2020

Laravel PHP framework

WEBDEVELOPMENT

P.C. DREIJER

Inleiding

Laravel is een PHP framework wat bestaat uit een verschillende tools en andere modules om een moderne PHP applicaties te bouwen. Laravel is populaire bij verschillende stagebedrijven. Als beginnend developer kan het dus een goede keuze zijn om dit frameworke te leren.

leerdoelen

Met deze tutorial leer je een eenvoudige Laravel applicatie maken en leer je hoe je deze installeert en configureert op de Raspberry PI en Ubuntu server.

Voorkennis

- Installatie Ubuntu server op de Raspberry PI
- SSH toegang Ubuntu server
- Basis commando: Linux
- LEMP (Linux Nginx, MySQL, PHP)

Tutorials youtube

De handelingen en stappen in deze tutorial zijn opgenomen en gepubliceerd op het youtube Awesome kanaal. Klik op de link naar de verzameling van filmpjes

Laravel
https://www.youtube.com/playlist?list=PLtD2txtVprMaUIqAcrGZZi0KaWoB5EOB2

Tutorials Raspberry PI
https://www.youtube.com/playlist?list=PLtD2txtVprMbCuLLDZCBLNGnVfYqeqw8r

Graag een duimpje omhoog als deze hebt bekeken.

Opmerking:

Tijdens het maken van deze tutorial heb ik ontdekt dat er soms een fout op treedt bij het installeren van composer en later de laravel omgeving. Naar veel testen en oplossing zoeken op het internet ben ik tot conclussie gekomen.

Composer en daarmee ook Laravel werkt niet op de 64 bits versies van Ubuntu. Dit levert problemen op de verwerking en het geheugen.

```
https://ubuntu.com/download/raspberry-pi
```

Nginx webserver met Laravel

De Nginx gebruiken we om het http-verkeer te regelen op de Raspberry PI of server. Al het verkeer dat binnenkomt op port 80 moet door Nginx geleid naar de PHP server en naar de Laravel omgeving. De Nginx omgeving moet geconfigureerd worden voor een extern IP adres.

Installeren Nginx webserver

```
$ sudo apt update

$ sudo apt upgrade

$ sudo apt install nginx
```

Controleren op de beschikbare firewall pakketten:

```
$ sudo ufw app list
```

Om te zorgen dat de webserver bereikbaar is voor de browser, moeten we deze toegang geven tot port 80 en 443 in de firewall. Dit doe je met het volgende commando:

```
$ sudo ufw allow 'Nginx http'
$ sudo ufw allow 'Nginx https'
```

```
$ sudo ufw allow 'OpenSSH'
```

Starten van de firewall:

```
$ sudo ufw enable
```

Status van de firewall:

```
$ sudo ufw status
```

Je moet weten welke IP-adressen je gebruikt in je netwerk. Het gaat om het interne adres en het externe adres. Met de volgende commando's doe je dit.

```
$ ip addr show eth0 | grep inet | awk '{ print $2; }' | sed 's/\/.*$//'

Output:

 XXX.XXX.XXX.XXX               {Intern adres

 fe80::ba27:ebff:fe5f:750      {hardware adres


$ curl -4 icanhazip.com

 Output:
 XXX.XXX.XXX.XXX
```

Configureren Nginx webserver

Na de installatie van de Nginx server, moeten deze gaan configureren. Dit doen we in verschillende bestanden en in verschillende directory's. Belangrijk is dat je regelmatig controleert of in de juist direcory en/of bestand werkt.

Voorbereiden webmap

Om alles goed te kunnen voorbereiden, maken we voor deze configuratie eerst een map voor de webserver aan. Dit doe je met het volgende commando en path op de server:

```
$ sudo mkdir /var/www/portfolio
```

In de aangemaakte map plaats je later de bestanden zoals PHP, HTML enz. De Nginx webserver kijkt straks ook naar deze map.

Check/verplaatsen naar de directory:

```
$ cd /var/www/portfolio
```

Configureren Nginx webserver profiel

Het externe ip adres van de Raspberry PI of server moet worden verwerkt in het profiel van Nginx. Dit doe dit als volgt:

```
$ sudo nano /etc/nginx/sites-available/laravel
```

In het profiel Nginx bestand verwerk je de volgende regels met code. Let op vervang **`IP adress`** voor het interne adres van de Raspberry PI of gebruikte server en het externe adress van de router en/of domein naam.

```
server {
        listen 80;
        root /var/www/portfolio;
        server_name IP-adres IP-extern domainname;

        add_header X-Frame-Options "SAMEORIGIN";
        add_header X-XSS-Protection "1; mode=block";
        add_header X-Content-Type-Options "nosniff";

        index index.html index.htm index.php;

        charset utf-8;

        location / {
                try_files $uri $uri/ /index.php?$query_string;
        }

        location = /favicon.ico { access_log off; log_not_found off; }
        location = /robots.txt  { access_log off; log_not_found off; }

        error_page 404 /index.php;

        location ~ \.php$ {
                fastcgi_pass unix:/var/run/php/php7.4-fpm.sock;
                fastcgi_index index.php;
                fastcgi_param SCRIPT_FILENAME
$realpath_root$fastcgi_script_name;
                include fastcgi_params;
        }

        location ~ /\.(?!well-known).* {
        deny all;

        }

}
```

Maak een link naar de map sites-enabled om het profiel laravel beschikbaar te maken voor de Nginx omgeving. Dit doe je met het volgende commando:

```
$ sudo ln -s /etc/nginx/sites-available/laravel /etc/nginx/sites-enabled/
```

Unlink de link naar de map sites-enabled om het profiel default niet meer beschikbaar te maken voor de Nginx omgeving. Dit doe je met het volgende commando:

```
unlink:
$ sudo unlink /etc/nginx/sites-enabled/default
```

Test of de Nginx omgeving goed is geconfigureerd met het volgende commando. Je krijgt hiermee de output: syntax ok en test is succesvol.

```
$ sudo nginx -t

OUTPUT:

nginx: the configuration file /etc/nginx/nginx.conf syntax is ok

nginx: configuration file /etc/nginx/nginx.conf test is successful
```

```
$ sudo systemctl enable nginx

$ sudo systemctl start nginx
```

HTML test file

Om te kijken of de configuratie werkt, gaan we het volgende test bestand aan maken.

```
$ sudo nano /var/www/portfolio/index.html
```

Voer het volgende php commando in het bestand en bewaar dit bestand.

```
<h1>Hello World!!</h1>
```

Bekijk de volgende url in de browser en test de HTML functionaliteit:

```
http://ip-adress/      : HTML configuratie
```

Installeren PHP

Toevoegen repositories

Een repositorie of wel een repo is een online opslag plekvoor software en de daarmee gerealateerde afhankelijkheden. Deze afhankelijkheden zijn noodzakelijk om een softwarpakket te kunnen installeren. Via het pakketbeheer commando apt installeer je de benodigde software voor de server.

Standaard zijn er een aantal links naar repositories geconfigureerd bij het configuren en installeren van de Ubuntu Linux server. Voordat we de laatste versie of een gewenste versie kunnen installeren van PHP en Nginx, moeten we een nieuwe link naar een repository toevoegen aan de server omgeving. Dit doe je met het volgende commando:

```
$  sudo apt install software-properties-common

$  sudo add-apt-repository ppa:ondrej/php
```

PHP versie 7.4

Nu kunnen we de gewenste 7.4 versie van PHP toevoegen aan de server met het volgende commando:

```
$  sudo apt install php7.4
```

Verwijderen Apache2 webserver

Met het vorige commando word er ook automatisch Apache2 webserver geïnstalleerd, maar bij deze tutorials maak je gebruik van Nginx webserver. Om problemen te voorkomen kun je deze verwijderen met het volgende commando:

```
$ sudo apt autoremove apache2

$ sudo systemctl disable --now apache2
```

Installatie van additionale onderdelen (additional packages) PHP

De Apache2 webserver is standaard geïnstalleerd, maar deze gebruiken we niet meer. Om problemen te voorkomen kun je deze verwijderen met het volgen de commando:

```
$  sudo apt install php-curl php-gd php-mbstring php-xml php-xmlrpc

$  sudo apt install php-soap php-intl php-zip php7.4-bcmath php7.4-bz2

$  sudo apt install php7.4-mysql

$  sudo apt install unzip
```

FastCGI Process Manager (FPM)

De FPM is een het alternatief voor PHP FastCGI software. Het biedt meer capaciteit. Het is noodzakenlijk voor het gebruik van PHP in de Nginx webserver.

```
$ sudo apt install php7.4-fpm

$ systemctl status php7.4-fpm Nginx

Klik op: controle c om het scherm af te sluiten.
```

PHP test file

Om te kijken of de configuratie werkt, gaan we het volgende test bestand aan maken.

```
$ sudo nano /var/www/portfolio/info.php
```

Voer het volgende php commando in het bestand en bewaar dit bestand.

```
<?php phpinfo() ?>
```

Reboot het de Raspberry PI met het volgende commando:

```
$ sudo reboot

Of

$ sudo systemctl restart nginx
```

Bekijk na een paar minute de volgende url in de browser en test de functionaliteit:

```
http://ip-adress/info.php      : PHP configuratie
```

Database: MySQL

Installeren MySQL server

```
$ sudo apt install mysql-server
$ sudo mysql_secure_installation
```

mysql_secure_installation

```
$ sudo mysql

mysql> SELECT user,authentication_string,plugin,host FROM mysql.user;
```

```
mysql> SELECT user,authentication_string,plugin,host FROM mysql.user;
+------------------+-------------------------------------------+-----------------------+-----------+
| user             | authentication_string                     | plugin                | host      |
+------------------+-------------------------------------------+-----------------------+-----------+
| root             |                                           | auth_socket           | localhost |
| mysql.session    | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE | mysql_native_password | localhost |
| mysql.sys        | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE | mysql_native_password | localhost |
| debian-sys-maint | *6F3D4A3F24D5766AB9C5D1CFF7DA46AEB3CB86D6 | mysql_native_password | localhost |
+------------------+-------------------------------------------+-----------------------+-----------+
4 rows in set (0.00 sec)
```

Uitloggen

```
$ exit
```

Database creatie

Met de volgende commando's in maak een database aan met met een gebruiker en een wachtwoord. Noteer de gebruikte naam en wachwoord. Deze heb je later nodig.
Database: portfolio_list aanmaken

```
$ sudo mysql

mysql> CREATE DATABASE `portfolio_list`;

mysql> CREATE USER `portfolio_user`@`localhost` IDENTIFIED BY 'admin123';

mysql> GRANT ALL PRIVILEGES ON portfolio_list.* TO
'portfolio_user'@'localhost';

mysql> exit
```

```
` = is voor database namen
' = is voor wachtwoorden/namen
```

Je hebt nu een nieuwe database aangemaakt met een eigen user. Deze gebruiker en database verwerken we later in de backend omgeving.

Database check

Check of opnieuw kunt inloggen met de volgende commando's.

```
$ mysql -u portfolio_user -p
```

```
mysql> SHOW DATABASES;

Output:
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| portfolio_list     |
+--------------------+
2 rows in set (0.01 sec)
```

Laravel composer

Voorbereiding: installeren composer

De basis PHP voor Nginx is geinstalleerd, we gaan nu de server update en benodigde componten voor composer installeren. Ga nu terug of check of in de home map werkt van de ingelogde gebruiker (ubuntu). Daarna kun je composer installeren voor Laravel.

```
$ cd ~          } terug naar de gebruikers home map.

$ pwd           } Print Working Directory
```

~ = terug naar de gebruikers home map.

```
$ curl -sS https://getcomposer.org/installer -o composer-setup.php

$ php composer-setup.php

$ php composer.phar

$ sudo mv composer.phar /usr/local/bin/composer

$ composer --version

Output:
Composer version 1.9.3 2020-02-04 12:58:49
```

```
$ composer

Output:
Overzicht commando's
```

Node.JS en NPM

Installeren node.js

https://nodejs.org/en/

Node Package Manager installeren

Installeren van de front-end dependencies (afhankelijkheden):

```
$ sudo apt -y install curl dirmngr apt-transport-https lsb-release ca-
certificates

$ curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash -

$ sudo apt -y install nodejs

$ sudo apt -y  install gcc g++ make

$ node --version

Output:
V12.16.2

$ npm --version

Output:
6.14.4

$ npm fund
```

LARAVEL

Installeren: laravel

We gaan met Laravel een complete Create Read Update en Delete gegevens of wel CRUD applicatie bouwen. In het vorige stappen hebben we de Mysql database omgeving geconfigureerd voor de applicatie. De gegevens van de database we hebben we nodig voor dit project.

In eerste instatie maken we met de composer software een en nieuwe Laravel project omgeving aan. Let op: we gebruiken hierbij de laatste versie van Laravel voor. Dit is versie 7.x.

```
$ composer global require laravel/installer

$ export PATH=$HOME/.config/composer/vendor/bin:$PATH

$ source ~/.bashrc
```

OSX 10.15 of macos Big Sur gebruikers Z-shell

```
$ composer global require laravel/installer

$ export PATH=$HOME/.config/composer/vendor/bin:$PATH

$ source ~/.zshrc
```

Wil je het PATH in de home-directory permanent toevoegen aan de Bash-shell. Voer dan het volgende commando uit.

```
$ echo 'export PATH=$HOME/.config/composer/vendor/bin:$PATH' >> ~/.bashrc

$ source ~/.zshrc
```

OSX 10.15 of macos Big Sur gebruikers Z-shell

```
$ echo 'export PATH=$HOME/.config/composer/vendor/bin:$PATH' >> ~/.bashrc

$ source ~/.zshrc
```

Ontstaat er het probleem dat de commando's niet werken. Voer dan he§t volgende herstel commando uit. De Bash terminaal is namenlijk dan het PATH kwijt naar de verschillende commando's.

```
$ export PATH=/bin:/usr/bin:/usr/local/bin:/sbin:/usr/sbin
```

Laravel: CRUD werken applicatie

We kunnen met de composer en het geinstalleerde Laravel commando een Laravel applicatie

```
$ laravel new werken_app
$ cd werken_app
```

Front-end afhankelijkheden

In de aangemaakte projectmap staat ook een package.json bestand. Dit bestand bevat de omschrijfing en afhankelijkheden voor deze Laravel installatie. Installeren front-end afhankelijkheden:

```
$  npm install
```

Mysql database koppelingen

Open het .env bestand en pas de gegevens van de mysql database. Dit bestand vind je in de root van de directorie.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=portfolio_list
DB_USERNAME=portfolio_user
DB_PASSWORD=password
```

Het PHP Artisan console commando

Artisan is de command-line interface voor Laravel. Het geeft de beschiking over een aantal commando's in de vorm van functies, die gebruikt moeten worden om een Laravel applicatie te kunnen bouwen. Een overzicht van beschikbaar commando's krijg je door het volgende commando uit te voeren in de terminal en in de project directorie.

```
$  php artisan list
```

Laravel model aanmaken

We hebben een standaard Laravel omgeving aangemaakt. Nu moeten we het gewenste databae model gaan bouwen. Dit doen we door het volgende artisan commando uit te voeren:

```
$  php artisan make:model Werk --migration

 Output:
 Model created successfully.
 Created Migration: 2020_03_08_191202_create_werks_table
```

Het vorige commando creert het volgende php bestand. Dit bestand wordt opgeslagen in de projectmap/database/migrations

Open in de editor omgeving het volgende gemaakte migratie bestand:

```
XXX_XX_XX_XXXXXX_create_werks_table
```

Het bestand is al redelijk ingevuld door de Laravel omgeving. Met de public function beschrijf de configuratie van de database tabel. In dit geval de tabel: werks. (de s wordt automatisch toegevoegd door Laravel.) Bestudeer dit database model eens goed. Wat valt je op?

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateWerksTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('werks', function (Blueprint $table) {
            $table->id();
            $table->timestamps();

    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('werken');
    }
}
```

Als je de het database model goed hebt bekeken, dan heb gezien dat nog niet alle gewenste kolomen van zijn verwerkt in het model. We vullen nu het bestand aan met de gewenste kolomen:

```php
   public function up()
    {
        Schema::create('werks', function (Blueprint $table) {
            $table->id();
            $table->timestamps();
            $table->string('title');
            $table->string('blog');
        });
```

```
    }
```

Nadat het model bestand is aangepast moeten opnieuw de migratie uitvoeren. Dit doen we door met het volgende commando het database model aan te passen.

```
$  php artisan migrate

Output:
Nothing to migrate.
```

Laravel controller aanmaken

Laravel werkt volgens het MVC ontwerpmethode. Dit staat voor Model View Controller. Het model, de database hebben we inmiddels gebouwd. Je gaat nu de controller bouwen. We maken een basis controller bestand aan met het volgen de commando:

```
$  php artisan make:controller WerkController --resource

Output:
Controller created successfully.
```

Laravel creert een standaard controller php bestand. Hier in zijn de volgende functies voor gedefinieerd:

- public function index()
- public function create()
- public function store(Request $request)
- public function show($id)
- public function edit($id)
- public function update(Request $request, $id)
- public function destroy($id)

Bij al deze functie in de controller wordt data uit en in de database geschreven en maken we in een later stadium een HTML view bestand om de data te tonen.

Ga naar de map in de Laravel rootmap: /app/Http/Controllers/WerkController.php
Open dit bestand in de editor.

Let op:
Indien je aanpassingen hebt gedaan in dit bestand of eventueel andere bestanden, dan moet je het bestand regelmatig bewaren in de werkomgeving.

WerkController aanpassen

Voeg aan het gemaakte bestand de volgende code in op regel 5:

```
use App\Werk;
```

Laravel routing

Route of routing functionaliteit is belangrijk voor Laravel. Want hoe weet de webserver straks welke view of wel html pagina met welke gegevens er moet worden gepresenteerd aan de gebruiker. De Laravel route is de oplossing voor dit probleem. Als softwaredeveloper moeten wel de juiste code bedenken voor het koppelen van de verschillende Models, Views en Controllers.

Route functie: basis

Ga naar de map in de Laravel rootmap: /routes/web.php
Open het volgende bestand web.php en voeg de volgende code toe.

```
Route::resource('werken', 'WerkController');
```

Bewaar en sluit het bestand. Voor Laravel zijn nu alle routes benodigde om te functioneren bekend.

Controller functie: Create

De eerste functie de we gaan maken is de Create functie.
Open daar voor het volgende bestand: app/Http/Controllers/WerkController.php

```
    public function create()
    {
        return view('werken.create');
    }
```

Bij deze functie maken een view bestand. De create functie doet namenlijk niet meer dan een view terug geven aan de browser.

Ga naar de map in de Laravel rootmap: /resources/views/
Maak een php bestand aan met de volgende naam:

```
base.blade.php
```

Vul dit bestand met standaard HTML met of zonder Bootstrap. Dit zijn vaak standaard opties in de editor die je gebruikt.
Je hebt nu een basis pagina. We gaan nu de standaard codes toevoegen voor include van de verschillende views met de uitvoer van de verschillende controller methodes. Voeg de volgende code toe aan de basis pagina: base.blade.php

```
<body>
    <div class="container">
      @yield('main')
    </div>
```

@yield('main') = Laravel code om ge-include bestanden in te lezen van de code.

Om de structuur en leesbaarheid van onze app en code te waarborgen, gaan we nu de structuur van onze app aan maken. Maak in de /resources/views/ een nieuwe map aan met de volgende naam: werken.

Werken nu even vooruit. Maak voor elke functie in de controller een bestand aan in de map: werken. Dit zijn de volgende bestanden:

```
create.blade.php
index.blade.php
```

View functie: Create

Ga naar de map in de Laravel rootmap: /resources/views/werken
Open het volgende bestand create.blade.php en voeg de volgende code toe.

```
@extends('base')

@section('main')

<div class="row">
 <div class="col-sm-8 offset-sm-2">
    <h1 class="display-3">Toevoegen</h1>
  <div>
    @if ($errors->any())
      <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
              <li>{{ $error }}</li>
            @endforeach
        </ul>
      </div><br />
    @endif
      <form method="post" action="{{ route('werken.store') }}">
          @csrf
          <div class="form-group">
              <label for="title">Title:</label>
              <input type="text" class="form-control" name="title"/>
          </div>

          <div class="form-group">
              <label for="blog">Blog</label>
              <input type="text" class="form-control" name="blog"/>
          </div>

          <button type="submit" class="btn btn-primary-outline">Toevoegen</button>
      </form>
  </div>
</div>
</div>
@endsection
```

Php artisan serve

Ga naar het terminal venster met de map Laravel rootmap: ./
Voor het volgende commando in om de Laravel server te starten:

```
$  php artisan serve
```

Bekijk daarna de volgende url in de browser:

```
localhost:8000/werken/create/
```

In de browser krijg je het gemaakte create formulier:

Toevoegen

Title:

Blog

Toevoegen

Controller functie: Store

In de het controller bestand hebben we ook een functie store. Deze functie zorgt er voor dat de data in het formulier wordt opgeslagen in de database.

Voeg de volgende code toe aan de store functie. De store functie maakt gebruik van een methode en van een $request variable. DE $request wordt gevuld met data uit het formulier. Gebruik de volgende code:

```
public function store(Request $request)
  {
       //
       $request->validate([
         'title'=>'required',
         'blog'=>'required',
       ]);

       $werk = new Werk([
         'title'=> $request->get('title'),
         'blog'=> $request->get('blog'),
       ]);
       $werk->save();
       return redirect('/werken')->with('succesvol', 'Werk opgeslagen!');
  }
```

De store functie wordt aangeroepen in het formulier in het bestand:
resources/views/werken/create.blade.php. Dit gebeurt met volgende html/laravel syntax:

```
      <form method="post" action="{{ route('werken.store') }}">
```

Controller functie: index

Met index functie in de controller willen bereiken dat alle ingevoerde data wordt getoond door de index.blade.php bestand. Hier voor moeten we de volgende code schrijven voor het controller bestand.
Open werkController.php en voeg de volgende code toe:

```
public function index()
    {
        $werken = Werk::all();
        return view('werken.index', compact('werken'));

    }
```

Met de variabele $werken vullen met door Laravel de opdracht te geven een alle data te tonen uit de gewenste tabel.

View functie: Index

Ga naar de map in de Laravel rootmap: /resources/views/werken
Open het volgende bestand index.blade.php en voeg de volgende code toe:

```
@extends('base')
@section('main')
<div class="row">
<div class="col-sm-12">
    <h1 class="display-3">Werken</h1>
  <table class="table table-striped">
    <thead>
        <tr>
          <td>ID</td>
          <td>Title</td>
          <td>Blog</td>
          <td colspan = 2>Actions</td>
        </tr>
    </thead>
    <tbody>
        @foreach($werken as $werk)
        <tr>
            <td>{{$werk->id}}</td>
            <td>{{$werk->title}}</td>
            <td>{{$werk->blog}}</td>
            <td>
                <a href="{{ route('werken.edit',$werk->id)}}" class="btn
btn-primary">Edit</a>
            </td>
            <td>
                <form action="{{ route('werken.destroy', $werk->id)}}"
method="post">
                  @csrf
                  @method('DELETE')
                  <button class="btn btn-danger"
type="submit">Delete</button>
                </form>
            </td>
        </tr>
        @endforeach
    </tbody>
  </table>
<div>
</div>
@endsection
```

Serve: index.base.php

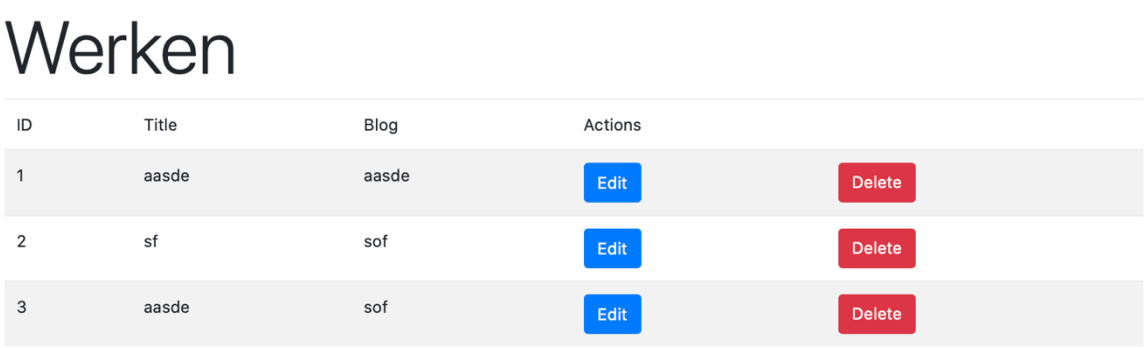
Ga naar het terminal venster met de map Laravel rootmap: ./
Voor het volgende commando in om de Laravel server te starten:

```
$  php artisan serve
```

Bekijk daarna de volgende url in de browser:

```
localhost:8000/werken/create/
```

In de browser krijg je een overzicht van de ingevoerde data:



Controller functie: overige

Er zijn nog elke functies in onze controller niet ingevuld of beschikbaar gemaakt. Dit gaan we doen met een tijdelijke code. Het voordeel hiervan is dat we de routing naar de gewenste view goed kunnen controleren.

Door in de functie de method en de $variabele terug te sturen naar de server, krijgt de browser de routing en data terug. Je doet dit met de volgende public functies in de controller aan te passen:

- public function show($id)
- public function edit($id)
- public function update(Request $request, $id)
- public function destroy($id)

Pas de volgende codes aan in de controller van WerkController.php:

```
public function show($id)
    {
        //
        return __METHOD__ . ':' . $id;
    }
```

```
public function edit($id)
    {
        //
        return __METHOD__ . ':' . $id;
    }
```

```
public function update(Request $request, $id)
    {
        //
        return __METHOD__ . ':' . $request . ':' . $id;
    }
```

```
public function destroy($id)
    {
        //
        return __METHOD__ . ':' . $id;
    }
```

Controller functie: Update / Edit

In het controller bestand hebben we ook een functie update. Deze functie zorgt er voor dat de data in het formulier wordt geupdate in de database. Om een goed werkende update functie te krijgen, moeten er eerste de edit functie worden aangepast. Het editen van een item, leidt tot een update van het item.

Voeg de volgende code toe aan de edit functie.

```
    public function edit($id)
    {
        //
        $werk = Werk::find($id);
       return view('werken.edit', compact('werk'));
```

Voeg de volgende code toe aan de update functie.

```
  public function update(Request $request, $id)
    {
      {
            $request->validate([
                'title'=>'required',
                'blog'=>'required',
            ]);

            $werk = Werk::find($id);
            $werk->title =  $request->get('title');
            $werk->blog = $request->get('blog');
            $werk->save();

            return redirect('/werken')->with('success', 'Werk updated!');
        }
    }
```

Bekijk daarna de volgende url in de browser en test de functionaliteit:

```
localhost:8000/werken/create   : om invoer te doen in de database.

localhost:8000/werken          : index, edit, update, delete
```

Controller functie: Destroy

Als laatste moeten we ook een functie maken waarbij er een item uit de database tabel kan worden verwijderd. Deze functie heeft niet de naam delete, maar destroy.

Voeg de volgende code toe aan de destroy functie.

```
  public function destroy($id)
    {
        {
        $werk = Werk::find($id);
        $werk->delete();

        return redirect('/werken')->with('success', 'Werk deleted!');
        }
    }
```

Laravel deployment op de webserver

Je Laravel project waarschijnlijk gemaakt op je laptop in ontwikkel omgeving. We gaan nu de gemaakt code deployen op een webserver. In dit geval kan dit heel goed de Ubuntu Rapsberry PI server zijn.

Push naar Gitlab of Git-server

Je moet je gemaakt Laravel project eerst plaatsen op een gitlab of git-server. Dit is een zgn. repository server. Lees eerst de uitleg en volg het stappenplan voor het plaatsen op de gitlab.glu.nl.

Project met git van de reposritory

We gaan het Laravel project van de git respository server plaasten op de Ubuntu Raspberry PI server.

```
$  cd /var/www/portfolio
```

```
$  git init

$  git pull https:// …….. : adres van de server

$  ls

Output:
root@ubuntu:/var/www/portfolio# ls
README.md  artisan     composer.json  config    package-lock.json  phpunit.xml  resources  server.php  tests
app        bootstrap   composer.lock  database  package.json       public       routes     storage     webpack.mix.js
root@ubuntu:/var/www/portfolio#
```

Foutmelding Git

Krijg je de volgende foutmelding, dan moet je de git config aanpassen. Doe je dit –global, dan gebruikt GIT dit altijd op de server. Doe je dit –local, dan is dit alleen voor de huidige configuratie.

```
fatal: unable to access 'https://gitlab.glu.nl/……/…….git/': server
certificate verification failed. CAfile: none CRLfile: none



Oplossing:

git config --global http.sslVerify false
```

Aanpassen van de Nginx is noodzakenlijk. De hoofd map in een laraval project is namenlijk de directorie met de naam Public. We gaan het volgende aanpassen.

```
$  sudo nano /etc/nginx/sites-available/Laravel
```

Voeg de directorie public toe aan de al bestaande regel.

```
root /var/www/portfolio/public;
```

```
$  sudo systemctl restart nginx
```

Laravel env.configuratie aanpassen

```
$  sudo cp .env.example .env

$  sudo nano .env
```

Nu gaan we het bestand aanpassen. Je de app_url aanpassen aan je eigen ip of domain naam. Daarna vul de gegevens bij DB_ in. Als deze turorial helemaal hebt gevolgd, dan gebruik je de gegevens van de gebruiker die je hebt aangemaakt in de mysql omgeving.

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:
APP_DEBUG=true
APP_URL=http://IP-ADRESS OF DOMAIN NAME

LOG_CHANNEL=stack

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=portfolio_list
DB_USERNAME=portfolio_user
DB_PASSWORD=password
```

Je moet nog rechten toekennen aan de Laravel mappen om de webserver user toegang te verlenen op deze mappen.

```
sudo chgrp -R www-data storage bootstrap/cache
sudo chmod -R ug+rwx storage bootstrap/cache
```

Laravel composer install

De gemaakte Laravel applicatie staat nu op de server, maar werkt nog niet helemaal. Met de volgende stappen maak de configratie gereed op de webserver.

```
$  cd /var/www/portfolio

$  sudo composer install (negeer de opmerking over sudo)
```

Nu is de configuratie compleet en moet het project worden geconfigureerd met de volgende commando's.p

```
$  sudo php artisan key:generate

$  sudo php artisan db:seed

$  sudo php artisan migrate --seed
```

Bekijk het project in de browser met het gebruikte IP-adress.