## Forms

Er zijn 7 restful controller acties (conventie/richtlijn):
`index`: toont een lijst van iets (artikelen, gebruikers, etc
`show` toont een specifiek item (artikel, gebruiker etc)
`create` toont een view om item aan te maken.
`store` slaat item op.
`edit` Toont een view om bestaand item aan te passen.
`update` slaat wijzigingen op.
`destroy` verwijdert item.

Met de artisan optie -r kunnen deze automatisch aangemaakt worden bij het maken van een controller. Met de `-m Modelnaam` optie erbij wordt een bijpassend database model aangemaakt.

Browsers ondersteunen eigenlijk allen GET en POST requests. Om toch een PUT of ander soort request te simuleren vanuit een form, stel het formulier in op POST en voeg een `@method('PUT')` toe in het formulier.

Bij het doen van een POST, bijvoorbeeld vanuit een form, kan een 419 error ontstaan als resultaat van bescherming tegen XSS expoits. Door het toevoegen van een `@csrf` statement in de view na het <form> element wordt deze fout voorkomen.

## Algemeen

Alle resources (afb, JS, etc) in de `public` map.

## Routing

Routes worden gedefiniëerd in `routes/web.php`

Een route wordt gedefinieerd met
`Route::get('url', $data);`
of
`Route::get('url', function(){});`
of
`Route::get('/url/{variable}', function($variable){});`
of (om functie functionname in class controllerclass aan te roepen)
`Route::get('/url/{variable}', 'controllerclass@functionname');`

Data die naar de view moet worden gestuurd moet gedaan worden in de closure functie in de route
`return view('viewnaam', ['key' => 'value']);`

POST of GET argumenten kunnen worden opgevraagt met de `request()` helper

Foutafhandeling (404 pagina) wordt gestart met `abort(int $statuscode, string 'bericht')`

Naast `Route::get('url', $data)` zijn er ook `Route::post`, `Route::put` en `Route::delete`

Om toekomstbestendig te zijn kun je routes benoemen, als volgt:
`Route::get('/artikles/{artikle}', 'Artikles@show')->name('artikles.show');`
In blade kan dan route worden weergegven met
`{{ route('artikles.show', $artikle->id) }}`

## Artisan

Een nieuwe controller aanmaken:
`make:controller ControllerNaam`

Een eloquent model aanmaken:
`make:model Modelnaam`

Een migratie aanmaken:
`make:migration create_table_name_table --create=table`

Een migratie toepassen:
`migrate`

Een migratie rollback:
`migrate:rollback`

Database legen en migraties opnieuw uitvoeren
`migrate:fresh`

Een migratie, factory en controller maken voor model:Project -a

## Blade language

Escape variabelen met
`{{ $variabele }}`

Variabelen expliciet NIET escapen met
`{!! $variabele !!}`

Elementen in objecten weergevem met
`{{ $object->element }}`

Een template kan bouwen op een andere door `@extends('bestandsnaam')` commando

Er kan een sectie uit een template-bestand worden ingevoegd op de huidige positie met het `@yield('sectienaam')` commando.

Een sectie word gedefinieerd met
`@section('sectienaam')`
en geëindigd met
`@endsection`

## Blade language (cont)

Veel PHP is gewoon beschikbaar, zoals `Reques t:: path()`

Meerdere items zijn te plaatsen als

```
@foreach ($articles as $article)
<li>Hier invoegen met {{ articl e-> title }}</li>
@endforeach
```

De asset map kan worden aagegeven met `{{ asset }}subm a p/ bes tan dsn aam.css`

Foutmeldingen (zoals bij formvalidatiefouten) kunnen worden weergegeven met `{{ $error s-> fir st( 've ldn aam') }}`

Alleen weergeven als een variabele/attribuut bestaat: `@if ($erro rs- >ha s(' tit le) )<h tml >@endif`

Inline text plaatsen met conditie: `{{ $error s-> has ('t itle') ? 'tekst' : '' }}`

Bij foutmelding iets weergeven: `@error ('v eld naa m') <b> uw tekst hier </b >@e - nderror`

Een veldwaarde van een vorige submit invullen (zoals bij het falen van formvalidatie): `{{ old('v eld naam') }}`

## Migrations (Database)

Een migratie is een layout voor de database, of een aanpassing daaraan.

Bestanden met migraties staan in `databa - se/ mig rations`

Voor development kunnen het beste bestaande migraties gewijzigd worden. Voor wijzigingen aan live servers maak een nieuwe migratie.

## Controllers

Controllers staan in `Http/C ont rollers`

Request data kan worden opgevraagd met de `reques t( 'vel dnaam)` functie.

De `request()` functie (zonder argumenten) retourneert een object met `valida te( array $opties)` functie, zoals dit:

```
request()->validate([
'title' => ['requ ired', 'min:3', 'max:255']
,
'excerpt' => 'required',
'body' => 'required'
]);
```

De validate() functie retourneert bovendien een array met valide elementen die gerefereerd kunnen worden bij verdere verwerking van de formuliergegevens.

In controllers wordt vaak gegevens uit een DB gehaald met functies als `$article = Articl e:: fin dOr Fai l($id);` er vanuitgaande dat er een eloquent model Article bestaat. In plaats daarvan kunnen we ook Laravel om het eloquent model vragen als argument op de functie, zoals

```
public function show(A rticle $article){
//code hier
}
```

De variabelenaam moet hierbij overeenkomen met een wildcard uit de route.

## Model (Database)

Configuratie in `config /da tab ase.ph p` en in `.env` file in de root

## Model (Database) (cont)

Eloquent modellen zijn classes die een databa kunnen uitgevoerd worden via deze class. Dez functies om die tabel te manipuleren. Het resu model is gerepresenteerd in diezelfde class.

Indien een eloquent model is aangemaakt kun model importeren zoals `use App\Mo delnaam` en dan `Modeln aam :wh ere ('k olo mnaam',`

Queries zien eruit als: `\DB::t abl e(' tab eln aam ')- >wh e)- >fi rst();`

`->f irs tOr Fail()` haalt 1 waarde op of `findOr Fail()`.

Eloquent modellen bevinden zich in `app/`

de `all()` functie haalt alle records uit een tab functie (eerste resultaat), `take(int $aanta` ophalen) en `pagina te(int $aanta l-p` ophalen).

De `latest ('d atu mve ld- rec ord r` record op. `find($id)` haalt een record op me

## Model (Database) (cont)

Wildcards in routes worden standaard gematched aan het id veld in een database. Om de wildcard naar een ander veld te laten matchen, definiëer de functie

```
public function getRouteKeyName(){
return 'keynaam';
}
```

in het eloquent model.

Een nieuw record aanmaken kan als volgt:

```
$article = new EloquentModelNaam();
$article->title = request('title');
$article->excerpt = request('excerpt');
```

Maar ook als:

```
EloquentModelNaam::create([
'title' => request('title'),
'excerpt' => request('excerpt')
]);
```

Bij deze laatste methode moet er opgepast worden: de array zou onbedoeld meer velden kunnen bevatten dan bedoeld. Daarom moet er bij Laravel in het eloquent model worden aangegeven welke velden met een publiek form in te vullen zijn door een class property $fillable te definiëren:

```
protected $fillable = ['title', 'excer pt']
;
```

Alle velden expliciet **niet** bewaken kan met

```
protected $guarded = [];
```

Relaties tussen eloquent models kunnen worden gedefiniëerd in een functie in een eloquent model. Bijvoorbeeld

```
return $this- >ha sMa ny( Art icl e:: class
);
```

geeft een relatie aan met meerdere articles.

## Views

Views worden opgeslagen in `resour ces /views`

Gegenereerde resources worden opgeslagen in de `resources` map. Overige in de `public` map.

In de `/webpa ck.m ix.js` wordt aangegeven wat voor bestanden gecompileerd moeten worden, zoals SASS en enkele JS bestanden. Dit wordt gedefinieerd met een regel zoals

```
mix.sa ss( 're sou rce s/s ass /be sta nd.s css', 'publi c
/d oel map');
```

Het bestand `/packa ge.json` definieert dependencies. Deze dependencies moeten met de hand geïnstalleerd worden met npm (op te halen via nodejs.org), met het commando `npm install`.

Resources compileren wordt in gang gezet met `npm dev`.

`npm run watch` houdt resources in de gaten en compileert ze automatisch.