Afbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijving

Instructie crud

Gemaakt door Ryan H.

Datum 06/09/2023

Inhoud

[Git instructies 4](#_Toc147584895)

[Branch maken 4](#_Toc147584896)

[Klonen 4](#_Toc147584897)

[Pushen 4](#_Toc147584898)

[Pullen 5](#_Toc147584899)

[Migraten en seeden 5](#_Toc147584900)

[Laravel: Wat doet wat? 6](#_Toc147584901)

[Modellen 6](#_Toc147584902)

[Migraties 6](#_Toc147584903)

[Seeders 6](#_Toc147584904)

[Controllers 6](#_Toc147584905)

[web.php (route) 6](#_Toc147584906)

[Resources > css 6](#_Toc147584907)

[Resources > view 7](#_Toc147584908)

[Tailwind hulpbron 8](#_Toc147584909)

[Color 8](#_Toc147584910)

[Flex 8](#_Toc147584911)

[Img 8](#_Toc147584912)

[Background img 8](#_Toc147584913)

[Padding 8](#_Toc147584914)

[Scroll padding 8](#_Toc147584915)

[Width & height 8](#_Toc147584916)

[Margin 8](#_Toc147584917)

[Tailwind cheatsheet 9](#_Toc147584918)

[Breakpoints (als je responsive wil maken) 9](#_Toc147584919)

[Algemeen, hoe maak je een crud? (not recommended) 10](#_Toc147584920)

[Beginsituatie, hoe begin ik? 11](#_Toc147584921)

[Bezoekers kunnen wel website bezoeken zonder inloggen (4.1) 12](#_Toc147584922)

[Bezoekers kunnen niet de website bezoeken zonder inloggen (4.1.1) 16](#_Toc147584923)

[Maken van one-to-many crud 18](#_Toc147584924)

[Voordat je verder gaat 18](#_Toc147584925)

[Info wat wij gaan maken 18](#_Toc147584926)

[ERD 18](#_Toc147584927)

[Design 18](#_Toc147584928)

[Maken van many-to-many crud 19](#_Toc147584929)

[NIET GEBRUIKEN, OPNIEUW MAKEN 20](#_Toc147584930)

[Info wat voor one-to-many crud we gaan maken 20](#_Toc147584931)

[Maken van de begin blade om de crud te benaderen 21](#_Toc147584932)

[Migration maken 24](#_Toc147584933)

[Het maken van models 27](#_Toc147584934)

[Seeders aanmaken 33](#_Toc147584935)

[Ons begin blade verder uitbreiden 37](#_Toc147584936)

[CRUD-maken 38](#_Toc147584937)

# Git instructies

## 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

# 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.

# Tailwind hulpbron

## 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/>

## Breakpoints (als je responsive wil maken)

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

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

# Algemeen, hoe maak je een crud? (not recommended)

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:

|  |
| --- |
| * 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) |

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('frontpage');  }); |

4.3 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-white dark:bg-gray-800 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')">  {{ \_\_('Dashboard') }}  </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 md: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>  <!-- 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> |

4.4 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 |

# Maken van one-to-many crud

## 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

Doe dit met ralf samen voor examen niet project groep

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 |

# Maken van many-to-many crud

Doe dit met ralf samen maar niet met project groep voor examen

# [NIET GEBRUIKEN, OPNIEUW MAKEN]

## 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',      ];      /\*\*       \* 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

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> --> |

7.

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