Laravel 8

Episode 5.

Return view: \Routes\web.php

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

Voorpagina: \resources\views\welcome.blade.php

Episode 6.

Maak css/js in: \public

Episode 8.

Variabele:

```
href="$variabeleNaam"
```

```
Route::get('variabeleNaam', function () {
    return view('variabeleNaam',[
        'variabeleNaam' => '<h1>Hello World</h1>'
    ]);
});
```

Als error message kan je gebruiken:

```
if (! file_exists($path)) {
      dd('File does not exist');                    // dump & die
      ddd('File does not exist');                   // dump, die & debug
      abort(404);                                   // display standard 404
       return redirect('/');                        // redirect to link: ('/');
```

Werkende functie:

```
Route::get('folder/{wildcard}', function ($slug) {
    $path = __DIR__ . "/../resources/folder/{$slug}.html";

    if (! file_exists($path)) {
        abort(404);

    }

    $post = file_get_contents($path);

    return view('wildcard', [
        'wildcard' => $post
    ]);
});
```

1. Get “folder/{wildcard}” (‘{wildcard}’ kan elk bestand zijn in die folder)

2. “Fuction ($slug)” ; slug laat de gebruiker de url veranderen om op jouw andere pagina’s te komen.

3. Variabele “$path” is het pad dat de functie neemt om bij de bestanden te komen.
   De “{slug}.html” geeft aan dat er verschillende pagina’s weergegeven kunnen worden op basis van wat de gebruiker invoert.

4. “if (! File path_exists($path)” ! = False, dus als File path niet bestaat, voer het volgende uit, in dit geval “abort(404);”.

5. Variabele ‘$post’ haalt de inhoud van het bestand van variabele ‘$path’ op

6. De inhoud van ‘$post’ wordt doorgegeven aan ‘wildcard’ en de view (webpagina) wordt geretourneerd.

Episode 9

Zet restricties op mogelijke invoer van ‘wildcard’ door op het einde van de functie te typen:

```
    ]);
})->where('wildcard', /*regex*/);
```

Ingebouwde regex voorbeelden:

```
    ]);
})->whereAlpa($parameters);


    ]);
})->whereAlphaNumeric($parameters);


    ]);
})->whereNumber($parameters);
```

Episode 10.

Cache gegevens (zodat de hele functie zich niet hoeft te herhalen elke keer dat de pagina geladen wordt):

Van:

```
    $post = file_get_contents($path);
```

Naar:

```
    $post = cache()->remember("folder.{$slug}", 5, function () use ($path){

        return file_get_content($path);

    });
```

1. Variabele post is gelijk aan:
2. Gebruik ‘remember’ uit de ‘cache’ functie.
3. Remember de unieke key “(“folder.{$slug}”)”
4. Remember het 5 seconden lang.
5. Maak een functie en haal variabele ‘$path’ op (anders werkt het niet want het is buiten deze functie)
6. Return de content uit ‘$path’

I.p.v. tijd in seconden gebruik:

```
now()->add…(),
```

reëlere manier van coderen

```
now()->addMinutes(30),
now()->addHours(12),
now()->addDays(6),
now()->addWeeks(2),
```

etc.

Episode 11.1

Maak eigen functies/methods in een extern bestanden voor cleanere code

Ga naar \App\Models en maak een nieuwe folder.

Episode 12

Maak met behulp van metadata bepaalde criteria om te bepalen welk bestand opgehaald moet worden. Ook kan je specifieke onderdelen ophalen van een bestand zoals specifiek de h1 bijvoorbeeld.

Episode 13

Hij gebruikt de cache methode van episode 10 in zijn eigen functie in de 'Models' folder met remember forever:

```
    return cache()->rememberForever('folder.all', function (){

});
```

'all' is in dit geval een functie die hij zelf heeft aangemaakt in een andere model.

Episode 14

Blade zorgt voor een makkelijkere manier van coderen, en vertaald het automatisch naar basis php in de '\framework\views' folder. Het volgende voorbeeld is iets wat makkelijker gaat met blade:

Blade

```
    @foreach ($post as $posts)
```

PHP

```
    <?php @foreach ($post as $posts) : ?>
```

De '$loop' methode geeft de verschillende manieren van checken of iets overeenkomt zoals:

- Remaining — Hoeveel zijn er over van een bepaald element?
- Count — De hoeveelste is dit element in een bepaalde telling?
- Odd — Is dit element een oneven getal binnen de telling?
- Even — Is dit element een even getal binnen de telling?
- First — Is dit element de eerste binnen de telling?
- Last — Is dit element de laatste binnen de telling?

Etc.

Episode 15

Maak een standaard layout ‘variabele’, waardoor je vaker voorkomende code kan vermijden zoals:

```
<!DOCTYPE html>
```

Maar ook wanneer je in meerdere bestanden wilt linken naar dezelfde stylesheet:

```
    <link rel="stylesheet" href="assets/styles/style.css" />
```

Zet deze code in een apart bestand met ‘@yield’ in de body:

```
  <body>
    @yield(‘content’)
  </body>
```

Voeg dit aan het begin van het andere bestand toe om het de code door te geven:

```
      @section(‘content’)
```

En als afsluiting

```
      @endsection
```

Hetzelfde effect bereik je met blade’s functie:

```
  <x-slot name= “content”>
    Testing!
  </x-slot>
```

En in het andere bestand:

```
  <body>
     {{ $slot }}
  </body>
```

Episode 16

-

Databases

Episode 17

In het '.env' bestand kun je belangrijke informatie declareren wat je niet zichtbaar wilt hebben (bijvoorbeeld wachtwoorden). Deze worden hierdoor weergegeven als de key die jij ze hebt gegeven, maar hebben de inhoud van de deze keys hebben de nodige informatie:

```
Key=informatie
```

Bijvoorbeeld:

```
DB_CONNECTION=mysql
```

In andere bestanden refereer je nu ook alleen nog maar naar DB_CONNECTION.