoeken zonder login gegevens ga dan naar 4.1.1.**

## Bezoekers kunnen wel website bezoeken zonder inloggen (4.1)

* 1. Je koos ervoor dat bezoekers jouw website mogen bezoeken
  2. Start met het vervangen van de welcome route met jouw persoonlijke pagina, bv:

|  |
| --- |
| Route::get('/', function () {      return view('welcome');  })->name('frontpage'); |

3.3 voer de volgende command uit om de routes te clearen van de cache

|  |
| --- |
| php artisan route:clear |

* 1. Ga naar navigation.blade.php en vervang de code vervolgens naar:

|  |
| --- |
| <nav x-data="{ open: false }" class="bg-blue-300 border-b border-gray-100 dark:border-gray-700">  <!-- Primary Navigation Menu -->  <div class="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">  <div class="flex justify-between h-16">  <div class="flex">  <!-- Logo -->  <div class="shrink-0 flex items-center">  <a href="{{ route('dashboard') }}">  <x-application-logo class="block h-9 w-auto fill-current text-gray-800 dark:text-gray-200" />  </a>  </div>  <!-- Navigation Links -->  <div class="hidden space-x-8 sm:-my-px sm:ml-10 sm:flex">  <x-nav-link :href="route('dashboard')" :active="request()->routeIs('dashboard')">  {{ \_\_('books') }}  </x-nav-link>  </div>  <div class="hidden space-x-8 sm:-my-px sm:ml-10 sm:flex">  <x-nav-link :href="route('dashboard')" :active="request()->routeIs('dashboard')">  {{ \_\_('authors') }}  </x-nav-link>  </div>  </div>  <!-- Settings Dropdown -->  @if(Auth::check())  <div class="hidden sm:flex sm:items-center sm:ml-6">  <x-dropdown align="right" width="48">  <x-slot name="trigger">  <button class="inline-flex items-center px-3 py-2 border border-transparent text-sm leading-4 font-medium rounded-md text-gray-500 dark:text-gray-400 bg-white dark:bg-gray-800 hover:text-gray-700 dark:hover:text-gray-300 focus:outline-none transition ease-in-out duration-150">  <div>{{ Auth::user()->name }}</div>  <div class="ml-1">  <svg class="fill-current h-4 w-4" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 20 20">  <path fill-rule="evenodd" d="M5.293 7.293a1 1 0 011.414 0L10 10.586l3.293-3.293a1 1 0 111.414 1.414l-4 4a1 1 0 01-1.414 0l-4-4a1 1 0 010-1.414z" clip-rule="evenodd" />  </svg>  </div>  </button>  </x-slot>  <x-slot name="content">  <x-dropdown-link :href="route('profile.edit')">  {{ \_\_('Profile') }}  </x-dropdown-link>  <!-- Authentication -->  <form method="POST" action="{{ route('logout') }}">  @csrf  <x-dropdown-link :href="route('logout')"  onclick="event.preventDefault();  this.closest('form').submit();">  {{ \_\_('Log Out') }}  </x-dropdown-link>  </form>  </x-slot>  </x-dropdown>  </div>  @else  @if (Route::has('login'))  <div class="hidden sm:flex sm:items-center sm:ml-6">  @auth  <a href="{{ url('/dashboard') }}" class="font-semibold text-white hover:text-gray-900 dark:text-gray-400 dark:hover:text-white focus:outline focus:outline-2 focus:rounded-sm focus:bg-none focus:outline-red-500">Dashboard</a>  @else  <a href="{{ route('login') }}" class="font-semibold text-white hover:text-gray-900 dark:text-gray-400 dark:hover:text-white focus:outline focus:outline-2 focus:rounded-sm focus:outline-red-500">Aanmelden</a>  @if (Route::has('register'))  <a href="{{ route('register') }}" class="ml-4 font-semibold text-white hover:text-gray-900 dark:text-gray-400 dark:hover:text-white focus:outline focus:outline-2 focus:rounded-sm focus:outline-red-500">Registeren</a>  @endif  @endauth  </div>  @endif  @endif  <!-- Hamburger -->  <div class="-mr-2 flex items-center sm:hidden">  <button @click="open = ! open" class="inline-flex items-center justify-center p-2 rounded-md text-gray-400 dark:text-gray-500 hover:text-gray-500 dark:hover:text-gray-400 hover:bg-gray-100 dark:hover:bg-gray-900 focus:outline-none focus:bg-gray-100 dark:focus:bg-gray-900 focus:text-gray-500 dark:focus:text-gray-400 transition duration-150 ease-in-out">  <svg class="h-6 w-6" stroke="currentColor" fill="none" viewBox="0 0 24 24">  <path :class="{'hidden': open, 'inline-flex': ! open }" class="inline-flex" stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M4 6h16M4 12h16M4 18h16" />  <path :class="{'hidden': ! open, 'inline-flex': open }" class="hidden" stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M6 18L18 6M6 6l12 12" />  </svg>  </button>  </div>  </div>  </div>  <!-- Responsive Navigation Menu -->  <div :class="{'block': open, 'hidden': ! open}" class="hidden sm:hidden">  <div class="pt-2 pb-3 space-y-1">  @if(Auth::check())  <x-responsive-nav-link :href="route('dashboard')" :active="request()->routeIs('dashboard')">  {{ \_\_('Dashboard') }}  </x-responsive-nav-link>  @endif  </div>  <div class="pt-2 pb-3 space-y-1">  <x-responsive-nav-link :href="route('dashboard')" :active="request()->routeIs('dashboard')">  {{ \_\_('books') }}  </x-responsive-nav-link>  </div>  <div class="pt-2 pb-3 space-y-1">  <x-responsive-nav-link :href="route('dashboard')" :active="request()->routeIs('dashboard')">  {{ \_\_('authors') }}  </x-responsive-nav-link>  </div>  <!-- Responsive Settings Options -->  <div class="pt-4 pb-1 border-t border-gray-200 dark:border-gray-600">  @if(Auth::check())  <div class="px-4">  <div class="font-medium text-base text-gray-800 dark:text-gray-200">{{ Auth::user()->name }}</div>  <div class="font-medium text-sm text-gray-500">{{ Auth::user()->email }}</div>  </div>  <div class="mt-3 space-y-1">  <x-responsive-nav-link :href="route('profile.edit')">  {{ \_\_('Profile') }}  </x-responsive-nav-link>  <!-- Authentication -->  <form method="POST" action="{{ route('logout') }}">  @csrf  <x-responsive-nav-link :href="route('logout')"  onclick="event.preventDefault();  this.closest('form').submit();">  {{ \_\_('Log Out') }}  </x-responsive-nav-link>  </form>  </div>  </div>  @else  <x-responsive-nav-link align="right" :href="route('login')">  <div class="font-medium text-base text-white" :active="request()->routeIs('Aanmelden')">{{ \_\_('Aanmelden') }}</div>  </x-responsive-nav-link>  @if (Route::has('register'))  <x-responsive-nav-link align="right" :href="route('register')" :active="request()->routeIs('register')">  <div class="font-medium text-base text-white">{{ \_\_('Registeren') }}</div>  </x-responsive-nav-link>  @endif  @endif  </div>  </nav> |

3.5 voer de volgende command uit om de routes te clearen van de cache

|  |
| --- |
| php artisan route:clear |

## Bezoekers kunnen niet de website bezoeken zonder inloggen (4.1.1)

* + 1. Je koos ervoor dat login only is. (Goed voor medewerker only sites)
    2. Ga naar localhost/registeren en maak een account aan
    3. Login in localhost/login. Je kan de registeratie ook wijzigen
    4. Je kan ook registeren onder verwijzing maken dat je ingelogd moet worden en je inlog pagina als voorpagina maken. Hierbij wijzig je de web.php
    5. Eerst zet je redirection naar login pagina dat doe je door:

|  |
| --- |
| Route::get('/', function () {  return redirect('/login');  }); |

* + 1. voer de volgende command uit om de routes te clearen van de cache

|  |
| --- |
| php artisan route:clear |

* + 1. vervolgens zet de registeren in beveiligde weergaven voor login user only. Je gaat naar auth.php en verander het vervolgens naar:

|  |
| --- |
| use App\Http\Controllers\Auth\AuthenticatedSessionController;  use App\Http\Controllers\Auth\ConfirmablePasswordController;  use App\Http\Controllers\Auth\EmailVerificationNotificationController;  use App\Http\Controllers\Auth\EmailVerificationPromptController;  use App\Http\Controllers\Auth\NewPasswordController;  use App\Http\Controllers\Auth\PasswordController;  use App\Http\Controllers\Auth\PasswordResetLinkController;  use App\Http\Controllers\Auth\RegisteredUserController;  use App\Http\Controllers\Auth\VerifyEmailController;  use Illuminate\Support\Facades\Route;  Route::middleware('auth')->group(function () {  // Move the 'register' route here to make it accessible only to authenticated users.  Route::get('register', [RegisteredUserController::class, 'create'])  ->name('register');  Route::post('register', [RegisteredUserController::class, 'store']);  // ... Other routes ...  Route::post('logout', [AuthenticatedSessionController::class, 'destroy'])  ->name('logout');  });  Route::middleware('guest')->group(function () {  Route::get('login', [AuthenticatedSessionController::class, 'create'])  ->name('login');  Route::post('login', [AuthenticatedSessionController::class, 'store']);  Route::get('forgot-password', [PasswordResetLinkController::class, 'create'])  ->name('password.request');  Route::post('forgot-password', [PasswordResetLinkController::class, 'store'])  ->name('password.email');  Route::get('reset-password/{token}', [NewPasswordController::class, 'create'])  ->name('password.reset');  Route::post('reset-password', [NewPasswordController::class, 'store'])  ->name('password.store');  // ... Other routes ...  }); |

* + 1. voer de volgende command uit om de routes te clearen van de cache

|  |
| --- |
| php artisan route:clear |

3.3.9 app.blade.php moet volgende hebben:

|  |
| --- |
| <!DOCTYPE html>  <html lang="{{ str\_replace('\_', '-', app()->getLocale()) }}">      <head>          <meta charset="utf-8">          <meta name="viewport" content="width=device-width, initial-scale=1">          <meta name="csrf-token" content="{{ csrf\_token() }}">          <title>{{ config('app.name', 'Laravel') }}</title>          <!-- Fonts -->          <link rel="preconnect" href="https://fonts.bunny.net">          <link href="https://fonts.bunny.net/css?family=figtree:400,500,600&display=swap" rel="stylesheet" />          <!-- Scripts -->          @vite(['resources/css/app.css', 'resources/js/app.js'])      </head>      <body class="font-sans antialiased">          <div class="min-h-screen bg-gray-100">              @include('layouts.navigation')              <!-- Page Heading -->              @if (isset($header))                  <header class="bg-white shadow">                      <div class="max-w-7xl mx-auto py-6 px-4 sm:px-6 lg:px-8">                          {{ $header }}                      </div>                  </header>              @endif              <!-- Page Content -->              <main>                  {{ $slot }}              </main>          </div>      </body>  </html> |

# **Maken van one-to-many crud (Beste manier) (!!BELANGRIJK!!)**

## Voordat je verder gaat

**Voordat je verder gaat beveel ik aan om het kopje** [**Beginsituatie, hoe begin ik?**](#_Begin_situatie,_hoe) **Voer deze compleet uit en kies vervolgens voor** [**bezoekers kunnen website wel bezoeken zonder inloggen**](#_Bezoekers_kunnen_niet)

## Info wat wij gaan maken

Een boek (one)

Kan meerdere authors hebben (to-many)

### ERD

Afbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijving



### Design

Open design.pdf voor design formaat of open design.fig in figma.



## CRUD Applicatie Structuur Opzetten

Dit commando maakt voor ons naast de Model zelf ook nog een Migration, Seeder, Factory en een resource Controller aan. Voer deze uit in onze commandobar en voer een naam in op “<naam>” enkelvoud.

|  |
| --- |
| php artisan make:model <naam> -msfc --resource |

(naam meerdere tekens is bv KamerInfo inplaats van Kamerinfo)

1. We maken eerst de Authors, dat doen we door volgende commando uit te voeren:

|  |
| --- |
| php artisan make:model Author -msfc --resource |

2. Daarna Book, dat doen we door volgende commando uit te voeren:

|  |
| --- |
| php artisan make:model Book -msfc --resource |

**Deze links zijn HEEL ERG BELANGRIJK:**

<http://127.0.0.1:8000/books>

<http://127.0.0.1:8000/authors>

**DEZE LINKS WERKEN NIET MAAR SLA DEZE OP EEN NOTITIE ZODAT ALS JE ROUTES & BLADES HEBT GEMAAKT HIER MAKKERLIJKER HEEN KAN GAAN**

### Maken van models

<https://laravel.com/docs/10.x/eloquent>

1. Ga naar **One-to-many\Applicatie\one-to-many-crud-app\app\Models\Author.php** en vul de volgende gegevens in:

|  |
| --- |
| <?php  namespace App\Models;  use Illuminate\Database\Eloquent\Factories\HasFactory;  use Illuminate\Database\Eloquent\Model;  use Illuminate\Database\Eloquent\Relations\HasMany;  class Author extends Model  {      use HasFactory;      protected $fillable = [          'name',      ];      public function Books(): HasMany{          return $this->hasMany(Book::class);      }  } |

2. Doe het volgende **One-to-many\Applicatie\one-to-many-crud-app\app\Models\Book.php** ook. Zie table hier beneden:

|  |
| --- |
| <?php  namespace App\Models;  use Illuminate\Database\Eloquent\Factories\HasFactory;  use Illuminate\Database\Eloquent\Model;  use Illuminate\Database\Eloquent\Relations\BelongsTo;  class Book extends Model  {      use HasFactory;      protected $fillable = [          'title',          'author\_id',      ];      public function Author(): BelongsTo{          return $this->belongsTo(Author::class);      }  } |

### Het maken van migrations

<https://laravel.com/docs/10.x/migrations>

1. Ga naar

**one-to-many-crud-app\database\migrations\2023\_10\_16\_113129\_create\_authors\_table.php**

En pas deze aan naar:

|  |
| --- |
| <?php  use Illuminate\Database\Migrations\Migration;  use Illuminate\Database\Schema\Blueprint;  use Illuminate\Support\Facades\Schema;  return new class extends Migration  {      /\*\*       \* Run the migrations.       \*/      public function up(): void      {          Schema::create('authors', function (Blueprint $table) {              $table->id(); // Maakt een auto-increment 'id' kolom              $table->string('name'); // Maakt een kolom voor de naam van de auteur              $table->timestamps(); // Voegt 'created\_at' en 'updated\_at' tijdstempelkolommen toe          });      }      /\*\*       \* Reverse the migrations.       \*/      public function down(): void      {          Schema::dropIfExists('authors'); // Verwijdert de 'authors' tabel      }  }; |

2. Dit gaan wij ook doen bij books migration dit is te vinden in

**One-to-many\Applicatie\one-to-many-crud-app\database\migrations\2023\_10\_16\_114608\_create\_books\_table.php**

En dan verandere we deze migration naar:

|  |
| --- |
| <?php  use Illuminate\Database\Migrations\Migration;  use Illuminate\Database\Schema\Blueprint;  use Illuminate\Support\Facades\Schema;  return new class extends Migration  {      /\*\*       \* Run the migrations.       \*/      public function up(): void      {          Schema::create('books', function (Blueprint $table) {              $table->id(); // Maakt een auto-increment 'id' kolom              $table->string('title'); // Maakt een kolom voor de titel van het boek              $table->foreignId('author\_id')->constrained(); // Maakt een 'author\_id' kolom als een foreign key naar een andere tabel (vermoedelijk de tabel 'authors')              $table->timestamps(); // Voegt 'created\_at' en 'updated\_at' tijdstempelkolommen toe          });      }      /\*\*       \* Reverse the migrations.       \*/      public function down(): void      {          Schema::dropIfExists('books'); // Verwijdert de 'books' tabel      }  }; |

Als je de database wilt inzien doe je met:

|  |
| --- |
| php artisan migrate:fresh --seed |

### Het maken van factories

<https://laravel.com/docs/10.x/eloquent-factories>

1. Ga naar **One-to-many\Applicatie\one-to-many-crud-app\database\factories\AuthorFactory.php** en verander de code naar:

|  |
| --- |
| <?php  namespace Database\Factories;  use Illuminate\Database\Eloquent\Factories\Factory;  /\*\*   \* @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\Author>   \*/  class AuthorFactory extends Factory  {      /\*\*       \* Define the model's default state.       \*       \* @return array<string, mixed>       \*/      public function definition(): array      {          return [              'name' => fake()->name(), // Generates a random name for the author          ];        }  } |

2. Ga naar **One-to-many\Applicatie\one-to-many-crud-app\database\factories\BookFactory.php** en verander de code naar:

|  |
| --- |
| <?php  namespace Database\Factories;  use Illuminate\Database\Eloquent\Factories\Factory;  /\*\*   \* @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\Book>   \*/  class BookFactory extends Factory  {      /\*\*       \* Define the model's default state.       \*       \* @return array<string, mixed>       \*/      public function definition(): array      {          return [              'title' => fake()->sentence(), // Generates a random sentence for the title          ];      }  } |

### Het maken van seeders

<https://laravel.com/docs/10.x/seeding>

1. Ga naar **One-to-many\Applicatie\one-to-many-crud-app\database\seeders\AuthorSeeder.php** en verander de code naar: (kies een method)

|  |
| --- |
| <?php  namespace Database\Seeders;  use Illuminate\Database\Console\Seeds\WithoutModelEvents;  use Illuminate\Database\Seeder;  use App\Models\Author;  use App\Models\Book;  use Illuminate\Database\Eloquent\Factories\Sequence;  class AuthorSeeder extends Seeder  {      /\*\*       \* Run the database seeds.       \*/      public function run(): void      {          // Create 5 author records, each with 3 associated books.          // Method 1          Author::factory(5)              ->has(Book::factory()->count(3), 'books')              ->create();          // Create an author named 'Jan Smith' with 3 specific books.          // Method 2 check bookseeder          Author::factory()              ->has(Book::factory() // relationship with book                      ->count(3)                      ->state(new Sequence(                          ['title' => 'Koken met Jan'],                          ['title' => 'Jan en pizzas'],                          ['title' => 'De beste desserts'],                      ))              )              ->create([                  'name' => 'Jan Smith',              ]);      }  } |

2. Ga naar **One-to-many\Applicatie\one-to-many-crud-app\database\seeders\BookSeeder.php** en verander de code naar:

|  |
| --- |
| <?php  namespace Database\Seeders;  use App\Models\Author;  use App\Models\Book;  use Illuminate\Database\Console\Seeds\WithoutModelEvents;  use Illuminate\Database\Seeder;  class BookSeeder extends Seeder  {      /\*\*       \* Run the database seeds.       \*/      public function run(): void      {          // Creëer 3 boekrecords en associeer elk boek met de auteur 'Jan Smith'.          // method 2 maar onhandig, maakt author aan.          Book::factory(3)              ->for(Author::factory()->state([                  'name' => 'Kees Steen'              ]))              ->create();      }  } |

3. Zet nog wel in dataseeder zodat hij de twee seeders uitvoert. Dus ga naar **One-to-many\Applicatie\one-to-many-crud-app\database\seeders\DatabaseSeeder.php** en over de volgende code uit:

|  |
| --- |
| <?php  namespace Database\Seeders;  // use Illuminate\Database\Console\Seeds\WithoutModelEvents;  use Illuminate\Database\Seeder;  class DatabaseSeeder extends Seeder  {      /\*\*       \* Seed the application's database.       \*/      public function run(): void      {          $this->call([              AuthorSeeder::class,              BookSeeder::class,          ]);      }  } |

4. Voer nu uit

|  |
| --- |
| php artisan migrate:fresh --seed |

Je kan de database vinden op:

Localhost/phpmyadmin

Of

Localhost:8080/phpmyadmin

Of

Localhost:8000/phpmyadmin

### Het maken van crud routes

<https://laravel.com/docs/10.x/controllers#resource-controllers>

Wij willen eerst controllers maken voordat we gaan beginnen met onze blades. Hierbij focusen wij op de routes en controllers.

1. In onze web.php vullen we de volgende code in de file **One-to-many\Applicatie\one-to-many-crud-app\routes\web.php**

|  |
| --- |
| <?php  use App\Http\Controllers\BookController;  use App\Http\Controllers\AuthorController;  use App\Http\Controllers\ProfileController;  use Illuminate\Support\Facades\Route;  /\*  |--------------------------------------------------------------------------  | Web Routes  |--------------------------------------------------------------------------  |  | Here is where you can register web routes for your application. These  | routes are loaded by the RouteServiceProvider and all of them will  | be assigned to the "web" middleware group. Make something great!  |  \*/  Route::get('/', function () {      return view('welcome');  });  Route::get('/dashboard', function () {      return view('dashboard');  })->middleware(['auth', 'verified'])->name('dashboard');  Route::middleware('auth')->group(function () {      Route::get('/profile', [ProfileController::class, 'edit'])->name('profile.edit');      Route::patch('/profile', [ProfileController::class, 'update'])->name('profile.update');      Route::delete('/profile', [ProfileController::class, 'destroy'])->name('profile.destroy');  });  // Method 1:  Route::resources([      'authors' => AuthorController::class,      'books' => BookController::class,  ]);  // Method 2:  // Route::get('/books', [BookController::class, 'index'])->name('books.index');  // Route::get('/books/create', [BookController::class, 'create'])->name('books.create');  // Route::post('/books', [BookController::class, 'store'])->name('books.store');  // Route::get('/books/{book}', [BookController::class, 'show'])->name('books.show');  // Route::get('/books/{book}/edit', [BookController::class, 'edit'])->name('books.edit');  // Route::put('/books/{book}', [BookController::class, 'update'])->name('books.update');  // Route::delete('/books/{book}', [BookController::class, 'destroy'])->name('books.destroy');  // Route::get('/authors', [AuthorController::class, 'index'])->name('authors.index');  // Route::get('/authors/create', [AuthorController::class, 'create'])->name('authors.create');  // Route::post('/authors', [AuthorController::class, 'store'])->name('authors.store');  // Route::get('/authors/{author}', [AuthorController::class, 'show'])->name('authors.show');  // Route::get('/authors/{author}/edit', [AuthorController::class, 'edit'])->name('authors.edit');  // Route::put('/authors/{author}', [AuthorController::class, 'update'])->name('authors.update');  // Route::delete('/authors/{author}', [AuthorController::class, 'destroy'])->name('authors.destroy');  require \_\_DIR\_\_.'/auth.php'; |

Hierin kan je methode 1 en 2 inzien. Je ziet dus dat methode 1 beter is omdat je maar paar regels schrijft in plaats van extreem veel lijnen.

Met route resources kunnen we alle routes maken die we nodig hebben voor books en authors. Deze commandos kan je inzien om vervolgende de volgende commando uit te voeren:

|  |
| --- |
| php artisan route:list |

Je kan ze ook inzien met het schema hieronder als het nog niet duidelijk is:

<https://laravel.com/docs/10.x/controllers#actions-handled-by-resource-controller>

Afbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijving

### Het maken van blades & controllers

### Controller klaargezette voor pagina inzien

1. Ga naar **One-to-many\Applicatie\one-to-many-crud-app\app\Http\Controllers\AuthorController.php** en vul vervolgens in bij index():

|  |
| --- |
| public function index()      {          // Gets all authors          $authors = Author::all();          // sends all authors to authors.index when going to http://127.0.0.1:8000/authors          return view('authors.index' ,['authors' => $authors]);      } |

2. Ga nu naar **One-to-many\Applicatie\one-to-many-crud-app\app\Http\Controllers\BookController.php** en vul vervolgens in bij index():

|  |
| --- |
| public function index()      {          // Gets all books          $books = Book::all();          // sends all books to books.index when going to http://127.0.0.1:8000/books          return view('books.index' ,['books' => $books]);      } |

### Navigatie aanpassen

1. Dit was nodig zodat de navigatie geen errors geeft en de routes kan vinden naar de juiste pagina. We gaan nu onze navigatie aanpassen. Dus ga naar **One-to-many\Applicatie\one-to-many-crud-app\resources\views\layouts\navigation.blade.php** en vul in:

|  |
| --- |
| <nav x-data="{ open: false }" class="bg-blue-300 border-b border-gray-100 dark:border-gray-700">      <!-- Primary Navigation Menu -->      <div class="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">          <div class="flex justify-between h-16">              <div class="flex">                  <!-- Logo -->                  <div class="shrink-0 flex items-center">                      <a href="{{ route('dashboard') }}">                          <x-application-logo class="block h-9 w-auto fill-current text-gray-800 dark:text-gray-200" />                      </a>                  </div>                  <!-- Navigation Links -->                  <div class="hidden space-x-8 sm:-my-px sm:ml-10 sm:flex">                      <x-nav-link :href="route('books.index')" :active="request()->routeIs('books.index')">                          {{ \_\_('books') }}                      </x-nav-link>                  </div>                  <div class="hidden space-x-8 sm:-my-px sm:ml-10 sm:flex">                      <x-nav-link :href="route('authors.index')" :active="request()->routeIs('authors.index')">                          {{ \_\_('authors') }}                      </x-nav-link>                  </div>              </div>              <!-- Settings Dropdown -->              @if(Auth::check())              <div class="hidden sm:flex sm:items-center sm:ml-6">                  <x-dropdown align="right" width="48">                      <x-slot name="trigger">                          <button class="inline-flex items-center px-3 py-2 border border-transparent text-sm leading-4 font-medium rounded-md text-gray-500 dark:text-gray-400 bg-white dark:bg-gray-800 hover:text-gray-700 dark:hover:text-gray-300 focus:outline-none transition ease-in-out duration-150">                              <div>{{ Auth::user()->name }}</div>                              <div class="ml-1">                                  <svg class="fill-current h-4 w-4" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 20 20">                                      <path fill-rule="evenodd" d="M5.293 7.293a1 1 0 011.414 0L10 10.586l3.293-3.293a1 1 0 111.414 1.414l-4 4a1 1 0 01-1.414 0l-4-4a1 1 0 010-1.414z" clip-rule="evenodd" />                                  </svg>                              </div>                          </button>                      </x-slot>                      <x-slot name="content">                          <x-dropdown-link :href="route('profile.edit')">                              {{ \_\_('Profile') }}                          </x-dropdown-link>                          <!-- Authentication -->                          <form method="POST" action="{{ route('logout') }}">                              @csrf                              <x-dropdown-link :href="route('logout')"                                      onclick="event.preventDefault();                                                  this.closest('form').submit();">                                  {{ \_\_('Log Out') }}                              </x-dropdown-link>                          </form>                      </x-slot>                  </x-dropdown>              </div>              @else              @if (Route::has('login'))              <div class="hidden sm:flex sm:items-center sm:ml-6">                  @auth                  <a href="{{ url('/dashboard') }}" class="font-semibold text-white hover:text-gray-900 dark:text-gray-400 dark:hover:text-white focus:outline focus:outline-2 focus:rounded-sm focus:bg-none focus:outline-red-500">Dashboard</a>                  @else                  <a href="{{ route('login') }}" class="font-semibold text-white hover:text-gray-900 dark:text-gray-400 dark:hover:text-white focus:outline focus:outline-2 focus:rounded-sm focus:outline-red-500">Aanmelden</a>                  @if (Route::has('register'))                  <a href="{{ route('register') }}" class="ml-4 font-semibold text-white hover:text-gray-900 dark:text-gray-400 dark:hover:text-white focus:outline focus:outline-2 focus:rounded-sm focus:outline-red-500">Registeren</a>                  @endif                  @endauth              </div>              @endif              @endif              <!-- Hamburger -->              <div class="-mr-2 flex items-center sm:hidden">                  <button @click="open = ! open" class="inline-flex items-center justify-center p-2 rounded-md text-gray-400 dark:text-gray-500 hover:text-gray-500 dark:hover:text-gray-400 hover:bg-gray-100 dark:hover:bg-gray-900 focus:outline-none focus:bg-gray-100 dark:focus:bg-gray-900 focus:text-gray-500 dark:focus:text-gray-400 transition duration-150 ease-in-out">                      <svg class="h-6 w-6" stroke="currentColor" fill="none" viewBox="0 0 24 24">                          <path :class="{'hidden': open, 'inline-flex': ! open }" class="inline-flex" stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M4 6h16M4 12h16M4 18h16" />                          <path :class="{'hidden': ! open, 'inline-flex': open }" class="hidden" stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M6 18L18 6M6 6l12 12" />                      </svg>                  </button>              </div>          </div>      </div>      <!-- Responsive Navigation Menu -->      <div :class="{'block': open, 'hidden': ! open}" class="hidden sm:hidden">          <div class="pt-2 pb-3 space-y-1">          @if(Auth::check())              <x-responsive-nav-link :href="route('dashboard')" :active="request()->routeIs('dashboard')">                  {{ \_\_('Dashboard') }}              </x-responsive-nav-link>              @endif          </div>          <div class="pt-2 pb-3 space-y-1">          <x-responsive-nav-link :href="route('books.index')" :active="request()->routeIs('books.index')">          {{ \_\_('books') }}          </x-responsive-nav-link>          </div>          <div class="pt-2 pb-3 space-y-1">          <x-responsive-nav-link :href="route('authors.index')" :active="request()->routeIs('authors.index')">          {{ \_\_('authors') }}          </x-responsive-nav-link>          </div>          <!-- Responsive Settings Options -->          <div class="pt-4 pb-1 border-t border-gray-200 dark:border-gray-600">             @if(Auth::check())              <div class="px-4">                  <div class="font-medium text-base text-gray-800 dark:text-gray-200">{{ Auth::user()->name }}</div>                  <div class="font-medium text-sm text-gray-500">{{ Auth::user()->email }}</div>              </div>              <div class="mt-3 space-y-1">                  <x-responsive-nav-link :href="route('profile.edit')">                      {{ \_\_('Profile') }}                  </x-responsive-nav-link>                  <!-- Authentication -->                  <form method="POST" action="{{ route('logout') }}">                      @csrf                      <x-responsive-nav-link :href="route('logout')"                              onclick="event.preventDefault();                                          this.closest('form').submit();">                          {{ \_\_('Log Out') }}                      </x-responsive-nav-link>                  </form>              </div>          </div>          @else          <x-responsive-nav-link align="right" :href="route('login')">              <div class="font-medium text-base text-white" :active="request()->routeIs('Aanmelden')">{{ \_\_('Aanmelden') }}</div>          </x-responsive-nav-link>          @if (Route::has('register'))          <x-responsive-nav-link align="right" :href="route('register')" :active="request()->routeIs('register')">              <div class="font-medium text-base text-white">{{ \_\_('Registeren') }}</div>          </x-responsive-nav-link>          @endif          @endif      </div>  </nav> |

### Toestaan van footers in je index

1. Ga naar **One-to-many\Applicatie\one-to-many-crud-app\resources\views\layouts\app.blade.php** en voeg de code toe:

|  |
| --- |
| <!DOCTYPE html>  <html lang="{{ str\_replace('\_', '-', app()->getLocale()) }}">  <head>      <meta charset="utf-8">      <meta name="viewport" content="width=device-width, initial-scale=1">      <meta name="csrf-token" content="{{ csrf\_token() }}">      <title>{{ config('app.name', 'Laravel') }}</title>      <!-- Fonts -->      <link rel="preconnect" href="https://fonts.bunny.net">      <link href="https://fonts.bunny.net/css?family=figtree:400,500,600&display=swap" rel="stylesheet" />      <!-- Scripts -->      @vite(['resources/css/app.css', 'resources/js/app.js'])  </head>  <body class="font-sans antialiased">      <div class="flex flex-col min-h-screen bg-gray-100">          @include('layouts.navigation')          <!-- Page Heading -->          @if (isset($header))          <header class="bg-white shadow">              <div class="max-w-7xl mx-auto py-6 px-4 sm:px-6 lg:px-8">                  {{ $header }}              </div>          </header>          @endif          <!-- Page Content -->          <main class="grow">              {{ $slot }}          </main>          <!-- Page Heading -->          @if (isset($footer))          <footer class="bg-white shadow">              <div class="max-w-7xl mx-auto py-6 px-4 sm:px-6 lg:px-8">                  {{ $footer }}              </div>          </footer>          @endif      </div>  </body>  </html> |

### Alle blades maken

1. Je begint met het maken van de standaard blades. Ga naar:

**One-to-many\Applicatie\one-to-many-crud-app\resources\views**

En vervolgens maak de volgende books/authors folders met de volgende php documenten:

Afbeelding met tekst, schermopname, software, Lettertype

Automatisch gegenereerde beschrijving

**Deze links zijn HEEL ERG BELANGRIJK:**

<http://127.0.0.1:8000/books>

<http://127.0.0.1:8000/authors>

### [AUTHORS] Index maken (blade & controller)

Nadat je dit hebt uitgevoerd wordt dit de begin pagina:

http://127.0.0.1:8000/authors

**Omdat authors kunnen aangemaakt worden zonder Books maar Books moeten perse een author hebben. (one-to-many)**

2. We gaan nu naar onze blade van index deze is te vinden op **One-to-many\Applicatie\one-to-many-crud-app\resources\views\authors\index.blade.php** en daar vullen we het volgende in:

|  |
| --- |
| <x-app-layout>  <div class="my-3">          <a href="{{ route('authors.create') }}" class="border px-5 bg-green-400 text-white border-black font-semibold">Create Artist</a>      </div>      <div class="flex flex-col p-3 gap-3 border border-black">          @foreach ($authors as $author)              <div class="flex flex-row bg-background border border-black bg-[#64CCC5]">                  <a href="{{ route('authors.show', ['author' => $author]) }}" class="grow flex flex-row p-3">                      <span class="font-semibold">{{ $author->name }}</span>                  </a>                  <div class="justify-end flex flex-row p-3 gap-3 ">                      <a href="{{ route('authors.edit', ['author' => $author]) }}" class="border px-5 bg-yellow-400 text-white border-black font-semibold">Edit</a>                      <form action="{{ route('authors.destroy', ['author'=> $author]) }}" method="POST">                          @csrf                          @method('DELETE')                          <input type="submit" value="Delete" class="border px-3 bg-red-400 text-white border-black font-semibold">                      </form>                  </div>              </div>          @endforeach      </div>  <x-slot name="footer">          <h2 class="font-semibold text-xl text-gray-800 leading-tight">              {{ \_\_('footer') }}          </h2>  </x-slot>  </x-app-layout> |

3. Pas de controller aan zodat je op de index terecht kan komen. Deze is te vinden op:

**One-to-many\Applicatie\one-to-many-crud-app\app\Http\Controllers\AuthorController.php**

|  |
| --- |
| /\*\*       \* Display a listing of the resource.       \*/      public function index()      {          // Gets all authors          $authors = Author::all();          // sends all authors to authors.index when going to http://127.0.0.1:8000/authors          return view('authors.index' ,['authors' => $authors]);      } |

**Afbeelding met schermopname, tekst, software, Besturingssysteem

Automatisch gegenereerde beschrijving**

### [AUTHORS] create maken (blade & controller)

1. We gaan nu naar onze blade van create deze is te vinden op **One-to-many\Applicatie\one-to-many-crud-app\resources\views\authors\create.blade.php** en daar vullen we het volgende in:

|  |
| --- |
| <x-app-layout>      <div class="flex flex-row justify-center">          <form action="{{ route('authors.store') }}" method="POST" class="flex flex-col items-center gap-3">              @csrf              <label for="name" class="text-5xl font-semibold">Name</label>              <input type="text" id="name" name="name" placeholder="Insert a name..." required class="text-center">              <input type="submit" value="Submit">          </form>      </div>  </x-app-layout> |

2. Pas de controller aan zodat je op de index terecht kan komen. Deze is te vinden op:

|  |
| --- |
| /\*\*       \* Show the form for creating a new resource.       \*/      public function create()      {          return view('authors.create');      }      /\*\*       \* Store a newly created resource in storage.       \*/      public function store(Request $request)      {          $newAuthor = new Author([              'name' => $request->name,          ]);          $newAuthor->save();          return redirect(route('authors.index'));      } |

**One-to-many\Applicatie\one-to-many-crud-app\app\Http\Controllers\AuthorController.phpAfbeelding met tekst, schermopname, software, nummer

Automatisch gegenereerde beschrijving**

Afbeelding met tekst, schermopname

Automatisch gegenereerde beschrijving

### [AUTHORS] edit maken (blade & controller)

1. We gaan nu naar onze blade van edit deze is te vinden op **One-to-many\Applicatie\one-to-many-crud-app\resources\views\authors\edit.blade.php** en daar vullen we het volgende in:

|  |
| --- |
| <x-app-layout>      <div class="flex flex-row justify-center">          <form action="{{ route('authors.update', ['author' => $author]) }}" method="POST" class="flex flex-col items-center gap-3">              @csrf              @method('PUT')              <label for="name" class="text-5xl font-semibold">Name</label>              <input type="text" id="name" name="name" value="{{ $author->name }}" placeholder="Insert a name..." required class="text-center">              <input type="submit" value="Submit">          </form>      </div>  </x-app-layout> |

2. Pas de controller aan zodat je op de index terecht kan komen nadat je edit hebt uigevoerd. Deze is te vinden op:

|  |
| --- |
| /\*\*       \* Show the form for editing the specified resource.       \*/      public function edit(author $author)      {          return view('authors.edit', ['author' => $author]);      }      /\*\*       \* Update the specified resource in storage.       \*/      public function update(Request $request, author $author)      {          $author->update([              'name' => $request->name,          ]);          return redirect(route('authors.index'));      } |

**One-to-many\Applicatie\one-to-many-crud-app\app\Http\Controllers\AuthorController.php**

Afbeelding met schermopname, tekst, software, Besturingssysteem

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, software

Automatisch gegenereerde beschrijving

### [AUTHORS] destroy maken (blade & controller)

1. We gaan geen destroy pagina maken want deze zit al op de index.blade.php. Koekenbakker, ga terug naar [index maken (blade & controller)](#_Index_maken_(blade) om dit te zien. (http://localhost:8000/authors)

2. Pas de controller aan zodat je op de index terecht kan komen nadat je edit hebt uigevoerd. Deze is te vinden op:

|  |
| --- |
| /\*\*       \* Remove the specified resource from storage.       \*/      public function destroy(author $author)      {          $author->delete();          return redirect(route('authors.index'));      } |

**One-to-many\Applicatie\one-to-many-crud-app\app\Http\Controllers\AuthorController.php**

Afbeelding met tekst, schermopname, software, Besturingssysteem

Automatisch gegenereerde beschrijving

### [AUTHORS] show maken (blade & controller)

1. We gaan nu naar onze blade van show deze is te vinden op **One-to-many\Applicatie\one-to-many-crud-app\resources\views\authors\show.blade.php** en daar vullen we het volgende in:

|  |
| --- |
| <x-app-layout>      <div class="flex flex-col items-center gap-3">          <span class="text-2xl font-semibold">{{ $author->name }}</span>          <div class="flex flex-col gap-2 items-center">              <span class="text-lg font-semibold">Books</span>              <div class="flex flex-col">                  @foreach ($author->books as $book)                      <a href="{{ route('books.show', ['book' => $book]) }}" class="font-semibold">{{ $book->title }}</a>                  @endforeach              </div>          </div>      </div>  </x-app-layout> |

2. Pas de controller aan zodat je op de show terecht kan komen nadat je show hebt uitgevoerd. Deze is te vinden op:

|  |
| --- |
| /\*\*       \* Display the specified resource.       \*/      public function show(author $author)      {          return view('authors.show', ['author' => $author]);      } |

**One-to-many\Applicatie\one-to-many-crud-app\app\Http\Controllers\AuthorController.php**

### [BOOKS] Index maken (blade & controller)

Nadat je dit hebt uitgevoerd wordt dit de begin pagina:

<http://127.0.0.1:8000/books>

2. We gaan nu naar onze blade van index deze is te vinden op **One-to-many\Applicatie\one-to-many-crud-app\resources\views\books\index.blade.php** en daar vullen we het volgende in:

|  |
| --- |
| <x-app-layout>      <div class="flex flex-col gap-2 p-10">          <div class="flex flex-row justify-start">              <a class="bg-background border border-black bg-[#64CCC5] text-[#04364A] font-semibold" href="{{ route('books.create') }}">Book aanmaken</a>          </div>          <div class="flex flex-col p-3 gap-3 border border-black">              @foreach ($books as $book)              <div class="flex flex-row bg-background border border-black bg-[#64CCC5]">                  <a href="{{ route('books.show', ['book' => $book]) }}" class="grow flex flex-row p-3">                      <span class="font-semibold text-[#04364A]">{{ $book->title }}</span>                  </a>                  <div class="justify-end flex flex-row p-3 gap-3 ">                      <a href="{{ route('books.edit', ['book' => $book]) }}" class="border px-5 bg-yellow-400 text-white border-black font-semibold">Edit</a>                      <form action="{{ route('books.destroy', ['book'=> $book]) }}" method="POST">                          @csrf                          @method('DELETE')                          <input type="submit" value="Delete" class="border px-3 bg-red-400 text-white border-black font-semibold hover:cursor-pointer">                      </form>                  </div>              </div>              @endforeach          </div>      </div>      <x-slot name="footer">          <h2 class="font-semibold text-xl text-gray-800 leading-tight">              {{ \_\_('footer') }}          </h2>      </x-slot>  </x-app-layout> |

3. Pas de controller aan zodat je op de index terecht kan komen. Deze is te vinden op:

**One-to-many\Applicatie\one-to-many-crud-app\app\Http\Controllers\BookController.php**

|  |
| --- |
| /\*\*       \* Display a listing of the resource.       \*/      public function index()      {          // Gets all books          $books = Book::all();          // sends all books to books.index when going to http://127.0.0.1:8000/books          return view('books.index' ,['books' => $books]);      } |

Afbeelding met tekst, schermopname, nummer, software

Automatisch gegenereerde beschrijving

### [BOOKS] show maken (blade & controller)

1. We gaan nu naar onze blade van show deze is te vinden op **One-to-many\Applicatie\one-to-many-crud-app\resources\views\Books\show.blade.php** en daar vullen we het volgende in:

|  |
| --- |
| <x-app-layout>      <div class="flex flex-col items-center gap-3">          <span class="text-2xl font-semibold">{{ $book->title }}</span>          <div class="flex flex-col gap-2 items-center">              <span class="text-lg font-semibold">Author</span>              <div class="flex flex-col">                  <!-- Because we only have one author, we don't need to use a foreach loop. -->                  <a href="{{ route('authors.show', ['author' => $book->author]) }}" class="font-semibold">{{ $book->author->name }}</a>              </div>          </div>      </div>  </x-app-layout> |

Omdat wij een one-to-many maken en boeken maar EEN author kunnen hebben hoeven we geen foreach te gebruiken. (zie [ERD](#_ERD))

2. Pas de controller aan zodat je op de show terecht kan komen nadat je show hebt uitgevoerd. Deze is te vinden op:

|  |
| --- |
| /\*\*       \* Display the specified resource.       \*/      public function show(Book $book)      {          return view('books.show', ['book' => $book]);      } |

**One-to-many\Applicatie\one-to-many-crud-app\app\Http\Controllers\ BookController.php**

### [BOOKS] create maken (blade & controller)

1. We gaan nu naar onze blade van create deze is te vinden op **One-to-many\Applicatie\one-to-many-crud-app\resources\views\books\create.blade.php** en daar vullen we het volgende in:

|  |
| --- |
| <x-app-layout>      <div class="flex flex-row justify-center">          <form action="{{ route('books.store') }}" method="POST" class="flex flex-col items-center gap-3">              @csrf              <label for="title" class="text-5xl font-semibold">Title</label>              <input type="text" id="title" name="title" placeholder="Insert a name..." required class="text-center">              <label for="author\_id" class="text-5xl font-semibold">Author</label>              <select id="author\_id" name="author\_id">                  @foreach ($authors as $author)                  <option value="{{ $author->id }}">{{ $author->name }}</option>                  @endforeach              </select>              <input type="submit" value="Submit">          </form>      </div>  </x-app-layout> |

Je kan zien dat we author\_id opvragen. Dit doen we omdat we altijd een author nodig hebben voor onze boeken.

Anders moet je in de controller aangeven dat het nullable moet zijn en haal je author weg. Maar beveel dit NIET aan.

2. Pas de controller aan zodat je op de index terecht kan komen. Deze is te vinden op:

**One-to-many\Applicatie\one-to-many-crud-app\app\Http\Controllers\BookController.php**

|  |
| --- |
| namespace App\Http\Controllers;  use App\Models\Book;  use App\Models\Author;  use Illuminate\Http\Request;  class BookController extends Controller  {      /\*\*       \* Display a listing of the resource.       \*/      public function index()      {          // Gets all books          $books = Book::all();          // sends all books to books.index when going to http://127.0.0.1:8000/books          return view('books.index' ,['books' => $books]);      }      /\*\*       \* Show the form for creating a new resource.       \*/      public function create()      {          // Gets all authors          $authors = Author::all();          return view('books.create',['authors' => $authors]);      }      /\*\*       \* Store a newly created resource in storage.       \*/      public function store(Request $request)      {          $newBook = new Book([              'title' => $request->title,              'author\_id' => $request->author\_id,          ]);          $newBook->save();          return redirect(route('books.index'));      } |

**Afbeelding met tekst, schermopname, software, nummer

Automatisch gegenereerde beschrijving**

Afbeelding met tekst, schermopname, wit, Lettertype

Automatisch gegenereerde beschrijving

### [BOOKS] edit maken (blade & controller)

1. We gaan nu naar onze blade van edit deze is te vinden op **One-to-many\Applicatie\one-to-many-crud-app\resources\views\books\edit.blade.php** en daar vullen we het volgende in:

|  |
| --- |
| <x-app-layout>      <div class="flex flex-row justify-center">          <form action="{{ route('books.update', ['book' => $book]) }}" method="POST" class="flex flex-col items-center gap-3">              @csrf              @method('PUT')              <label for="title" class="text-5xl font-semibold">Title</label>              <input type="text" id="title" name="title" value="{{ $book->title }}" placeholder="Insert a name..." required class="text-center">              <input type="submit" value="Submit">          </form>      </div>  </x-app-layout> |

2. Pas de controller aan zodat je op de index terecht kan komen nadat je edit hebt uigevoerd. Deze is te vinden op:

|  |
| --- |
| /\*\*       \* Show the form for editing the specified resource.       \*/      public function edit(Book $book)      {          return view('books.edit', ['book' => $book]);      }      /\*\*       \* Update the specified resource in storage.       \*/      public function update(Request $request, book $book)      {          $book->update([              'title' => $request->title,          ]);          // dd($book); // uncomment me to see the new data to be stored          return redirect(route('books.index'));      } |

**One-to-many\Applicatie\one-to-many-crud-app\app\Http\Controllers\AuthorController.php**

Afbeelding met tekst, schermopname, software, Besturingssysteem

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, wit, ontwerp

Automatisch gegenereerde beschrijving

### [BOOKS] destroy maken (blade & controller)

1. We gaan geen destroy pagina maken want deze zit al op de index.blade.php. Koekenbakker, ga terug naar [index maken (blade & controller)](#_Index_maken_(blade) om dit te zien. (http://localhost:8000/books)

2. Pas de controller aan zodat je op de index terecht kan komen nadat je edit hebt uigevoerd. Deze is te vinden op:

|  |
| --- |
| /\*\*       \* Remove the specified resource from storage.       \*/      public function destroy(Book $book)      {          $book->delete();          return redirect(route('books.index'));      } |

**One-to-many\Applicatie\one-to-many-crud-app\app\Http\Controllers\BookController.php**

Afbeelding met tekst, schermopname, software, nummer

Automatisch gegenereerde beschrijving

### Frontpage veranderen (extra)

Nu dat we alles klaar hebben kan je het beste de begin pagina op de crud zetten. Dit doe je door om naar **one -to-many\Applicatie\one-to-many-crud-app\routes\web.php** te gaan.

Vervolgens verander je de code van:

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

Naar:

|  |
| --- |
| Route::get('/', function () {      $books = Book::all(); // Fetch books from the database, adjust as needed      return view('artists.index', ['books' => $books]);  }); |

**NU BEN JE KLAAR! :D**

# **Maken van many-to-many crud (Beste manier) (!!BELANGRIJK!!)**

## Voordat je verder gaat

**Voordat je verder gaat beveel ik aan om het kopje** [**Beginsituatie, hoe begin ik?**](#_Begin_situatie,_hoe) **Voer deze compleet uit en kies vervolgens voor** [**bezoekers kunnen website wel bezoeken zonder inloggen**](#_Bezoekers_kunnen_niet)

## Info wat wij gaan maken

Meerdere artworks (many)

Kan meerdere artists hebben (to-many)

### ERD

Afbeelding met tekst, Lettertype, nummer, schermopname

Automatisch gegenereerde beschrijving



### Design

Ongveer zelfde design als one-to-many (zie [Design](#_Design))

## CRUD Applicatie Structuur Opzetten

Dit commando maakt voor ons naast de Model zelf ook nog een Migration, Seeder, Factory en een resource Controller aan. Voer deze uit in onze commandobar en voer een naam in op “<naam>” enkelvoud.

|  |
| --- |
| php artisan make:model <naam> -msfc --resource |

1. We maken eerst de Artworks, dat doen we door volgende commando uit te voeren:

|  |
| --- |
| php artisan make:model Artwork -msfc --resource |

2. Daarna Artists, dat doen we door volgende commando uit te voeren:

|  |
| --- |
| php artisan make:model Artist -msfc --resource |

3. En vergeet niet onze pivot migration table door volgende commando uit te voeren:

|  |
| --- |
| php artisan make:migration create\_artwork\_artist\_table |

**Deze links zijn HEEL ERG BELANGRIJK:**

<http://127.0.0.1:8000/artworks>

<http://127.0.0.1:8000/Artists>

**DEZE LINKS WERKEN NIET MAAR SLA DEZE OP EEN NOTITIE ZODAT ALS JE ROUTES & BLADES HEBT GEMAAKT HIER MAKKERLIJKER HEEN KAN GAAN**

### Maken van models

<https://laravel.com/docs/10.x/eloquent>

1. Ga naar **many-to-many\Applicatie\many-to-many-crud-app\app\Models\Artwork.php** en vul de volgende gegevens in:

|  |
| --- |
| <?php  namespace App\Models;  use Illuminate\Database\Eloquent\Factories\HasFactory;  use Illuminate\Database\Eloquent\Model;  use Illuminate\Database\Eloquent\Relations\BelongsToMany;  class Artwork extends Model  {      use HasFactory;      protected $fillable = [          'name',          'art\_picture'      ];      public function Artists(): BelongsToMany{          return $this->BelongsToMany(Artist::class);      }  } |

2. Doe het volgende **Many-to-many\Applicatie\many-to-many-crud-app\app\Models\Artist.php** ook. Zie table hier beneden:

|  |
| --- |
| <?php  namespace App\Models;  use Illuminate\Database\Eloquent\Factories\HasFactory;  use Illuminate\Database\Eloquent\Model;  use Illuminate\Database\Eloquent\Relations\BelongsToMany;  class Artist extends Model  {      use HasFactory;      protected $fillable = [          'name',          'profile\_picture'      ];      public function Artworks(): BelongsToMany{          return $this->BelongsToMany(Artwork::class);      }    } |

### Het maken van migrations

<https://laravel.com/docs/10.x/migrations>

1. Ga naar

**Many-to-many\Applicatie\many-to-many-crud-app\database\migrations\2023\_11\_03\_121422\_create\_artworks\_table.php**

En pas deze aan naar:

|  |
| --- |
| <?php  use Illuminate\Database\Migrations\Migration;  use Illuminate\Database\Schema\Blueprint;  use Illuminate\Support\Facades\Schema;  return new class extends Migration  {      /\*\*       \* Run the migrations.       \*/      public function up(): void      {          Schema::create('artworks', function (Blueprint $table) {              $table->id();              $table->string('name', 255); // Maakt een kolom voor de naam van de artwork, maximale lengte van 255 tekens              $table->string('art\_picture', 255); // Maakt een kolom voor de foto voor de artwork, maximale lengte van 255 tekens              $table->timestamps();          });      }      /\*\*       \* Reverse the migrations.       \*/      public function down(): void      {          Schema::dropIfExists('artworks');      }  }; |

2. Dit gaan wij ook doen bij artists migration dit is te vinden in

**Many-to-many\Applicatie\many-to-many-crud-app\database\migrations\2023\_11\_03\_121459\_create\_artists\_table.php**

En dan verandere we deze migration naar:

|  |
| --- |
| <?php  use Illuminate\Database\Migrations\Migration;  use Illuminate\Database\Schema\Blueprint;  use Illuminate\Support\Facades\Schema;  return new class extends Migration  {      /\*\*       \* Run the migrations.       \*/      public function up(): void      {          Schema::create('artists', function (Blueprint $table) {              $table->id();              $table->string('name', 255); // Maakt een VARCHAR kolom voor de naam van de artwork met een maximale lengte van 255 tekens              $table->string('profile\_picture', 255); // Maakt een kolom voor de foto voor de artist, maximale lengte van 255 tekens              $table->timestamps();          });      }      /\*\*       \* Reverse the migrations.       \*/      public function down(): void      {          Schema::dropIfExists('artists');      }  }; |

3. Als laatste gaan wij ook onze pivot table maken. Deze regelt de many-to-many connectie en verbind artworks en artists met elkaar komen. **Wegens alfabetisch volgorde doen wij artist\_artwork inplaats van artwork\_artist. Anders krijgt hij een error.**

**Many-to-many\Applicatie\many-to-many-crud-app\database\migrations\2023\_11\_03\_123235\_create\_artwork\_artist\_table.php**

|  |
| --- |
| <?php  use Illuminate\Database\Migrations\Migration;  use Illuminate\Database\Schema\Blueprint;  use Illuminate\Support\Facades\Schema;  return new class extends Migration  {      /\*\*       \* Run the migrations.       \*/      public function up(): void      {          Schema::create('artist\_artwork', function (Blueprint $table) {              // method 1              $table->integer('artwork\_id'); // maak artwork\_id aan als int              $table->integer('artist\_id'); // maak artist\_id aan als int              $table->foreign('artwork\_id')->references('id')->on('artworks'); // maak artwork\_id een connectie aan met de echte tabel artworks en verbind het aan de ID (forein index/relation)              $table->foreign('artist\_id')->references('id')->on('artists'); // maak artist\_id een connectie aan met de echte tabel artist en verbind het aan de ID (forein index/relation)              $table->timestamps();            // // method 2          //     $table->foreignId('artwork\_id')->constrained(); // gebruik foreignId en constrained() voor vreemde sleutel          //     $table->foreignId('artist\_id')->constrained(); // gebruik foreignId en constrained() voor vreemde sleutel          //     $table->timestamps();            //     // Je hebt geen 'id' kolom nodig als je de combinatie van artwork\_id en artist\_id als de primaire sleutel beschouwt.          //     // $table->id(); // Deze regel is verwijderd          });      }      /\*\*       \* Reverse the migrations.       \*/      public function down(): void      {          Schema::dropIfExists('artist\_artwork');      }  }; |

Als je de database wilt inzien doe je met:

|  |
| --- |
| php artisan migrate:fresh --seed |

### Het maken van factories

<https://laravel.com/docs/10.x/eloquent-factories>

1. Ga naar **many-to-many\Applicatie\many-to-many-crud-app\database\factories\ArtistFactory.php** en verander de code naar:

|  |
| --- |
| <?php  namespace Database\Factories;  use Illuminate\Database\Eloquent\Factories\Factory;  /\*\*   \* @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\Artist>   \*/  class ArtistFactory extends Factory  {      /\*\*       \* Define the model's default state.       \*       \* @return array<string, mixed>       \*/      public function definition(): array      {          //          return [              'name' => fake()->name(), // Generates a random name for the author              'profile\_picture' => $this->faker->imageUrl(), // Generates a random image URL              'created\_at' => now(),              'updated\_at' => now(),          ];      }  } |

2. Ga naar **many-to-many\Applicatie\many-to-many-crud-app\database\factories\ArtworkFactory.php** en verander de code naar:

|  |
| --- |
| <?php  namespace Database\Factories;  use Illuminate\Database\Eloquent\Factories\Factory;  /\*\*   \* @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\Artwork>   \*/  class ArtworkFactory extends Factory  {      /\*\*       \* Define the model's default state.       \*       \* @return array<string, mixed>       \*/      public function definition(): array      {          return [              'name' => fake()->name(), // Generates a random name for the author              'art\_picture' => $this->faker->imageUrl(), // Generates a random image URL              'created\_at' => now(),              'updated\_at' => now(),          ];      }  } |

### Het maken van seeders

<https://laravel.com/docs/10.x/seeding>

1. Ga naar **Many-to-many\Applicatie\many-to-many-crud-app\database\seeders\ArtistSeeder.php** en vul de volgende code in:

|  |
| --- |
| <?php  namespace Database\Seeders;  use Illuminate\Database\Console\Seeds\WithoutModelEvents;  use Illuminate\Database\Seeder;  use App\Models\Artist;  use App\Models\Artwork;  class ArtistSeeder extends Seeder  {      /\*\*       \* Run the database seeds.       \*/      public function run(): void      {          Artist::factory(5)->create();      }  } |

2. Ga naar **Many-to-many\Applicatie\many-to-many-crud-app\database\seeders\ArtworkSeeder.php** en verander de code naar:

|  |
| --- |
| <?php  namespace Database\Seeders;  use App\Models\Artwork;  use Illuminate\Database\Console\Seeds\WithoutModelEvents;  use Illuminate\Database\Seeder;  class ArtworkSeeder extends Seeder  {      /\*\*       \* Run the database seeds.       \*/      public function run(): void      {          Artwork::factory(5)->create();      }  } |

3. Ga naar **many-to-many\Applicatie\many-to-many-crud-app\database\seeders** en **MAAK** aan **ArtworkArtistSeeder.php**, vervolgens geef je hem de volgende code:

Afbeelding met tekst, schermopname, Lettertype, lijn

Automatisch gegenereerde beschrijving

In deze code zijn er aantal methods hoe je artist en artwork aan elkaar kan koppelen. Uncomment de lijn die voor jouw het beste is. (Suggesteer method 3)

|  |
| --- |
| <?php  namespace Database\Seeders;  use Illuminate\Database\Console\Seeds\WithoutModelEvents;  use Illuminate\Database\Seeder;  use App\Models\Artist;  use App\Models\Artwork;  class ArtworkArtistSeeder extends Seeder  {      /\*\*       \* Run the database seeds.       \*/      public function run(): void      {          $artists = Artist::all();          $artworks = Artwork::all();          // Each artist get 5 artworks (method 1)          // $artists->each(function($artist) use ($artworks) {$artist->artworks()->attach($artworks);});          // each artist get artwork id 1 (method 2)          // $artists->each(function($artist) use ($artworks) {$artist->artworks()->attach($artworks[0]);});            // each artist get 1 artwork. The loop goes to 5 meaning artist id 1 > artwork id 1. If you have more artists then artwork then it doesn't work. (method 3)          for(              $i = 0;              $i < 5;              $i++          ){$artists[$i]->Artworks()->attach($artworks[$i]);}      }  } |

4. Zet nog wel in dataseeder zodat hij de twee seeders uitvoert. Dus ga naar **many-to-many\Applicatie\many-to-many-crud-app\database\seeders\DatabaseSeeder.php** en over de volgende code uit:

|  |
| --- |
| <?php  namespace Database\Seeders;  // use Illuminate\Database\Console\Seeds\WithoutModelEvents;  use Illuminate\Database\Seeder;  class DatabaseSeeder extends Seeder  {      /\*\*       \* Seed the application's database.       \*/      public function run(): void      {          $this->call([              ArtworkSeeder::class,              ArtistSeeder::class,              ArtworkArtistSeeder::class,          ]);      }  } |

### Het maken van crud routes

<https://laravel.com/docs/10.x/controllers#resource-controllers>

Wij willen eerst controllers maken voordat we gaan beginnen met onze blades. Hierbij focusen wij op de routes en controllers.

1. In onze web.php vullen we de volgende code in de file **Many-to-many\Applicatie\many-to-many-crud-app\routes\web.php**

|  |
| --- |
| <?php  use App\Http\Controllers\ArtistController;  use App\Http\Controllers\ArtworkController;  use App\Http\Controllers\ProfileController;  use App\Models\Artist;  use App\Models\Artwork;  use Illuminate\Support\Facades\Route;  /\*  |--------------------------------------------------------------------------  | Web Routes  |--------------------------------------------------------------------------  |  | Here is where you can register web routes for your application. These  | routes are loaded by the RouteServiceProvider and all of them will  | be assigned to the "web" middleware group. Make something great!  |  \*/  Route::get('/', function () {      return view('welcome');  });  Route::get('/dashboard', function () {      return view('dashboard');  })->middleware(['auth', 'verified'])->name('dashboard');  Route::middleware('auth')->group(function () {      Route::get('/profile', [ProfileController::class, 'edit'])->name('profile.edit');      Route::patch('/profile', [ProfileController::class, 'update'])->name('profile.update');      Route::delete('/profile', [ProfileController::class, 'destroy'])->name('profile.destroy');  });  // Method 1:  Route::resources([      'artists' => ArtistController::class,      'artworks' => ArtworkController::class,  ]);  // Method 2:  // Route::get('/artists', [ArtistController::class, 'index'])->name('artists.index');  // Route::get('/artists/create', [ArtistController::class, 'create'])->name('artists.create');  // Route::post('/artists', [ArtistController::class, 'store'])->name('artists.store');  // Route::get('/artists/{artist}', [ArtistController::class, 'show'])->name('artists.show');  // Route::get('/artists/{artist}/edit', [ArtistController::class, 'edit'])->name('artists.edit');  // Route::put('/artists/{artist}', [ArtistController::class, 'update'])->name('artists.update');  // Route::delete('/artists/{artist}', [ArtistController::class, 'destroy'])->name('artists.destroy');  // Route::get('/artworks', [ArtworkController::class, 'index'])->name('artworks.index');  // Route::get('/artworks/create', [ArtworkController::class, 'create'])->name('artworks.create');  // Route::post('/artworks', [ArtworkController::class, 'store'])->name('artworks.store');  // Route::get('/artworks/{artwork}', [ArtworkController::class, 'show'])->name('artworks.show');  // Route::get('/artworks/{artwork}/edit', [ArtworkController::class, 'edit'])->name('artworks.edit');  // Route::put('/artworks/{artwork}', [ArtworkController::class, 'update'])->name('artworks.update');  // Route::delete('/artworks/{artwork}', [ArtworkController::class, 'destroy'])->name('artworks.destroy');  require \_\_DIR\_\_.'/auth.php'; |

Hierin kan je methode 1 en 2 inzien. Je ziet dus dat methode 1 beter is omdat je maar paar regels schrijft in plaats van extreem veel lijnen.

Met route resources kunnen we alle routes maken die we nodig hebben voor artists en authors. Deze commandos kan je inzien om vervolgende de volgende commando uit te voeren:

|  |
| --- |
| php artisan route:list |

Je kan ze ook inzien met het schema hieronder als het nog niet duidelijk is:

<https://laravel.com/docs/10.x/controllers#actions-handled-by-resource-controller>

Afbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijving

### Controller klaargezette voor pagina inzien

1. Ga naar **many-to-many\Applicatie\many-to-many-crud-app\app\Http\Controllers\ArtistController.php** en vul vervolgens in bij index():

|  |
| --- |
| public function index()      {          // Gets all Artists          $artists = Artist::all();          // sends all Artists to Artists.index when going to http://127.0.0.1:8000/Artists          return view(artists.index' ,['artists' => $artists]);      } |

2. Ga nu naar **many-to-many\Applicatie\many-to-many-crud-app\app\Http\Controllers\ArtworkController.php** en vul vervolgens in bij index():

|  |
| --- |
| /\*\*       \* Display a listing of the resource.       \*/      public function index()      {          // Gets all Artworks          $artworks = Artwork::all();          // sends all Artworks to Artworks.index when going to http://127.0.0.1:8000/Artworks          return view('artworks.index' ,['artworks' => $artworks]);      } |

### Navigatie aanpassen

1. Dit was nodig zodat de navigatie geen errors geeft en de routes kan vinden naar de juiste pagina. We gaan nu onze navigatie aanpassen. Dus ga naar **many-to-many\Applicatie\many-to-many-crud-app\resources\views\layouts\navigation.blade.php** en vul in:

|  |
| --- |
| <nav x-data="{ open: false }" class="bg-blue-300 border-b border-gray-100 dark:border-gray-700">      <!-- Primary Navigation Menu -->      <div class="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">          <div class="flex justify-between h-16">              <div class="flex">                  <!-- Logo -->                  <div class="shrink-0 flex items-center">                      <a href="{{ route('dashboard') }}">                          <x-application-logo class="block h-9 w-auto fill-current text-gray-800 dark:text-gray-200" />                      </a>                  </div>                  <!-- Navigation Links -->                  <div class="hidden space-x-8 sm:-my-px sm:ml-10 sm:flex">                      <x-nav-link :href="route('artists.index')" :active="request()->routeIs('artists.index')">                          {{ \_\_('artists') }}                      </x-nav-link>                  </div>                  <div class="hidden space-x-8 sm:-my-px sm:ml-10 sm:flex">                      <x-nav-link :href="route('artworks.index')" :active="request()->routeIs('artworks.index')">                          {{ \_\_('artworks') }}                      </x-nav-link>                  </div>              </div>              <!-- Settings Dropdown -->              @if(Auth::check())              <div class="hidden sm:flex sm:items-center sm:ml-6">                  <x-dropdown align="right" width="48">                      <x-slot name="trigger">                          <button class="inline-flex items-center px-3 py-2 border border-transparent text-sm leading-4 font-medium rounded-md text-gray-500 dark:text-gray-400 bg-white dark:bg-gray-800 hover:text-gray-700 dark:hover:text-gray-300 focus:outline-none transition ease-in-out duration-150">                              <div>{{ Auth::user()->name }}</div>                              <div class="ml-1">                                  <svg class="fill-current h-4 w-4" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 20 20">                                      <path fill-rule="evenodd" d="M5.293 7.293a1 1 0 011.414 0L10 10.586l3.293-3.293a1 1 0 111.414 1.414l-4 4a1 1 0 01-1.414 0l-4-4a1 1 0 010-1.414z" clip-rule="evenodd" />                                  </svg>                              </div>                          </button>                      </x-slot>                      <x-slot name="content">                          <x-dropdown-link :href="route('profile.edit')">                              {{ \_\_('Profile') }}                          </x-dropdown-link>                          <!-- Authentication -->                          <form method="POST" action="{{ route('logout') }}">                              @csrf                              <x-dropdown-link :href="route('logout')"                                      onclick="event.preventDefault();                                                  this.closest('form').submit();">                                  {{ \_\_('Log Out') }}                              </x-dropdown-link>                          </form>                      </x-slot>                  </x-dropdown>              </div>              @else              @if (Route::has('login'))              <div class="hidden sm:flex sm:items-center sm:ml-6">                  @auth                  <a href="{{ url('/dashboard') }}" class="font-semibold text-white hover:text-gray-900 dark:text-gray-400 dark:hover:text-white focus:outline focus:outline-2 focus:rounded-sm focus:bg-none focus:outline-red-500">Dashboard</a>                  @else                  <a href="{{ route('login') }}" class="font-semibold text-white hover:text-gray-900 dark:text-gray-400 dark:hover:text-white focus:outline focus:outline-2 focus:rounded-sm focus:outline-red-500">Aanmelden</a>                  @if (Route::has('register'))                  <a href="{{ route('register') }}" class="ml-4 font-semibold text-white hover:text-gray-900 dark:text-gray-400 dark:hover:text-white focus:outline focus:outline-2 focus:rounded-sm focus:outline-red-500">Registeren</a>                  @endif                  @endauth              </div>              @endif              @endif              <!-- Hamburger -->              <div class="-mr-2 flex items-center sm:hidden">                  <button @click="open = ! open" class="inline-flex items-center justify-center p-2 rounded-md text-gray-400 dark:text-gray-500 hover:text-gray-500 dark:hover:text-gray-400 hover:bg-gray-100 dark:hover:bg-gray-900 focus:outline-none focus:bg-gray-100 dark:focus:bg-gray-900 focus:text-gray-500 dark:focus:text-gray-400 transition duration-150 ease-in-out">                      <svg class="h-6 w-6" stroke="currentColor" fill="none" viewBox="0 0 24 24">                          <path :class="{'hidden': open, 'inline-flex': ! open }" class="inline-flex" stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M4 6h16M4 12h16M4 18h16" />                          <path :class="{'hidden': ! open, 'inline-flex': open }" class="hidden" stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M6 18L18 6M6 6l12 12" />                      </svg>                  </button>              </div>          </div>      </div>      <!-- Responsive Navigation Menu -->      <div :class="{'block': open, 'hidden': ! open}" class="hidden sm:hidden">          <div class="pt-2 pb-3 space-y-1">          @if(Auth::check())              <x-responsive-nav-link :href="route('dashboard')" :active="request()->routeIs('dashboard')">                  {{ \_\_('Dashboard') }}              </x-responsive-nav-link>              @endif          </div>          <div class="pt-2 pb-3 space-y-1">          <x-responsive-nav-link :href="route('artists.index')" :active="request()->routeIs('artists.index')">          {{ \_\_('artists') }}          </x-responsive-nav-link>          </div>          <div class="pt-2 pb-3 space-y-1">          <x-responsive-nav-link :href="route('artworks.index')" :active="request()->routeIs('artworks.index')">          {{ \_\_('artworks') }}          </x-responsive-nav-link>          </div>          <!-- Responsive Settings Options -->          <div class="pt-4 pb-1 border-t border-gray-200 dark:border-gray-600">             @if(Auth::check())              <div class="px-4">                  <div class="font-medium text-base text-gray-800 dark:text-gray-200">{{ Auth::user()->name }}</div>                  <div class="font-medium text-sm text-gray-500">{{ Auth::user()->email }}</div>              </div>              <div class="mt-3 space-y-1">                  <x-responsive-nav-link :href="route('profile.edit')">                      {{ \_\_('Profile') }}                  </x-responsive-nav-link>                  <!-- Authentication -->                  <form method="POST" action="{{ route('logout') }}">                      @csrf                      <x-responsive-nav-link :href="route('logout')"                              onclick="event.preventDefault();                                          this.closest('form').submit();">                          {{ \_\_('Log Out') }}                      </x-responsive-nav-link>                  </form>              </div>          </div>          @else          <x-responsive-nav-link align="right" :href="route('login')">              <div class="font-medium text-base text-white" :active="request()->routeIs('Aanmelden')">{{ \_\_('Aanmelden') }}</div>          </x-responsive-nav-link>          @if (Route::has('register'))          <x-responsive-nav-link align="right" :href="route('register')" :active="request()->routeIs('register')">              <div class="font-medium text-base text-white">{{ \_\_('Registeren') }}</div>          </x-responsive-nav-link>          @endif          @endif      </div>  </nav> |

### Toestaan van footers in je index

1. Ga naar **many -to-many\Applicatie\many-to-many-crud-app\resources\views\layouts\app.blade.php** en voeg de code toe:

|  |
| --- |
| <!DOCTYPE html>  <html lang="{{ str\_replace('\_', '-', app()->getLocale()) }}">  <head>      <meta charset="utf-8">      <meta name="viewport" content="width=device-width, initial-scale=1">      <meta name="csrf-token" content="{{ csrf\_token() }}">      <title>{{ config('app.name', 'Laravel') }}</title>      <!-- Fonts -->      <link rel="preconnect" href="https://fonts.bunny.net">      <link href="https://fonts.bunny.net/css?family=figtree:400,500,600&display=swap" rel="stylesheet" />      <!-- Scripts -->      @vite(['resources/css/app.css', 'resources/js/app.js'])  </head>  <body class="font-sans antialiased">      <div class="flex flex-col min-h-screen bg-gray-100">          @include('layouts.navigation')          <!-- Page Heading -->          @if (isset($header))          <header class="bg-white shadow">              <div class="max-w-7xl mx-auto py-6 px-4 sm:px-6 lg:px-8">                  {{ $header }}              </div>          </header>          @endif          <!-- Page Content -->          <main class="grow">              {{ $slot }}          </main>          <!-- Page Heading -->          @if (isset($footer))          <footer class="bg-white shadow">              <div class="max-w-7xl mx-auto py-6 px-4 sm:px-6 lg:px-8">                  {{ $footer }}              </div>          </footer>          @endif      </div>  </body>  </html> |

### Alle blades maken

1. Je begint met het maken van de standaard blades. Ga naar:

**many -to-many\Applicatie\many-to-many-crud-app\resources\views**

En vervolgens maak de volgende artists/artworks folders met de volgende php documenten:

Afbeelding met tekst, schermopname, software, Multimediasoftware

Automatisch gegenereerde beschrijving

**Deze links zijn HEEL ERG BELANGRIJK:**

<http://127.0.0.1:8000/artists>

<http://127.0.0.1:8000/artworks>

### [ARTISTS] Index maken (blade & controller)

Nadat je dit hebt uitgevoerd wordt dit de begin pagina:

http://127.0.0.1:8000/artists

2. We gaan nu naar onze blade van index deze is te vinden op **many-to-many\Applicatie\ many -to-many-crud-app\resources\views\artists\index.blade.php** en daar vullen we het volgende in:

|  |
| --- |
| <x-app-layout>      <div class="flex flex-col p-3 gap-3 border border-black">          @foreach ($artists as $artist)              <div class="flex flex-row bg-background border border-black bg-[#64CCC5]">                  <a href="{{ route('artists.show', ['artist' => $artist]) }}" class="grow flex flex-row p-3">                      <span class="font-semibold">{{ $artist->name }}</span>                  </a>                  <div class="justify-end flex flex-row p-3 gap-3 ">                      <a href="{{ route('artists.edit', ['artist' => $artist]) }}" class="border px-5 bg-yellow-400 text-white border-black font-semibold">Edit</a>                      <form action="{{ route('artists.destroy', ['artist'=> $artist]) }}" method="POST">                          @csrf                          @method('DELETE')                          <input type="submit" value="Delete" class="border px-3 bg-red-400 text-white border-black font-semibold">                      </form>                  </div>              </div>          @endforeach      </div>  <x-slot name="footer">          <h2 class="font-semibold text-xl text-gray-800 leading-tight">              {{ \_\_('footer') }}          </h2>  </x-slot>  </x-app-layout> |

3. Pas de controller aan zodat je op de index terecht kan komen. Deze is te vinden op:

**many-to-many\Applicatie\ many -to-many-crud-app\app\Http\Controllers\ArtistController.php**

|  |
| --- |
| /\*\*       \* Display a listing of the resource.       \*/      public function index()      {          // Gets all artists          $artists = Artist::all();          // sends all artists to artists.index when going to http://127.0.0.1:8000/artists          return view('artists.index' ,['artists' => $artists]);      } |

**Afbeelding met tekst, schermopname, software, Besturingssysteem

Automatisch gegenereerde beschrijving**

### [ARTISTS] create maken (blade & controller)

1. We gaan nu naar onze blade van create deze is te vinden op **Many-to-many\Applicatie\many-to-many-crud-app\resources\views\artists\create.blade.php** en daar vullen we het volgende in:

We gaan nog geen many-to-many relation bij toevoegen dit doen we bij artworks. Omdat artworks altijd een artiest moet hebben.

|  |
| --- |
| <x-app-layout>      <div class="flex flex-row justify-center">          <form action="{{ route('artists.store') }}" method="POST" class="flex flex-col items-center gap-3">              @csrf              <label for="name" class="text-5xl font-semibold">Name</label>              <input type="text" id="name" name="name" placeholder="Insert a name..." required class="text-center">              <label for="profile\_picture" class="text-5xl font-semibold">Profile picture</label>              <input type="text" id="profile\_picture" name="profile\_picture" placeholder="Insert an image URL..." required class="text-center">                <!-- Optionally add artwork if the user wants to assign an artwork to an artist. -->              <label for="artwork" class="text-5xl font-semibold">Select Artworks (optional)</label>              <select name="artwork[]" id="artwork" multiple>                  @foreach($artworks as $artwork)                      <option value="{{ $artwork->id }}">{{ $artwork->name }}</option>                  @endforeach              </select>              <small class="text-gray-500">Hold down the Ctrl key (Windows/Linux) or Command key (Mac) to select multiple artworks.</small>              <input type="submit" value="Submit">          </form>      </div>  </x-app-layout> |

2. Pas de controller aan zodat je op de index terecht kan komen. Deze is te vinden op:

|  |
| --- |
| /\*\*       \* Show the form for creating a new resource.       \*/      public function create()      {              // Haal alle kunstwerken op              $artworks = Artwork::all();          return view('artists.create', ['artworks' => $artworks]);      }      public function store(Request $request)      {          $newArtist = new Artist([              'name' => $request->name,              'profile\_picture' => $request->profile\_picture,          ]);            $newArtist->save();            // Attach selected artworks          if ($request->has('artwork')) {              $newArtist->artworks()->attach($request->artwork);          }                return redirect(route('artists.index'));      } |

**many-to-many\Applicatie\many-to-many-crud-app\app\Http\Controllers\ArtistController.php**

Afbeelding met schermopname, tekst, software, Besturingssysteem

Automatisch gegenereerde beschrijvingAfbeelding met tekst, schermopname, software, Lettertype

Automatisch gegenereerde beschrijving

### [ARTISTS] edit maken (blade & controller) )

1. We gaan nu naar onze blade van edit deze is te vinden op **Many-to-many\Applicatie\many-to-many-crud-app\resources\views\artists\edit.blade.php** en daar vullen we het volgende in:

|  |
| --- |
| <x-app-layout>      <div class="flex flex-row justify-center">          <form action="{{ route('artists.update', ['artist' => $artist]) }}" method="POST" class="flex flex-col items-center gap-3">              @csrf              @method('PUT')              <label for="name" class="text-5xl font-semibold">Edit name</label>              <input type="text" id="name" name="name" value="{{ $artist->name }}" placeholder="Insert a name..." required class="text-center">              <label for="profile\_picture" class="text-5xl font-semibold">Edit profile picture</label>              <input type="text" id="profile\_picture" name="profile\_picture" value="{{ $artist->profile\_picture }}" placeholder="Insert an image URL..." required class="text-center">              <!-- Multi-select dropdown for artworks to select different artworks -->              <label for="artworks" class="text-5xl font-semibold">Edit Artworks</label>              <select name="artworks[]" id="artworks" multiple>                  @foreach($artworks as $artwork)                      <option value="{{ $artwork->id }}" {{ in\_array($artwork->id, $artist->artworks->pluck('id')->toArray()) ? 'selected' : '' }}>                          {{ $artwork->name }}                      </option>                  @endforeach              </select>              <small class="text-gray-500">Hold down the Ctrl key (Windows/Linux) or Command key (Mac) to select multiple artworks.</small>              <input type="submit" value="Submit">          </form>      </div>  </x-app-layout> |

2. Pas de controller aan zodat je op de index terecht kan komen nadat je edit hebt uigevoerd. Deze is te vinden op:

**many-to-many\Applicatie\many-to-many-crud-app\app\Http\Controllers\ArtistController.php**

|  |
| --- |
| /\*\*       \* Show the form for editing the specified resource.       \*/      public function edit(Artist $artist)      {                      // Haal alle kunstwerken op                      $artworks = Artwork::all();          return view('artists.edit', ['artist' => $artist, 'artworks' => $artworks]);      }      /\*\*       \* Update the specified resource in storage.       \*/      public function update(Request $request, Artist $artist)      {          $artist->update([              'name' => $request->name,              'profile\_picture' => $request->profile\_picture,          ]);            // Sync selected artworks          if ($request->has('artworks')) {              $artist->artworks()->sync($request->artworks);          } else {              // If no artworks are selected, detach all existing artworks              $artist->artworks()->detach();          }            return redirect(route('artists.index'));      } |

Afbeelding met schermopname, tekst, lijn, nummer

Automatisch gegenereerde beschrijving

### [ARTISTS] destroy maken (blade & controller) )

1. We gaan geen destroy pagina maken want deze zit al op de index.blade.php. Koekenbakker, ga terug naar index maken (artist) om dit te zien. (http://localhost:8000/artists)

2. Pas de controller aan zodat je op de index terecht kan komen nadat je edit hebt uigevoerd. Deze is te vinden op:

Hier gaan we **WEL** een deattach aanmaken zodat als je artist verwijderd ook alle artworks deattach. Das meest logische!

|  |
| --- |
| /\*\*       \* Remove the specified resource from storage.       \*/      public function destroy(Artist $artist)      {      // Detach the artist from all artworks      $artist->artworks()->detach();      // Delete the artist      $artist->delete();      return redirect(route('artists.index'));      } |

**many-to-many\Applicatie\many-to-many-crud-app\app\Http\Controllers\ArtistController.php**

Afbeelding met schermopname, tekst, lijn, software

Automatisch gegenereerde beschrijving

### [ARTISTS] show maken (blade & controller) )

1. We gaan nu naar onze blade van show deze is te vinden op **many-to-many\Applicatie\many-to-many-crud-app\resources\views\artists\show.blade.php** en daar vullen we het volgende in:

|  |
| --- |
| <x-app-layout>      <div class="flex flex-col items-center gap-6 py-8">          <span class="text-2xl font-semibold">{{ $artist->name }}</span>          <img src="{{ $artist->profile\_picture }}" alt="Profile Picture" class="w-32 h-32 rounded-full">          <div class="flex flex-col gap-4 items-center mt-4">              <span class="text-lg font-semibold">Artworks</span>              <div class="flex flex-col">                  @foreach ($artist->artworks as $artwork)                      <div class="flex items-center gap-4">                          <img src="{{ $artwork->art\_picture }}" alt="Art Picture" class="w-16 h-16 rounded">                          <a href="{{ route('artworks.show', ['artwork' => $artwork]) }}" class="font-semibold">{{ $artwork->name }}</a>                      </div>                  @endforeach              </div>          </div>      </div>  </x-app-layout> |

2. Pas de controller aan zodat je op de show terecht kan komen nadat je show hebt uitgevoerd. Deze is te vinden op:

|  |
| --- |
| /\*\*       \* Display the specified resource.       \*/      public function show(artist $artist)      {          return view('artists.show', ['artist' => $artist]);      } |

**many-to-many\Applicatie\many-to-many-crud-app\app\Http\Controllers\ArtistController.php**

### [ARTWORKS] Index maken (blade & controller)

Nadat je dit hebt uitgevoerd wordt dit de begin pagina:

http://127.0.0.1:8000/artworks

2. We gaan nu naar onze blade van index deze is te vinden op **many-to-many\Applicatie\ many -to-many-crud-app\resources\views\artworks\index.blade.php** en daar vullen we het volgende in:

|  |
| --- |
| <x-app-layout>      <div class="my-3 mx-4">          <a href="{{ route('artworks.create') }}" class="border px-5 bg-green-400 text-white border-black font-semibold">Create artwork</a>      </div>      <div class="flex flex-col p-3 gap-3 border border-black mx-4">          @foreach ($artworks as $artwork)              <div class="flex flex-row bg-background border border-black bg-[#64CCC5]">                  <a href="{{ route('artworks.show', ['artwork' => $artwork]) }}" class="grow flex flex-row p-3">                      <span class="font-semibold">{{ $artwork->name }}</span>                  </a>                  <div class="justify-end flex flex-row p-3 gap-3">                      <a href="{{ route('artworks.edit', ['artwork' => $artwork]) }}" class="border px-5 bg-yellow-400 text-white border-black font-semibold">Edit</a>                      <form action="{{ route('artworks.destroy', ['artwork'=> $artwork]) }}" method="POST">                          @csrf                          @method('DELETE')                          <input type="submit" value="Delete" class="border px-3 bg-red-400 text-white border-black font-semibold">                      </form>                  </div>              </div>          @endforeach      </div>      <x-slot name="footer">          <h2 class="font-semibold text-xl text-gray-800 leading-tight">              {{ \_\_('footer') }}          </h2>      </x-slot>  </x-app-layout> |

3. Pas de controller aan zodat je op de index terecht kan komen. Deze is te vinden op:

**many-to-many\Applicatie\ many -to-many-crud-app\app\Http\Controllers\artworkcontroller.php**

|  |
| --- |
| /\*\*       \* Display a listing of the resource.       \*/      public function index()      {          // Gets all Artworks          $artworks = Artwork::all();          // sends all Artworks to Artworks.index when going to http://127.0.0.1:8000/Artworks          return view('artworks.index' ,['artworks' => $artworks]);      } |

Afbeelding met schermopname, tekst, software, Besturingssysteem

Automatisch gegenereerde beschrijving

### [ARTWORKS] create maken (blade & controller)

1. We gaan nu naar onze blade van create deze is te vinden op **Many-to-many\Applicatie\many-to-many-crud-app\resources\views\artworks\create.blade.php** en daar vullen we het volgende in:

We gaan nog geen many-to-many relation bij toevoegen dit doen we bij artworks. Omdat artworks altijd een artiest moet hebben.

|  |
| --- |
| <x-app-layout>      <div class="flex flex-row justify-center">          <form action="{{ route('artworks.store') }}" method="POST" class="flex flex-col items-center gap-3">              @csrf              <label for="name" class="text-5xl font-semibold">Name</label>              <input type="text" id="name" name="name" placeholder="Insert a name..." required class="text-center">              <label for="art\_picture" class="text-5xl font-semibold">art picture</label>              <input type="text" id="art\_picture" name="art\_picture" placeholder="Insert an image URL..." required class="text-center">                <!-- Optionally add artwork if the user wants to assign an artwork to an artwork. -->              <label for="artist" class="text-5xl font-semibold">Select artists (optional)</label>              <select name="artist[]" id="artist" multiple>                  @foreach($artists as $artist)                      <option value="{{ $artist->id }}">{{ $artist->name }}</option>                  @endforeach              </select>              <small class="text-gray-500">Hold down the Ctrl key (Windows/Linux) or Command key (Mac) to select multiple artists.</small>              <input type="submit" value="Submit">          </form>      </div>  </x-app-layout> |

2. Pas de controller aan zodat je op de index terecht kan komen. Deze is te vinden op:

|  |
| --- |
| public function create()      {              // Haal alle kunstwerken op              $artists = artist::all();          return view('artworks.create', ['artists' => $artists]);      }      public function store(Request $request)      {          $newArtwork = new Artwork([              'name' => $request->name,              'art\_picture' => $request->art\_picture,          ]);            $newArtwork->save();            // Attach selected artworks          if ($request->has('artist')) {              $newArtwork->artists()->attach($request->artist);          }                return redirect(route('artworks.index'));      } |

**many-to-many\Applicatie\many-to-many-crud-app\app\Http\Controllers\ArtworkController.php**

**Afbeelding met schermopname, tekst, software, Besturingssysteem

Automatisch gegenereerde beschrijvingAfbeelding met tekst, schermopname, Lettertype, wit

Automatisch gegenereerde beschrijving**

### [ARTWORKS] edit maken (blade & controller)

1. We gaan nu naar onze blade van edit deze is te vinden op **Many-to-many\Applicatie\many-to-many-crud-app\resources\views\artworks\edit.blade.php** en daar vullen we het volgende in:

|  |
| --- |
| <x-app-layout>      <div class="flex flex-row justify-center">          <form action="{{ route('artworks.update', ['artwork' => $artwork]) }}" method="POST" class="flex flex-col items-center gap-3">              @csrf              @method('PUT')              <label for="name" class="text-5xl font-semibold">Edit name</label>              <input type="text" id="name" name="name" value="{{ $artwork->name }}" placeholder="Insert a name..." required class="text-center">              <label for="art\_picture" class="text-5xl font-semibold">Edit art picture</label>              <input type="text" id="art\_picture" name="art\_picture" value="{{ $artwork->art\_picture }}" placeholder="Insert an image URL..." required class="text-center">              <!-- Multi-select dropdown for artists to select different artists -->              <label for="artists" class="text-5xl font-semibold">Edit Artists</label>              <select name="artists[]" id="artists" multiple>                  @foreach($artists as $artist)                      <option value="{{ $artist->id }}" {{ in\_array($artist->id, $artwork->artists->pluck('id')->toArray()) ? 'selected' : '' }}>                          {{ $artist->name }}                      </option>                  @endforeach              </select>              <small class="text-gray-500">Hold down the Ctrl key (Windows/Linux) or Command key (Mac) to select multiple artists or to unselect them.</small>              <input type="submit" value="Submit">          </form>      </div>  </x-app-layout> |

2. Pas de controller aan zodat je op de index terecht kan komen nadat je edit hebt uigevoerd. Deze is te vinden op:

**many-to-many\Applicatie\many-to-many-crud-app\app\Http\Controllers\ArtworkController.php**

|  |
| --- |
| /\*\*       \* Show the form for editing the specified resource.       \*/      public function edit(Artwork $artwork)      {                      // Haal alle artiesten op                      $artists = Artist::all();          return view('artworks.edit', ['artwork' => $artwork, 'artists' => $artists]);      }      /\*\*       \* Update the specified resource in storage.       \*/      public function update(Request $request, Artwork $artwork)      {          $artwork->update([              'name' => $request->name,              'art\_picture' => $request->art\_picture,          ]);            // Sync selected artists          if ($request->has('artists')) {              $artwork->Artists()->sync($request->artists);          } else {              // If no artists are selected, detach all existing artists              $artwork->artists()->detach();          }            return redirect(route('artworks.index'));      } |

Afbeelding met schermopname, tekst, lijn, nummer

Automatisch gegenereerde beschrijving

### [ARTWORKS] destroy maken (blade & controller)

1. We gaan geen destroy pagina maken want deze zit al op de index.blade.php. Koekenbakker, ga terug naar index maken (artwork) om dit te zien. (http://localhost:8000/artworks)

2. Pas de controller aan zodat je op de index terecht kan komen nadat je edit hebt uigevoerd. Deze is te vinden op:

Hier gaan we **WEL** een deattach aanmaken zodat als je artist verwijderd ook alle artists deattach. Das meest logische!

|  |
| --- |
| /\*\*      /\*\*       \* Remove the specified resource from storage.       \*/      public function destroy(artwork $artwork)      {      // Detach the artwork from all artworks      $artwork->artists()->detach();      // Delete the artwork      $artwork->delete();      return redirect(route('artworks.index'));      } |

**many-to-many\Applicatie\many-to-many-crud-app\app\Http\Controllers\ArtworkController.php**

Afbeelding met schermopname, tekst, lijn, nummer

Automatisch gegenereerde beschrijving

### [ARTWORKS] show maken (blade & controller)

1. We gaan nu naar onze blade van show deze is te vinden op **many-to-many\Applicatie\many-to-many-crud-app\resources\views\artworks\show.blade.php** en daar vullen we het volgende in:

|  |
| --- |
| <x-app-layout>      <div class="flex flex-col items-center gap-6 py-8">          <span class="text-2xl font-semibold">{{ $artist->name }}</span>          <img src="{{ $artist->profile\_picture }}" alt="Profile Picture" class="w-32 h-32 rounded-full">          <div class="flex flex-col gap-4 items-center mt-4">              <span class="text-lg font-semibold">Artworks</span>              <div class="flex flex-col">                  @foreach ($artist->artworks as $artwork)                      <div class="flex items-center gap-4">                          <img src="{{ $artwork->art\_picture }}" alt="Art Picture" class="w-16 h-16 rounded">                          <a href="{{ route('artworks.show', ['artwork' => $artwork]) }}" class="font-semibold">{{ $artwork->name }}</a>                      </div>                  @endforeach              </div>          </div>      </div>  </x-app-layout> |

2. Pas de controller aan zodat je op de show terecht kan komen nadat je show hebt uitgevoerd. Deze is te vinden op:

|  |
| --- |
| /\*\*       \* Display the specified resource.       \*/      public function show(artist $artist)      {          return view('artists.show', ['artist' => $artist]);      } |

**many-to-many\Applicatie\many-to-many-crud-app\app\Http\Controllers\ArtworkController.php**

### Frontpage veranderen (extra)

Nu dat we alles klaar hebben kan je het beste de begin pagina op de crud zetten. Dit doe je door om naar **Many-to-many\Applicatie\many-to-many-crud-app\routes\web.php** te gaan.

Vervolgens verander je de code van:

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

Naar:

|  |
| --- |
| Route::get('/', function () {      $artists = Artist::all(); // Fetch artists from the database, adjust as needed      return view('artists.index', ['artists' => $artists]);  }); |

**NU BEN JE KLAAR! :D**

# Alternatieve versie van het maken voor seeders/models/seeders **(Suggesteer dit niet)**

## Migration maken

Migraties worden gebruikt om de databaseschemastructuur te creëren en te wijzigen. Migraties stellen je in staat om de databaseschema te beheren en wijzigingen geleidelijk toe te passen.

1. Begin met het maken van migration (users table als die al bestaat, hoef nie opnieuw te maken)

|  |
| --- |
| php artisan make:migration create\_users\_table |
| php artisan make:migration create\_movies\_table |
| php artisan make:migration create\_review\_movies\_table |

2. Vervang de **migration user\_table** data met het volgende:

(database > migrations > create\_users\_table.php)

|  |
| --- |
| <?php  use Illuminate\Database\Migrations\Migration;  use Illuminate\Database\Schema\Blueprint;  use Illuminate\Support\Facades\Schema;  return new class extends Migration  {      /\*\*       \* Run the migrations.       \*/      public function up(): void      {          Schema::create('users', function (Blueprint $table) {              $table->id();              $table->string('name');              $table->string('email')->unique();              $table->timestamp('email\_verified\_at')->nullable();              $table->string('password');              $table->rememberToken();              $table->timestamps();          });      }      /\*\*       \* Reverse the migrations.       \*/      public function down(): void      {          Schema::dropIfExists('users');      }  }; |

3. Vervang de **migration movie\_table** data met het volgende:

(database > migrations > create\_movies\_table.php)

|  |
| --- |
| <?php  use Illuminate\Database\Migrations\Migration;  use Illuminate\Database\Schema\Blueprint;  use Illuminate\Support\Facades\Schema;  return new class extends Migration  {      /\*\*       \* Run the migrations.       \*/      public function up(): void      {          Schema::create('movies', function (Blueprint $table) {              $table->id();              $table->string('title');              $table->text('description')->nullable();              $table->string('language');              $table->unsignedInteger('runtime')->nullable();              $table->year('year')->nullable();              $table->string('poster\_path')->nullable();              $table->softDeletes();              $table->timestamps();          });      }      /\*\*       \* Reverse the migrations.       \*/      public function down(): void      {          Schema::dropIfExists('movies');      }  }; |

4. In **de migration review\_movies table** vervang je naar het volgende. Hier wordt er een koppeling gemaakt naar user\_id en movie\_id. Deze unsigned je omdat deze foreign keys worden die de movie informatie ontvangt en user.

|  |
| --- |
| <?php  use Illuminate\Database\Migrations\Migration;  use Illuminate\Database\Schema\Blueprint;  use Illuminate\Support\Facades\Schema;  return new class extends Migration  {      public function up()      {          Schema::create('review\_movies', function (Blueprint $table) {              $table->id();              $table->unsignedBigInteger('user\_id'); // Unsigned to match the 'id' column in 'users' table              $table->unsignedBigInteger('movie\_id'); // Unsigned to match the 'id' column in 'movies' table              $table->string('description');              $table->bigInteger('rating');              $table->timestamps();              $table->foreign('movie\_id')->references('id')->on('movies')->onDelete('cascade');              $table->foreign('user\_id')->references('id')->on('users')->onDelete('cascade');          });      }      public function down()      {          Schema::dropIfExists('movies');      }  }; |

5. Voer de commando uit om vervolgens de migration uit te voeren

|  |
| --- |
| php artisan migrate:fresh --seed |

## Het maken van models

Een model vertegenwoordigt een databasetabel en de bijbehorende operaties voor gegevensbeheer en -interactie. Modellen bieden een objectgeoriënteerde interface om met de database te communiceren.

1. Nu dat je migration hebt gemaakt kunnen we beginnen aan models. We maken de movie model om de volgende commando uit te voeren: **(Als model user al bestaat kan je deze overslaan)**

|  |
| --- |
| php artisan make:model User |
| php artisan make:model Movie |
| php artisan make:model ReviewMovie |

2. In de user model zet je vervolgens het volgende in:

(app > Models > User.php)

|  |
| --- |
| <?php  namespace App\Models;  // use Illuminate\Contracts\Auth\MustVerifyEmail;  use Illuminate\Database\Eloquent\Factories\HasFactory;  use Illuminate\Foundation\Auth\User as Authenticatable;  use Illuminate\Notifications\Notifiable;  use Laravel\Sanctum\HasApiTokens;  class User extends Authenticatable  {      use HasApiTokens, HasFactory, Notifiable;      /\*\*       \* The attributes that are mass assignable.       \*       \* @var array<int, string>       \*/      protected $fillable = [          'name',          'email',          'password',          'is\_admin',      ];      /\*\*       \* The attributes that should be hidden for serialization.       \*       \* @var array<int, string>       \*/      protected $hidden = [          'password',          'remember\_token',      ];      /\*\*       \* The attributes that should be cast.       \*       \* @var array<string, string>       \*/      protected $casts = [          'email\_verified\_at' => 'datetime',          'password' => 'hashed',      ];      /\*\*       \* Returns admin state       \*/      public function isAdmin(): bool      {          return $this->is\_admin;      }  } |

3. In de movie model zet je vervolgens het volgende in:

(app > Models > Movie.php)

|  |
| --- |
| <?php  namespace App\Models;  use Illuminate\Database\Eloquent\Factories\HasFactory;  use Illuminate\Database\Eloquent\Model;  use Illuminate\Database\Eloquent\Relations\BelongsToMany;  use Illuminate\Database\Eloquent\SoftDeletes;  class Movie extends Model  {      use HasFactory;      use SoftDeletes;      protected $fillable = [          'title',          'year',          'language',          'runtime',          'description',          'poster\_path',      ];  } |

4. Vervolgens in de model ReviewMovie zet je het volgende erin:  
Dit regelt dat je data kan invullen in description en rating. En geef je door dat movie\_id van movies table en user\_id van user table

Deze PHP Laravel-code definieert twee relaties voor het model `**ReviewMovie**`. De methode `**movies**()` geeft aan dat een `**ReviewMovie**`-model veel-op-veel-relaties heeft met `**Movie**`. De methode `**user**()` specificeert een één-op-één-relatie met `**User**` via het `**user\_id**`-veld. Er is een kleine correctie in de code gemaakt door **'hasOne(User::class)**'. Hierdoor wordt de juiste gebruiker aan de filmrecensie gekoppeld.

**BelongsToMany** is een Laravel-relatie voor veel-op-veel-relaties tussen modellen, bijv. gebruikers en rollen. **HasOne** wordt gebruikt voor één-op-één-relaties, zoals gebruikers en hun profielen. Deze relaties helpen bij het definiëren van de koppeling tussen modellen in Laravel.

(app > Models > ReviewMovie.php)

|  |
| --- |
| <?php  namespace App\Models;  use Illuminate\Database\Eloquent\Factories\HasFactory;  use Illuminate\Database\Eloquent\Model;  use Illuminate\Database\Eloquent\Relations\BelongsToMany;  use Illuminate\Database\Eloquent\Relations\HasOne;  use Illuminate\Database\Eloquent\SoftDeletes;  class ReviewMovie extends Model  {      use HasFactory;      protected $fillable = [          'description',          'rating',          'user\_id',          'movie\_id'      ];      /\*\*       \* The movies that belong to the ReviewMovie.       \*         'movie\_id',       \*/      public function movies(): BelongsToMany      {          return $this->belongsToMany(Movie::class);      }      /\*\*       \* The user\_id that belong to the ReviewMovie.       \*         'user\_id',       \*/      public function user(): HasOne      {          return $this->hasOne(User::class);      }  } |

5. Over opnieuw de migration uit, als dit error geeft dan verwijder alle tables in the table.

Afbeelding met tekst, schermopname, software, Computerpictogram

Automatisch gegenereerde beschrijving

|  |
| --- |
| php artisan migrate:fresh --seed |

## Seeders aanmaken

Nu we onze model en migrations klaar hebben willen we ze wel aanvullen met standaard gegevens. In dit hoofdstuk gaan we dit doen.

1. Laten we beginnen net als daarvoor om de drie seeders te maken. Dit doen we door de volgende commandos uit te voeren: **(je kan UserSeeder skippen met maken als hij er al in staat)**

|  |
| --- |
| php artisan make:seeder UserSeeder |
| php artisan make:seeder MovieSeeder |
| php artisan make:seeder ReviewMovieSeeder |

2. open de User seeder en vul de vervolgende gegevens

(database > seeders > UserSeeder.php)

|  |
| --- |
| <?php  namespace Database\Seeders;  use App\Models\User;  use Illuminate\Database\Seeder;  use Illuminate\Support\Facades\Hash;  class UserSeeder extends Seeder  {      /\*\*       \* Run the database seeds.       \*/      public function run(): void      {          User::factory(10)->create();          User::create([              'name'      => 'admin',              'email'     => 'admin@filmaura.com',              'password'  => Hash::make('password'),              'is\_admin'  => 1,          ]);      }  } |

Dit maakt 10 verschillende users en onze admin user. Zodat als we admin gebruiken en password als wachtwoord kunnen we inloggen.

3. Open de MovieSeeder en vul de volgende gegevens in:

(database > seeders > MovieSeeder.php)

Zelfsprekend regelt dat er dan een film wordt gemaakt in de database van movies

|  |
| --- |
| <?php  namespace Database\Seeders;  use App\Models\Movie;  use Illuminate\Database\Console\Seeds\WithoutModelEvents;  use Illuminate\Database\Seeder;  class MovieSeeder extends Seeder  {      /\*\*       \* Run the database seeds.       \*/      public function run(): void      {          $movie = Movie::create([              'title'       => 'Oppenheimer',              'description' => 'test description',              'language'    => 'English',              'runtime'     => 180,              'year'        => 2023,          ]);      }  } |

4. Verander de seeder van reviewmovieseeder naar het volgende

(database > seeders > reviewMovieSeeder)

|  |
| --- |
| <?php  namespace Database\Seeders;  use Illuminate\Database\Console\Seeds\WithoutModelEvents;  use Illuminate\Database\Seeder;  use App\Models\Movie;  use App\Models\User;  use App\Models\ReviewMovie;  class ReviewMovieSeeder extends Seeder  {      /\*\*       \* Run the database seeds.       \*/      public function run(): void      {            ReviewMovie::create([              'user\_id' => 1,              'movie\_id' => 1,              'description' => 'Was geweldig! 5/5 sterren!',              'rating'  => 5,          ]);          ReviewMovie::create([              'user\_id' => 5,              'movie\_id' => 1,              'description' => 'Fucking troep! 1/5 sterren!',              'rating'  => 1,          ]);      }  } |

5. In de DatabaseSeeder moet je gegevens aanpassen zodat hij de seeder inziet, zie het volgende tabel waarbij reviewmovieseeder erin staat en kopieer deze:

(database > seeders > databaseSeeder.php)

|  |
| --- |
| <?php  namespace Database\Seeders;  // use Illuminate\Database\Console\Seeds\WithoutModelEvents;  use Illuminate\Database\Seeder;  class DatabaseSeeder extends Seeder  {      /\*\*       \* Seed the application's database.       \*/      public function run(): void      {          $this->call([              UserSeeder::class,              MovieSeeder::class,              ReviewMovieSeeder::class,          ]);      }  } |

6. Run de volgende commando om seeder bij te vullen

|  |
| --- |
| php artisan migrate:fresh --seed |