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](#_Toc150523727)

[Git instructies (documentatie) 6](#_Toc150523728)

[Branch maken 6](#_Toc150523729)

[Klonen 6](#_Toc150523730)

[Pushen 6](#_Toc150523731)

[Pullen 7](#_Toc150523732)

[Migraten en seeden 7](#_Toc150523733)

[**Tailwind hulpbron (documentaties)** **(!!BELANGRIJK!!)** 8](#_Toc150523734)

[Breakpoints (als je responsive wil maken) 8](#_Toc150523735)

[Color 8](#_Toc150523736)

[Flex 8](#_Toc150523737)

[Img 8](#_Toc150523738)

[Background img 8](#_Toc150523739)

[Padding 8](#_Toc150523740)

[Scroll padding 8](#_Toc150523741)

[Width & height 8](#_Toc150523742)

[Margin 9](#_Toc150523743)

[Tailwind cheatsheet 9](#_Toc150523744)

[**Laravel hulpbron (one-to-many + many-to-many Documentatie) (!!BELANGRIJK!!)** 10](#_Toc150523745)

[One-to-one 10](#_Toc150523746)

[One-to-many 10](#_Toc150523747)

[Many-to-many 10](#_Toc150523748)

[Datadumps 10](#_Toc150523749)

[Controller 10](#_Toc150523750)

[Blade 10](#_Toc150523751)

[**Model voorbeeld** **(!!BELANGRIJK!!)** 11](#_Toc150523752)

[Uitleg van gebruikte relatie-methoden 11](#_Toc150523753)

[Many-to-Many Model uitleg 12](#_Toc150523754)

[One-to-Many Model uitleg 13](#_Toc150523755)

[Laravel: Wat doet wat? 14](#_Toc150523756)

[Modellen 14](#_Toc150523757)

[Migraties 14](#_Toc150523758)

[Seeders 14](#_Toc150523759)

[Controllers 14](#_Toc150523760)

[web.php (route) 14](#_Toc150523761)

[Resources > css 14](#_Toc150523762)

[Resources > view 15](#_Toc150523763)

[Algemeen, hoe maak je een crud? (Stappenplan) 17](#_Toc150523764)

[Beginsituatie, hoe begin ik? 18](#_Toc150523765)

[Bezoekers kunnen wel website bezoeken zonder inloggen (4.1) 19](#_Toc150523766)

[Bezoekers kunnen niet de website bezoeken zonder inloggen (4.1.1) 24](#_Toc150523767)

[**Maken van one-to-many crud (Beste manier) (!!BELANGRIJK!!)** 27](#_Toc150523768)

[Voordat je verder gaat 27](#_Toc150523769)

[Info wat wij gaan maken 27](#_Toc150523770)

[ERD 27](#_Toc150523771)

[Design 27](#_Toc150523772)

[CRUD Applicatie Structuur Opzetten 28](#_Toc150523773)

[Maken van models 29](#_Toc150523774)

[Het maken van migrations 31](#_Toc150523775)

[Het maken van factories 33](#_Toc150523776)

[Het maken van seeders 35](#_Toc150523777)

[Het maken van crud routes 38](#_Toc150523778)

[Het maken van blades & controllers 41](#_Toc150523779)

[**Maken van many-to-many crud (Beste manier) (!!BELANGRIJK!!)** 67](#_Toc150523780)

[Voordat je verder gaat 67](#_Toc150523781)

[Info wat wij gaan maken 67](#_Toc150523782)

[ERD 67](#_Toc150523783)

[Design 67](#_Toc150523784)

[CRUD Applicatie Structuur Opzetten 68](#_Toc150523785)

[Maken van models 69](#_Toc150523786)

[Het maken van migrations 71](#_Toc150523787)

[Het maken van factories 75](#_Toc150523788)

[Het maken van seeders 77](#_Toc150523789)

[Het maken van crud routes 81](#_Toc150523790)

[**[NIET GEBRUIKEN, OUD WERK]** 84](#_Toc150523791)

[Info wat voor one-to-many crud we gaan maken 84](#_Toc150523792)

[Maken van de begin blade om de crud te benaderen 85](#_Toc150523793)

[Migration maken 88](#_Toc150523794)

[Het maken van models 91](#_Toc150523795)

[Seeders aanmaken 97](#_Toc150523796)

[Ons begin blade verder uitbreiden 101](#_Toc150523797)

[CRUD-maken (Create) 102](#_Toc150523798)

[CRUD-maken (Store) 109](#_Toc150523799)

# **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.

## 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) |

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 |

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="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.

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.

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

**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'); // Maakt een kolom voor de naam van de artwork              $table->string('art\_picture'); // Maakt een kolom voor de foto voor de artwork              $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'); // Maakt een kolom voor de naam van de artist              $table->string('profile\_picture'); // Maakt een kolom voor de foto voor de artist              $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) {              $table->id();              $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();          });      }      /\*\*       \* 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('authors.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():

|  |
| --- |
| 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('authors.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:

|  |
| --- |
|  |

# **[NIET GEBRUIKEN, OUD WERK]**

## Info wat voor one-to-many crud we gaan maken

Wij gaan een crud maken waarbij een user van een filmwebsite Meerdere reviews achter kan laten onder een film.

**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)

(bv <https://www.rottentomatoes.com/>)

**Dit betekent dus:**

User (persoon die review maakt) (one)

Kan meerdere

Film Reviews (many)

Maken

**One-to-many relationship dus.**



ERD:

Afbeelding met tekst, schermopname, diagram

Automatisch gegenereerde beschrijving

## Maken van de begin blade om de crud te benaderen

**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)

1. Ga naar resources > views

2. Maak een php file aan en noem deze filmpagina.blade.php

3. In deze php file zet de volgende code in:

|  |
| --- |
| <x-app-layout>      <x-slot name="header">      </x-slot>      <div class="flex container">          <div class="flex-none w-16 ">          </div>          <div class="flex-1 w-64 ">              <!-- Column 02 with an image -->              <img src="https://cdn.discordapp.com/attachments/1148233632096464936/1156596080167890996/image.png?ex=65158b8a&is=65143a0a&hm=012970fdffd077929e0b119b530289ed79f5d15fd940fd4c799a6464a65d4b0a&" alt="Your Image" class="max-w-full h-auto">          </div>          <div class="flex-none w-32 ">          </div>          <div class="flex-none w-32 ">          </div>          <div class="flex-1 w-64 ">              <div class="text-center flex-col">                  <h1 class="mt-10 text-black font-bold lg:text-4xl stroke-black text-xl">Film name</h1>                  <h1 class="mt-10 text-black lg:text-2xl stroke-black text-base">Film description</h1>                  <div>                      <button class="mt-10 text-fuchsia-700 font-bold lg:text-4xl stroke-black text-base">                          < terug</button>                  </div>                  <div>                      <button class="bg-green-400 hover:bg-green-700 text-white font-bold py-2 px-4 rounded-full my-8 text-xs lg:text-xl">Create review</button>                  </div>              </div>          </div>          <div class="flex-none w-16 ">          </div>      </div>      <div class="container">          <div class=" h-24 ">              <table class="text-left w-full">                  <thead class="bg-black flex text-white w-full">                      <tr class="flex w-full mb-4">                          <th class="p-4 w-1/3">Rating</th>                          <th class="p-4 w-1/3">Reviewer</th>                          <th class="p-4 w-1/3">Review</th>                      </tr>                  </thead>                  <!-- Remove the nasty inline CSS fixed height on production and replace it with a CSS class — this is just for demonstration purposes! -->                  <tbody class="bg-grey-light flex flex-col items-center justify-between overflow-y-scroll w-full" style="height: 50vh;">                      <tr class="flex w-full mb-4">                          <td class="p-4 w-1/3">Dogs</td>                          <td class="p-4 w-1/3">Cats</td>                          <td class="p-4 w-1/3">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque convallis finibus ante. Nulla facilisi. Praesent sem nibh, mattis sit amet ultricies ac, mattis ac ante. In neque orci, consectetur non varius nec, sollicitudin sed velit. Aenean nec enim porttitor, efficitur nisi in, vulputate sem. Nullam iaculis enim a dolor posuere, eu hendrerit massa cursus. Praesent vehicula bibendum eros id posuere. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae;                              Nunc lobortis mattis tincidunt. Vivamus massa quam, bibendum viverra pharetra sed, vestibulum id eros. Suspendisse porta faucibus turpis in ullamcorper. Sed elementum, elit eu bibendum tincidunt, arcu velit fermentum est, quis auctor nunc dolor sit amet felis. Fusce urna justo, posuere ac ex in, malesuada luctus lacus. Sed vitae tortor et ligula egestas gravida. Sed tincidunt imperdiet purus, vitae ultricies mauris rhoncus dignissim. Aliquam ut risus lorem.                              Curabitur sit amet sagittis mi. Morbi facilisis dui at urna vulputate varius. Aenean mollis, tellus in lobortis lacinia, arcu risus ultrices metus, in condimentum libero sem non turpis. Aliquam pellentesque, odio sit amet rutrum rhoncus, odio nisl ornare orci, sed ultricies sem odio at urna. Donec aliquam consequat urna vel tempor. Integer tristique feugiat dignissim. Nullam tellus massa, gravida nec sapien nec, placerat lobortis diam. Aenean non nunc eget leo tincidunt vehicula a non massa. Aliquam sit amet est risus. Vestibulum ullamcorper sed elit ac ullamcorper. In non rhoncus urna, eget fermentum ante. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Sed ante magna, suscipit ut blandit ut, tincidunt vel sapien. Sed ex nibh, mollis sed sagittis dapibus, vehicula quis tellus. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae;</td>                      </tr>                      <tr class="flex w-full mb-4">                          <td class="p-4 w-1/3">Dogs</td>                          <td class="p-4 w-1/3">Cats</td>                          <td class="p-4 w-1/3">Birds</td>                      </tr>                  </tbody>              </table>          </div>          <div class="flex-none w-16 ...">          </div>      </div>  </x-app-layout> |

4. Ga naar je web.php (Deze locatie is op routes > web.php)

5. toevoeg de volgende route: (localhost:8000/filmpagina kan je nu gebruiken om de pagina te benaderen)

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

## 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 |

## Ons begin blade verder uitbreiden

Nu dat we onze migration & seeder en model af hebben gaan we onze begin blade verder uitbreiden. We willen namelijk dat we nog een blade krijgen waarbij we reviews van ons kunnen aanpassen/verwijderen of maken.

1. Ga naar resources > view

2. Maak hier een folder genaamd “review-movies”

3. Maak hier vervolgens een edit.blade.php, index.blade.php en een show.blade.php

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

4.

## CRUD-maken (Create)

1. Begin met de volgende commando uit te voeren om een controller te maken. Deze regelt standaard layout voor crud

|  |
| --- |
| php artisan make:controller ReviewMovieController --resource |

2. Zoals je kan lezen hebben we in onze create hier beneden een return view gedaan zodat hij naar die pagina verwezen wordt maar we willen ook direct onze films doorsturen naar de create.blade. Dit doen de door **movie::all()** erin te stoppen. De “**’movies’ => movie::all()**” betekent alleen dat we hem de naam movies geven. Kopieer de volgende code in je controller om dit door te voeren.

(app > http > controllers > ReviewMovieController.php)

|  |
| --- |
| <?php  namespace App\Http\Controllers;  use Illuminate\Http\Request;  use App\Models\User;  use App\Models\ReviewMovie;  use App\Models\Movie;  use Carbon\Carbon;  class ReviewMovieController extends Controller  {      /\*\*       \* Display a listing of the resource.       \*/      public function index()      {          // return view('review-movies.index', ['reviewMovies' => ReviewMovie::all()]);      }      /\*\*       \* Show the form for creating a new resource.       \*/      public function create()      {          return view('review-movies.create', ['movies' => Movie::all()]);      }      /\*\*       \* Store a newly created resource in storage.       \*/      public function store(Request $request)      {          //      }      /\*\*       \* Display the specified resource.       \*/      public function show(string $id)      {          //      }      /\*\*       \* Show the form for editing the specified resource.       \*/      public function edit(string $id)      {          //      }      /\*\*       \* Update the specified resource in storage.       \*/      public function update(Request $request, string $id)      {          //      }      /\*\*       \* Remove the specified resource from storage.       \*/      public function destroy(string $id)      {          //      }  } |

3. De route moet nu schreven worden voor de controller. Deze doen we in de web.php zodat hij benaderd kan worden. Dit doe je door de volgende code toe te passen in onze web.php. Dus verander de volgende code van web.php naar de volgende:

(routes > web.php)

|  |
| --- |
| Route::get('/reviews-movies/create', [ReviewMovieController::class, 'create'])->name('reviews-movies.create'); |

Volledige web.php code (is niet persee nodig maar als er een error is kan je hem overschrijven):

|  |
| --- |
| <?php  use App\Http\Controllers\ProfileController;  use Illuminate\Support\Facades\Route;  use App\Http\Controllers\GenreController;  use App\Http\Controllers\ReviewMovieController;  /\*  |--------------------------------------------------------------------------  | 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('frontpage');  });  Route::get('/filmpagina', function () {      return view('filmpagina');  });  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');  });  Route::get('/reviews-movies/create', [ReviewMovieController::class, 'create'])->name('reviews-movies.create');  // Route::get('/reviews-movies/index', [ReviewMovieController::class, 'index'])->name('reviews-movies.index');  require \_\_DIR\_\_.'/auth.php';  require \_\_DIR\_\_.'/admin.php'; |

4. We hebben onze controller gemaakt en route gemaakt via web.php. Nu gaan we naar onze view filmpagina en passen de volgende code toe hier beneden.  
(resources > views > filmpagina.blade.php)

Deze regelt dat wij via de naam die gegeven is in **web.php** (reviews-movies.create) deze kunnen benaderen. In onze **ReviewMovieController.php** kan je ook nalezen welke pagina hij naar gestuurd wordt.

|  |
| --- |
| <x-app-layout>      <x-slot name="header">      </x-slot>      <div class="flex container">          <div class="flex-none w-16 ">          </div>          <div class="flex-1 w-64 ">              <!-- Column 02 with an image -->              <img src="https://cdn.discordapp.com/attachments/1148233632096464936/1156596080167890996/image.png?ex=65158b8a&is=65143a0a&hm=012970fdffd077929e0b119b530289ed79f5d15fd940fd4c799a6464a65d4b0a&" alt="Your Image" class="max-w-full h-auto">          </div>          <div class="flex-none w-32 ">          </div>          <div class="flex-none w-32 ">          </div>          <div class="flex-1 w-64 ">              <div class="text-center flex-col">                  <h1 class="mt-10 text-black font-bold lg:text-4xl stroke-black text-xl">Film name</h1>                  <h1 class="mt-10 text-black lg:text-2xl stroke-black text-base">Film description</h1>                  <div>                      <button class="mt-10 text-fuchsia-700 font-bold lg:text-4xl stroke-black text-base">                          < terug</button>                  </div>                  @if(Auth::check())                  <div>                      <a href="{{ route('reviews-movies.create') }}" class="bg-green-400 hover:bg-green-700 text-white font-bold py-2 px-4 rounded-full my-8 text-xs lg:text-xl">Create review</a>                  </div>                  @endif              </div>          </div>          <div class="flex-none w-16 ">          </div>      </div>      <div class="container">          <div class=" h-24 ">              <table class="text-left w-full">                  <thead class="bg-black flex text-white w-full">                      <tr class="flex w-full mb-4">                          <th class="p-4 w-1/3">Rating</th>                          <th class="p-4 w-1/3">Reviewer</th>                          <th class="p-4 w-1/3">Review</th>                      </tr>                  </thead>                  <!-- Remove the nasty inline CSS fixed height on production and replace it with a CSS class — this is just for demonstration purposes! -->                  <tbody class="bg-grey-light flex flex-col items-center justify-between overflow-y-scroll w-full" style="height: 50vh;">                      <tr class="flex w-full mb-4">                          <td class="p-4 w-1/3">Dogs</td>                          <td class="p-4 w-1/3">Cats</td>                          <td class="p-4 w-1/3">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque convallis finibus ante. Nulla facilisi. Praesent sem nibh, mattis sit amet ultricies ac, mattis ac ante. In neque orci, consectetur non varius nec, sollicitudin sed velit. Aenean nec enim porttitor, efficitur nisi in, vulputate sem. Nullam iaculis enim a dolor posuere, eu hendrerit massa cursus. Praesent vehicula bibendum eros id posuere. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae;                              Nunc lobortis mattis tincidunt. Vivamus massa quam, bibendum viverra pharetra sed, vestibulum id eros. Suspendisse porta faucibus turpis in ullamcorper. Sed elementum, elit eu bibendum tincidunt, arcu velit fermentum est, quis auctor nunc dolor sit amet felis. Fusce urna justo, posuere ac ex in, malesuada luctus lacus. Sed vitae tortor et ligula egestas gravida. Sed tincidunt imperdiet purus, vitae ultricies mauris rhoncus dignissim. Aliquam ut risus lorem.                              Curabitur sit amet sagittis mi. Morbi facilisis dui at urna vulputate varius. Aenean mollis, tellus in lobortis lacinia, arcu risus ultrices metus, in condimentum libero sem non turpis. Aliquam pellentesque, odio sit amet rutrum rhoncus, odio nisl ornare orci, sed ultricies sem odio at urna. Donec aliquam consequat urna vel tempor. Integer tristique feugiat dignissim. Nullam tellus massa, gravida nec sapien nec, placerat lobortis diam. Aenean non nunc eget leo tincidunt vehicula a non massa. Aliquam sit amet est risus. Vestibulum ullamcorper sed elit ac ullamcorper. In non rhoncus urna, eget fermentum ante. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Sed ante magna, suscipit ut blandit ut, tincidunt vel sapien. Sed ex nibh, mollis sed sagittis dapibus, vehicula quis tellus. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae;</td>                      </tr>                      <tr class="flex w-full mb-4">                          <td class="p-4 w-1/3">Dogs</td>                          <td class="p-4 w-1/3">Cats</td>                          <td class="p-4 w-1/3">Birds</td>                      </tr>                  </tbody>              </table>          </div>          <div class="flex-none w-16 ...">          </div>      </div>  </x-app-layout> |

5. app.cs moet ook aangepast worden omdat we een apparte container gebruiken

(resources > css > app.css)

|  |
| --- |
| @tailwind base;  @tailwind components;  @tailwind utilities;  .container {      margin-left: auto;      margin-right: auto;    } |

6. Nu gaan we naar resources > views > review-movies > create.blade.php. En hierin zetten we de volgende code in om een lijst te krijgen. Deze lijst heeft hij doorgestuurd gekregen van de **ReviewMovieController.php.** want in deze controller hebben wij alle films opgenoemd via **['movies' => Movie::all()** en de naam **movies** gegeven. Dit is na te lezen in de **@foreach ($movies as $movie)**loop die doorgaat totdat hij alle films heeft in een lijst.

Dus voer de volgende code in create.blade.php: (je kan de **dd($movies);** uitcommenten om het te zien in actie. Hij krijgt hem wel binnen via een array dus moet in een list. )

(resources > views > review-movies > create.blade.php)

|  |
| --- |
| <?php  // dd($movies);  // foreach($movies as $movie) {  //     echo($movie->title);  // }  ?>  <!-- <form method="POST" action="{{ route('members.store') }}"> -->  <select name="movie" id="movie">    @foreach ($movies as $movie)      <option value="{{$movie->id}}">{{$movie->title}}</option>    @endforeach  </select>  <!-- </from> --> |

## CRUD-maken (Store)

Nu gaan wij de store maken die regelt dat de gegevens opgeslagen worden. Omdat onze create regelt dat hij naar juiste pagina gaat willen wij wel namelijk dat de gegevens die wij opgeven op deze pagina, opslaan.

1. Laten we beginnen om onze **reviewmoviecontroller** aan te passen. Voer de volgende code door voor de store om gegevens op te slaan en te valideren:

(app > http > controllers > ReviewMovieController.php)

|  |
| --- |
| <?php  namespace App\Http\Controllers;  use Illuminate\Http\Request;  use App\Models\User;  use App\Models\ReviewMovie;  use App\Models\Movie;  use Carbon\Carbon;  use Illuminate\Support\Facades\Auth;  use Illuminate\Support\Facades\Validator;  class ReviewMovieController extends Controller  {      /\*\*       \* Display a listing of the resource.       \*/      public function index()      {          // return view('review-movies.index', ['reviewMovies' => ReviewMovie::all()]);      }      /\*\*       \* Show the form for creating a new resource.       \*/      public function create()      {          return view('review-movies.create', ['movies' => Movie::all()]);      }      /\*\*       \* Store a newly created resource in storage.       \*/      public function store(Request $request)      {          // Define validation rules          $rules = [              'movie\_id' => 'required|exists:movies,id', // Check if the movie exists              'rating' => 'required|integer|min:1|max:5', // Example: Rating should be an integer between 1 and 5              'description' => 'required|string|max:255', // Example: Description should be a string with a maximum length of 255          ];            // Validate the request data          $validator = Validator::make($request->all(), $rules);            // Check if the validation fails          if ($validator->fails()) {              return redirect()->route('reviews-movies.create')                  ->withErrors($validator) // Pass the validation errors back to the form                  ->withInput(); // Retain the user's input data          }            // If validation passes, create the ReviewMovie instance          $review = new ReviewMovie([              'user\_id' => Auth::id(),              'movie\_id' => $request->movie\_id,              'rating' => $request->rating,              'description' => $request->description,          ]);            $review->save();            return redirect(route('filmpagina'));      }        /\*\*       \* Display the specified resource.       \*/      public function show(string $id)      {          //      }      /\*\*       \* Show the form for editing the specified resource.       \*/      public function edit(string $id)      {          //      }      /\*\*       \* Update the specified resource in storage.       \*/      public function update(Request $request, string $id)      {          //      }      /\*\*       \* Remove the specified resource from storage.       \*/      public function destroy(string $id)      {          //      }  } |

2. Vervolgens gaan wij onze blade aan te passen zodat we de gegevens kunnen opslaan en de errors gezien kunnen worden op de create.blade

(resources > views > review-movies > create.blade.php)

|  |
| --- |
| <x-app-layout>      <div class="px-2 xl:px-52">      <div class="flex flex-col">      @if ($errors->any())  <div class="alert alert-danger">      <ul>          @foreach ($errors->all() as $error)              <li>{{ $error }}</li>          @endforeach      </ul>  </div>  @endif  <div>      <a href="{{ route('filmpagina') }}" class="mt-10 text-fuchsia-700 font-bold text-base md:text-4xl stroke-black">          &lt; Terug      </a>  </div>        <form method="post" action="{{route('review-movies.store')}}" class="flex flex-col self-center w-full xl:w-1/2">          @csrf          <select name="movie\_id" id="movie\_id">            @foreach ($movies as $movie)            <option value="{{$movie->id}}">{{$movie->title}}</option>            @endforeach          </select>          <!-- justify center kan ook inplaats van items-center -->          <div class="flex items-center flex-col">          <p> Geef hier uw beoordeling in sterren! </p>            <div class="flex items-center">              <input type="radio" id="rating1" name="rating" value="0" class="text-yellow-500">              <label for="rating1" class="text-lg">0 ★</label>              <input type="radio" id="rating1" name="rating" value="1" class="text-yellow-500">              <label for="rating1" class="text-lg">1 ★</label>              <input type="radio" id="rating2" name="rating" value="2" class="text-yellow-500">              <label for="rating2" class="text-lg">2 ★</label>              <input type="radio" id="rating3" name="rating" value="3" class="text-yellow-500">              <label for="rating3" class="text-lg">3 ★</label>              <input type="radio" id="rating4" name="rating" value="4" class="text-yellow-500">              <label for="rating4" class="text-lg">4 ★</label>              <input type="radio" id="rating5" name="rating" value="5" class="text-yellow-500">              <label for="rating5" class="text-lg">5 ★</label>            </div>          </div>          <textarea id="description" name="description" rows="4"></textarea>          <input type="submit">        </form>      </div>      </div>      </div>  </x-app-layout> |

3. Passen onze filmpagina nog even aan

(resources > views > filmpagina.blade.php)

|  |
| --- |
| <x-app-layout>      <x-slot name="header">      </x-slot>      <div class="flex container">          <div class="flex-none w-16 ">          </div>          <div class="flex-1 w-64 ">              <!-- Column 02 with an image -->              <img src="https://cdn.discordapp.com/attachments/1148233632096464936/1156596080167890996/image.png?ex=65158b8a&is=65143a0a&hm=012970fdffd077929e0b119b530289ed79f5d15fd940fd4c799a6464a65d4b0a&" alt="Your Image" class="max-w-full h-auto">          </div>          <div class="flex-none w-32 ">          </div>          <div class="flex-none w-32 ">          </div>          <div class="flex-1 w-64 ">              <div class="text-center flex-col">                  <h1 class="mt-10 text-black font-bold lg:text-4xl stroke-black text-xl">Film name</h1>                  <h1 class="mt-10 text-black lg:text-2xl stroke-black text-base">Film description</h1>                  <div >                      <button class="mt-10 text-fuchsia-700 font-bold lg:text-4xl stroke-black text-base">                          < terug</button>                  </div>                  @if(Auth::check())                  <div class="mt-10">                      <a href="{{ route('reviews-movies.create') }}" class="bg-green-400 hover:bg-green-700 text-white font-bold py-2 px-4 rounded-full my-8 text-xs lg:text-xl">Create review</a>                  </div>                  @endif              </div>          </div>          <div class="flex-none w-16 ">          </div>      </div>      <div class="container mt-10">          <div class=" h-24 ">              <table class="text-left w-full">                  <thead class="bg-black flex text-white w-full">                      <tr class="flex w-full mb-4">                          <th class="p-4 w-1/3">Rating</th>                          <th class="p-4 w-1/3">Reviewer</th>                          <th class="p-4 w-1/3">Review</th>                      </tr>                  </thead>                  <!-- Remove the nasty inline CSS fixed height on production and replace it with a CSS class — this is just for demonstration purposes! -->                  <tbody class="bg-grey-light flex flex-col items-center justify-between overflow-y-scroll w-full" style="height: 50vh;">                      <tr class="flex w-full mb-4">                          <td class="p-4 w-1/3">Dogs</td>                          <td class="p-4 w-1/3">Cats</td>                          <td class="p-4 w-1/3">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque convallis finibus ante. Nulla facilisi. Praesent sem nibh, mattis sit amet ultricies ac, mattis ac ante. In neque orci, consectetur non varius nec, sollicitudin sed velit. Aenean nec enim porttitor, efficitur nisi in, vulputate sem. Nullam iaculis enim a dolor posuere, eu hendrerit massa cursus. Praesent vehicula bibendum eros id posuere. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae;                              Nunc lobortis mattis tincidunt. Vivamus massa quam, bibendum viverra pharetra sed, vestibulum id eros. Suspendisse porta faucibus turpis in ullamcorper. Sed elementum, elit eu bibendum tincidunt, arcu velit fermentum est, quis auctor nunc dolor sit amet felis. Fusce urna justo, posuere ac ex in, malesuada luctus lacus. Sed vitae tortor et ligula egestas gravida. Sed tincidunt imperdiet purus, vitae ultricies mauris rhoncus dignissim. Aliquam ut risus lorem.                              Curabitur sit amet sagittis mi. Morbi facilisis dui at urna vulputate varius. Aenean mollis, tellus in lobortis lacinia, arcu risus ultrices metus, in condimentum libero sem non turpis. Aliquam pellentesque, odio sit amet rutrum rhoncus, odio nisl ornare orci, sed ultricies sem odio at urna. Donec aliquam consequat urna vel tempor. Integer tristique feugiat dignissim. Nullam tellus massa, gravida nec sapien nec, placerat lobortis diam. Aenean non nunc eget leo tincidunt vehicula a non massa. Aliquam sit amet est risus. Vestibulum ullamcorper sed elit ac ullamcorper. In non rhoncus urna, eget fermentum ante. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Sed ante magna, suscipit ut blandit ut, tincidunt vel sapien. Sed ex nibh, mollis sed sagittis dapibus, vehicula quis tellus. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae;</td>                      </tr>                      <tr class="flex w-full mb-4">                          <td class="p-4 w-1/3">Dogs</td>                          <td class="p-4 w-1/3">Cats</td>                          <td class="p-4 w-1/3">Birds</td>                      </tr>                  </tbody>              </table>          </div>          <div class="flex-none w-16 ...">          </div>      </div>  </x-app-layout> |

4. Als laatste passen we onze web.php aan om store route toe te passen:

(Routes > web.php)

|  |
| --- |
| <?php  use App\Http\Controllers\ProfileController;  use Illuminate\Support\Facades\Route;  use App\Http\Controllers\GenreController;  use App\Http\Controllers\ReviewMovieController;  /\*  |--------------------------------------------------------------------------  | 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('frontpage');  });  Route::get('/filmpagina', function () {      return view('filmpagina');  })->name('filmpagina');  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');      Route::get('/reviews-movies/create', [ReviewMovieController::class, 'create'])->name('reviews-movies.create');  // Route::get('/reviews-movies/index', [ReviewMovieController::class, 'index'])->name('reviews-movies.index');  Route::post('/reviews-movies', [ReviewMovieController::class, 'store'])->name('review-movies.store');  });    require \_\_DIR\_\_.'/auth.php';  require \_\_DIR\_\_.'/admin.php'; |

TO DO

Vertel over de controller en web.php locatie naar juiste create pagina

Op create pagina kan je vervolgens rating geven en commentaar

Gebruik store op vervolgens op te slaan