

Scriptie ingediend tot het behalen van de graad van
PROFESSIONELE BACHELOR IN DE ELEKTRONICA-ICT

The Pitchpoint agency tool

Nick Lauwerijs

academiejaar 2015-2016

AP Hogeschool Antwerpen
Wetenschap & Techniek
Elektronica-ICT

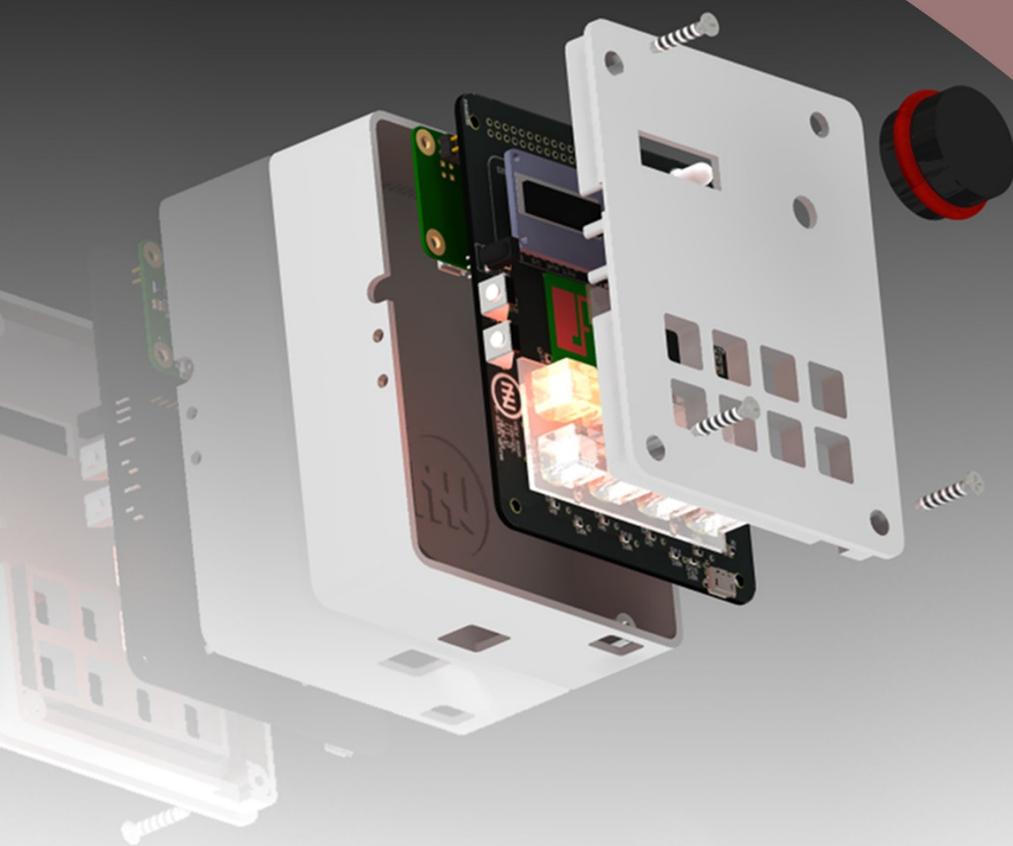


Table of Contents

abstract	1.1
dankwoord	1.2
introduce	1.3
Situering	1.3.1
opdracht	1.3.2
probleemstelling/opgave	1.3.3
Technisch	1.4
Gebruikte technologieën	1.4.1
Laravel	1.4.1.1
Wat is Laravel ?	1.4.1.1.1
Applicatie structuur	1.4.1.1.2
Conventie over configuratie	1.4.1.1.3
artisan	1.4.1.1.4
model-view-controller	1.4.1.1.5
database	1.4.1.1.6
composer	1.4.1.1.7
bootstrap	1.4.1.2
Sass	1.4.1.3
gulp	1.4.1.4
Bespreking ontwikkeling	1.4.2
Gebruikte tools	1.4.3
Sublime Text	1.4.3.1
Mamp pro	1.4.3.2
heidisql	1.4.3.3
commander	1.4.3.4
chrome	1.4.3.5
Conclusie en samenvatting	1.5
Uitbreidingen/Future work	1.6
Kleinere opdrachten	1.7
html email	1.7.1
website voor Nucleaire Forum	1.7.2
Gulp	1.7.2.1
Sass	1.7.2.2
Susy	1.7.2.3
tweenmax.js	1.7.2.4
appendices	1.8
Bibliografie/Geraadpleegde bronnen	1.9

Abstract

Prophets is een digitaal marketing bureau gesitueerd in Antwerpen. Tijdens mijn stage hier moest ik werken aan een web applicatie voor het bedrijf Pitchpoint. De ontwikkeling deed ik samen met Veerle De Vos (andere stagiaire van Prophets).

Pitchpoint helpt adverteerders om een passend communicatiebureau te vinden, maar dit is niet altijd even makkelijk. Daarom vroegen ze om een web platform te ontwikkelen waar communicatiebureau's zich kunnen registreren zodat ze een overzicht hebben van mogelijke kandidaten.

Deze scriptie beschrijft hoe dit project structureel in elkaar zit , de technologieën en software die we hebben gebruikt en hoe het project verliep.



Dankwoord

Na een intensieve stage periode waarin ik heel veel heb bijgeleerd omtrend Web Development zou ik ten eerste graag Kris Van Hauwermeiren bedanken die deze stage plaats heeft aangeboden en mij heeft begeleid gedurende de stage. Ook zou ik graag Samuel Joos en Peter Van Wyck bedanken omdat deze altijd klaar stonden voor eventuele vragen en advies en ook omdat deze mijn code hebben nagekeken en hierop heel nuttige feedback hebben gegeven.

Verder wil ik mijn interne promotor Patrick Van Houtven bedanken omdat hij zijn tijd en moeite heeft gestoken in het verbeteren van al de documenten die ik moest inleveren.

introductie

Bedrijf

Prophets is een digitaal marketingbureau gesitueerd in Antwerpen. Mijn opdracht was om te werken aan projecten voor klanten van Prophets. Eerst ben ik enkele weken bezig geweest om een website te maken voor het [Nucleaire Forum](#). De rest van de stage heb ik gewerkt aan een web applicatie voor Pitchpoint. Aangezien ik het meeste van de tijd aan het Pitchpoint project heb gewerkt heb ik de scriptie hierover geschreven.

Probleem

Het is moeilijk om een overzicht te hebben van mogelijke kandidaat marketingbureau's om projecten aan te geven. Pitchpoint probeert een zo goed mogelijke match te vinden tussen de opdrachtgever en het communicatiebureau. Hierbij is veel info nodig om de beste kandidaat te vinden.

Doelstelling

Een web applicatie ontwikkelen waarop communicatiebureau's zich kunnen registreren. Er moet heel wat data worden ingegeven zoals: Belangrijke werknemers , expertises , klanten , filosofie,... dit moet achteraf ook worden kunnen aangepast, omdat de info kan veranderen met de tijd.

Dit alles moet worden weergegeven in een admin sectie zodat Pitchpoint een overzicht heeft van elk communicatiebedrijf. Deze agencies kunnen in detail bekeken en volledig aangepast worden door Pitchpoint zelf. Per Agency heeft de Pitchpoint ook de mogelijkheid om commentaar en sterke punten toe te voegen.

Om deze applicatie te ontwikkelen hebben we gebruik gemaakt van een PHP framework genaam Laravel in combinatie met enkele andere technologieën zoals Bootstrap, Sass, JQuery, Gulp.

Technisch

Wat is laravel ?

Laravel is een open-source PHP web-framework die gebruikt wordt voor de ontwikkeling van web applicaties. Laravel kent een grote populariteit omdat het framework de reputatie heeft gemakkelijk in gebruik te zijn en omdat het voorzien is van een heel leesbare syntax. Ook is het framework uitstekend [gedocumenteerd](#) en heeft het een e-learning service genaamd [laracasts](#) die de developer helpt bij het gebruiken van het framework.

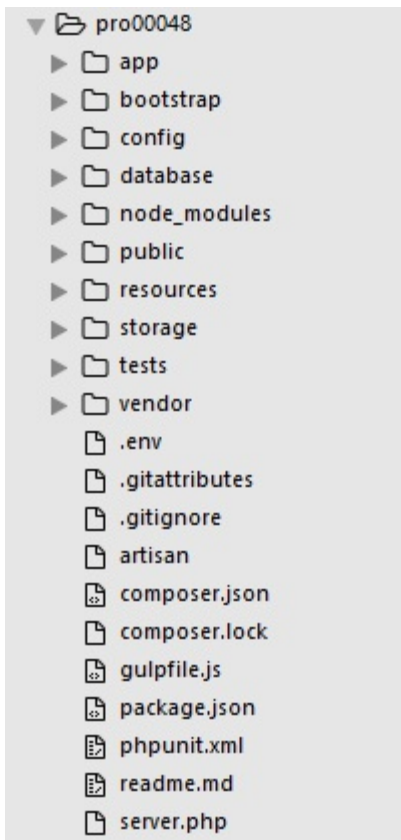
Het wordt een full-stack framework genoemd omdat het elk onderdeel van een web-applicatie kan hanteren. Dit gaat van data opslaan in de databank tot het renderen van views door de built-in templating engine. Veel componenten die regelmatig terugkomen bij web applicaties zijn al in het framework ingebakken. Zo verliest de developer geen tijd meer met functionaliteit zoals authentication, sessions, caching, ... te implementeren.

Ook bieden ze de developer de mogelijkheid tot het gebruik van Laravel Homestead. Dit is een virtuele development environment met alle mogelijke tools die nodig zijn om laravel in gebruik te nemen zoals :

- Ubuntu 14.04
- Git
- PHP 7.0
- HHVM
- Nginx
- MySQL
- MariaDB
- Sqlite3
- Postgres
- Composer
- Node (With PM2, Bower, Grunt, and Gulp)
- Redis
- Memcached
- Beanstalkd

Applicatie structuur

Laravel heeft wat beperkingen als het gaat over applicatie structuur. Het forceert de ontwikkelaar een bepaalde mappenstructuur te volgen. In het begin lijkt dit vrij ingewikkeld omdat er zoveel mappen en submappen zijn, maar snel wordt duidelijk welk voordeel dit biedt. Omdat elke ontwikkelaar geforceerd wordt om dezelfde structuur aan te houden resulteert dit op een overzichtelijke mappenstructuur die in elk laravel project hetzelfde is.



App: de core code van de applicatie , dit houd in : models, controllers,

middleware,... **Bootstrap:** enkele files voor het configureren van autoloading, ... (framework files) **config:** folder waarin alle configuratie files te vinden zijn. Bijvoorbeeld: database, mail, authenticatie, ... **database:** folder waarin alle migration en seeder files te vinden zijn, deze worden gebruikt om aan database management te doen en de database te vullen met data. **node_modules:** folder die alle packages bevat die binnengehaald zijn door de npm package manager. **public:** ++dit is de folder waarin alle css, js files worden bewaard. De root van de server moet wijzen naar deze map want de frontcontroller + alle assets staan hier in.++ **storage:** deze folder bevat door framework gegenereerde files en applicatie log files. **tests:** deze folder bevat al de tests van het project. **vendor** deze folder bevat alle dependencies die zijn ingevoerd met composer.

Conventie over configuratie

Het framework volgt het principe van conventie over configuratie , dit wil zeggen dat laravel bijna geen configuratie nodig heeft om te starten. Het framework beperkt het aantal keuzes dat de developer moet maken door default configuratie waardes te gebruiken. Uiteraard kunnen deze default waardes verandert worden.

Artisan

Artisan is een command-line tool die bij het laravel framework hoort , deze tool kan de developer helpen door een aantal behulpzame commands. Wij hebben gebruik gemaakt van :

- **db:seed**
 - Vult de database met dummy data om zo de applicatie te kunnen testen met werkelijke data in de database
- **make:Auth**
 - genereert de authenticatie logica en views.
- **make:controller**
 - genereert een controller met boilerplate code.
- **make:middleware**
 - genereert middleware met boilerplate code.

- **make:model**
 - genereert een model met boilerplate code.
- **make:seeder**
 - genereert een seeder met boilerplate code.
- **make:migration**
 - genereert een migration met boilerplate code.
- **migrate**
 - opzetten van de databank aan de hand van migrations.
- **migrate:refresh**
 - na het toevoegen van een nieuwe migration moest deze command worden uitgevoerd om de databank volledig opnieuw op te zetten.
- **session:table**
 - Creeërt migration voor de sessie tabel in de DB.

Model-View-Controller

Laravel volgt het MVC (model-view-controller) design pattern , wat wil zeggen dat onze applicatie in 3 delen word gesplitst.

- **Model** Het model representeerd de data. Het is volledig onafhankelijk van de view of controller.
- **View** De view geeft de model data weer, users kunnen interacteren met de view en user acties kunnen verzonden worden naar de controller.
- **Controller** De controller doet het ontvangen , interpreteren en valideren van user input, data opvragen en updaten van views. Het is een scheiding tussen de view en het model.

De applicatie logica word gescheiden van de presentatie logica. Deze aanpak zorgt voor een vermindering aan complexiteit in de applicatie.

Database

migrations

Migrations laten ons toe om gemakkelijk in een team met een databank te werken. Telkens een nieuwe table moet aangemaakt worden moet er een migration file worden toegevoegd. Als nu iemand anders van het team pult van git dan zal hij deze nieuwe migration files in zijn project krijgen en zal hij zijn database kunnen bijwerken door het `php artisan migrate` command uit te voeren. Ook een handig voordeel met het werken van migrations is dat als er iets misloopt met de databank dan kan men deze helemaal opnieuw aanmaken met Artisan. Migrations zijn een soort van version control van je databank , het houdt een geschiedenis bij van wat er allemaal verandert is gedurende de project duur. Er kan dus makkelijk gezien worden wat er allemaal is verandert.

Migrations kunnen gegenereerd worden met artisan en bestaan uit telkens 2 functies :

- **up** Hierin worden er nieuwe tables , kolommen of indexen toegevoegd aan de databank.
- **down** Hierin worden de nieuwe tables , kolommen of indexen die in de up methode werden aangemaakt ongedaan gemaakt.


Seeders

Om de applicatie te testen is het handig dat de ontwikkelaar wat test data kan gebruiken. Dit kan gedaan worden door het gebruik van seeders. Een seeder classe kan gegenereerd worden met Artisan, deze classe bevat maar één functie genaamd run en zal aangesproken worden als het artisan seed command word uitgevoerd. In deze functie word er data in de database opgeslagen door gebruik van de laravel query builder, men kan ook grote hoeveelheden van data random genereren door gebruik van de model factories.

Model

Voor dat de seeders uitgevoerd kunnen worden moet het model aangemaakt worden. Models worden aangemaakt door dit command : `php artisan make:model modelName` . Het aangemaakt model zal geplaatst worden in de map `App` . Accessors worden gebruikt om attributen te formateren wanneer deze uit de database worden gehaald. In dit project hebben we onderstaande accessors gedefinieerd:

- **welke fields er mass-assignable moeten zijn**
 - Als een field als fillable word gedefinieerd dan zal deze mass-assignable zijn. Dit wil zeggen dat als er een input array word verstuurd als de gebruiker zijn form submit. De fields die in deze array zitten en fillable zijn zullen in 1 keer kunnen worden opgeslagen. En niet allemaal apart , als deze fields niet worden aangeduid als fillable dan zou alles mass assignable zijn en dan kan men iets meesturen met de input dat normaal niet verandert zou mogen worden bv: `is_admin=true`
- **welke fields er gecast moeten kunnen worden**
 - Booleans worden in de databank opgeslagen als TINYINT (=0 of 1) maar als we deze waardes uit de databank halen , is het handig dat we verder met het boolean type kunnen werken , dit is mogelijk door in het model deze fields te definiëren en te zeggen naar welke data-type deze gecast moeten worden.
- **welke fields hidden zijn.**
 - als er bv: users uit de databank moeten gehaald worden dan zal het password meegegeven worden maar als er een hidden attribute password word gedefinieerd dan zal dit niet meer gebeuren.

Laravel maakt gebruik van een ORM(object-relational-mapper) genaamd Eloquent. Eloquent voorziet een object georiënteerde query taal zodat de ontwikkelaar zich kan concentreren op het object model in plaats van de database structuur. Eloquent zal deze taal vertalen in de juiste syntax voor de database. Elke Table heeft een bijhorend model dat gebruikt word om data uit de database te halen en toe te voegen. Om models aan te maken maakt men gebruik van de command line tool genaamd artisan , deze tool zit ingebakken in het framework. Door het command `php artisan make:model User` uit te voeren zal er een model genaamd User worden aangemaakt. Als er nergens expliciet word gedefinieerd welke table er gebruikt moet worden voor het model zal eloquent het meervoud van de modelnaam + enkel kleine letters uit de db gebruiken als table. bv. model User zal standaard de users table uit de db gebruiken. In deze models zullen ook de relaties tussen elkaar gedeclareerd moeten worden want de User zal bv geassocieerd worden met een agency en een agency zal dan weer geassocieerd worden met enkele klanten. Zo zijn er wel meerdere relaties die gedefinieerd moesten worden , hieronder een overzicht van ons databank schema waar je makkelijk kan zien welke relaties er bestaan in onze applicatie.  Om deze relaties te definiëren zal men eerst moeten bepalen welk type relatie er gelegd moet worden. Zo bestaan er een aantal types (die wij gebruikt hebben):

1. One To One relationship Bv: one Agency has one User account => om deze relatie te definiëren moeten we in het Agency model een methode user plaatsen met hierin de functie :

```
public function user()
{
    return $this->hasOne('App\User');
}
```

2. One To Many relationship Bv: one Agency has many Employees om deze relatie te definiëren moeten we in het Agency model een methode user plaatsen met hierin de functie :

```
public function employees()
{
    return $this->hasMany('App\Employee', 'agency_id')->get();
}
```

De relatie moet ook andersom gedefinieerd worden zodat we bv de agency kunnen achterhalen aan de hand van een employee. Dit word gedaan aan de hand van deze methode toe te voegen.

```
public function agency()
{
    return $this->belongsTo('App\Agency');
}
```

Nu kunnen we beginnen met de controllers te gebruiken zodat we de juiste data in de view kunnen krijgen , maar eerst wat info over hoe de gebruiker navigeert tussen verschillende pagina's.

Routes

De gebruiker kan naar verschillende pagina's navigeren met behulp van routes. Deze routes dienen zelf gedefinieerd te worden in de file `app/Http/routes.php`. Als er nu een request word gedaan door de client en deze is gelijk aan de gespecificeerde route uri, dan zal de server weten welke controller er moet worden aangesproken om een bepaalde functie uit te voeren en/of de juiste pagina terugsturen naar de client. Bv: `Route::get('/{id}/step1', 'AgencyController@showStep1');` deze route verteld ons dat als je naar `/2/step1` surft dan zal de `id=2` meegegeven worden als parameter in de `showStep1` functie in de `AgencyController` en dan zal de eerste stap terug gestuurd worden.

resourceful routes

We hebben ook gebruik gemaakt van resourceful routes , dit zijn routes die geregistreerd worden voor een resource controller. Als men een controller aanmaakt dat bv requests verwerkt aangaande werknemers kan men gebruik maken van resource controllers. Deze kunnen aangemaakt worden via artisan door het command : `php artisan make:controller EmployeeController --resource` uit te voeren. Hierna zal er een resource route worden gedefinieerd die geassocieerd word met deze controller. Deze resource route zal meerdere routes bevatten die we nodig hebben om data op te slaan , data te updaten , data te verwijderen en data weer te geven. Deze routes zullen naar methodes linken in de controller. Al deze methodes worden ook al gegenereerd bij het aanmaken van de controller. Zo hebben we enkele resourceful routes gecreeërd samen met hun bijhorende controller:

- voor werknemers van een agency
- voor awards van een agency
- voor referenties van een agency
- voor klanten van een agency
- voor commentaar over een agency
- voor sterke punten van een agency

HTTP Middleware

HTTP Middleware gebruiken we als filter om op bepaald requests een gepaste response te geven. Bv. middleware kan controleren of de persoon die de request stuurt geauthenticeerd is of niet. Als deze persoon ingelogd is dan zal hij naar register step 1 worden gestuurd en anders naar de login page. Welke middleware gebruiken wij in onze applicatie ?

- **web group middleware** Word gebruikt voor cookies , sessies en Cross site request forgery protection. Deze middleware word toegepast op elke route in onze applicatie omdat overal sessies en cookies en CSRF protection nodig zijn deze middleware zit ingebakken in het framework.
- **Auth middleware** Word gebruikt bij routes die enkel toegankelijk zijn voor geauthenticeerde users. Als user niet geauthenticeerd is dan zal deze een pagina krijgen met een 'unauthorized' message. Deze middleware komt ook out of the box bij het laravel framework.

- **Admin middleware** Word gebruikt bij routes die enkel toegankelijk zijn voor geauthenticeerde users met admin rechten. Deze middleware hebben we zelf geschreven.

Middleware kan aangemaakt worden via artisan door het command `php artisan make:middleware Middlename` uit te voeren. Om deze middleware te gebruiken dienen we te declareren op welke routes deze middleware moet uitgevoerd worden. Dit gebeurt in de route.php file waar alle routes staan gedefinieerd. Men kan ook groepen maken van routes waar deze middleware moet worden uitgevoerd. Elke keer als er nu een request verstuurd wordt naar een bepaalde route met middleware zal eerst de code van de middleware worden uitgevoerd voordat de controller aangesproken zal worden.

Controllers

Controllers worden gebruikt om gerelateerde http requests te verwerken in een klasse. Zo kunnen we, zoals bij routes vermeld, een route aanmaken die deze controller actie zal triggeren. Wij hebben 2 "hoofd" controllers :

Admin controller

In de admin controller worden (bijna) alle requests verwerkt die uit het admin gedeelte van de webapplicatie komen. Als er wordt ingelogd met een admin account zal de `index()` functie uitgevoerd worden.

- **index()** Deze functie zal data uit de databank halen om een overzicht te kunnen weergeven van alle geregistreerde agencies en de user sturen naar de dashboard pagina.
- **show(\$id)** als er nu op een agency wordt geklikt in het dashboard dan zal deze functie worden aangesproken in de controller en dan zal de controller een nieuwe view terug sturen namelijk het overzicht van het geklikte agency met alle data die nodig is om dit weer te geven. Het `$id` vertelt de controller op welke agency er is geklikt.
- **store(\$id , Request \$request)** In het agency overzicht zal de admin commentaar, sterke punten en een laatste contactdatum kunnen toevoegen. Dit opslaan naar de databank zal gebeuren door deze functie. In de request parameter zit de data die opgeslagen moet worden en het `$id` vertelt de controller naar welk agency er moet worden opgeslagen.

Agency controller Deze controller zal zich bezig houden met het verwerken van requests aangaande het registratie proces.

- **showStep1(\$id)** deze functie zal uitgevoerd worden wanneer een user op de registreer knop drukt. De controller zal hierna de view van de eerste registratiestap terug sturen naar de client.
- **show(\$id , \$stepId)** Deze functie zal alle verschillende stappen van de registratie weergeven met de correcte data , want alle data die al ingevuld is geweest moet zich terug vullen in het form zodat geëditeerd kan worden. Als parameters gebruiken we `$id` en `$stepId` , `$id` gebruiken we voor de correcte data van het juiste agency terug te sturen en `$stepId` gebruiken we zodat de controller weet welke stap er moet weergegeven worden. telkens dat deze functie wordt uitgevoerd zal deze ook updaten tot welke stap van het registratieproces de gebruiker alles al heeft ingevuld , dit wordt allemaal opgeslagen in de database. Deze waarde wordt gebruikt om een afbakening te maken tot waar de gebruiker allemaal kan navigeren : de gebruiker kan navigeren tot de stap na de laatst ingevulde stap , zodat er geen stappen kunnen overgeslagen worden. Deze functie zal ook een mail sturen naar de gebruiker als deze de registratie volledig heeft doorlopen. Deze mail bevat een bedankje voor de registratie en een link waar het account kan geëditeerd worden. Ook zal er een mail gestuurd worden naar pitchpoint zelf als een nieuw agency zich heeft geregistreerd , deze mail bevat de naam van het geregistreerde agency en een link naar de admin pagina van pitchpoint.
- **store(\$id , \$stepId , Request \$request)** Deze functie zal alle ingevoerde data van het registratie proces verwerken en opslaan in de databank. Telkens er een submit wordt gedaan in het registratieproces zal deze functie checken op welke stap er juist gesubmit is , hierna zal hij de juiste validatie regels in de validator steken. De data zal gevalideerd worden , als de data niet voldoet aan de regels dan zal de gebruiker geredirect worden naar dezelfde stap maar deze keer zullen er error messages meegestuurd worden zodat de gebruiker

bovenaan de pagina een message te zien krijgt over oncorrect ingevulde data. Als de data wel correct gevalideerd is dan zal deze opgeslagen worden en zal de user geredirect worden naar de volgende stap.

- **downloadWorkProcess(\$id)** bevat code om een file te downloaden die geupload is door een user.
- **Redirect** redirect gebruikers die inloggen naar de laatst ingevulde stap van de registratie.
- **registerStep(\$stepId, \$user)** slaagt op in de database welke stap de laatst ingevulde stap is van de gebruiker.
- **sendSupportMail(Request \$request)** als een user gebruik maakt van het support formulier (om een vraag te stellen) zal deze functie een mail sturen naar pitchpoint met de desbetreffende vraag.

We hebben ook nog aparte controllers voor de resource routes (employees, awards, references, clients, comments, strongpoints). In employees en clients zijn er functies voor het aanmaken, updaten, verwijderen, en editeren. Bij awards, references, comments, strongpoints kan er enkel aangemaakt en verwijderd worden.

Validation

Zoals hierboven vermeld zal de controller de binnenkomende data valideren aan de hand van een aantal regels. Hoe werkt dit nu juist? Wel, de eerste stap is het definiëren van de validatie regels. Laravel heeft enkele out of the box validatie regels die men kan gebruiken. Hieronder een kort lijstje van de validatie regels die wij gebruiken in het project.

- **required** Deze field moet worden ingevuld.
- **min:#** Deze field moet min x aantal characters hebben.
- **max:#** Deze field mag max x aantal characters hebben.
- **email** Deze field moet een geldig email adress bevatten.
- **numeric** Deze field mag enkel numerieke characters bevatten.
- **unique** Deze field moet uniek zijn in een bepaalde database table en kolom. dus bv: email adress in de users table.
- **between:#,#** Deze field moet tussen de 2 ingevulde waardes liggen.
- **integer** Deze field moet een integer bevatten.
- **confirmed** Als deze field validatie regel op bv: field password word toegepast dan zal er eenzelfde waarde in password_confirmation field moeten staan.

Deze regels worden allemaal opgeslagen in een array die mee word gegeven aan de validate methode. De validate methode accepteert 2 parameters, de http request met alle ingevoerde data en de array met de validatie regels. Als de validatie gelukt is zal de controller verder de functie uit voeren, maar als de validatie mislukt zal de controller de user terugsturen naar de huidige pagina maar deze keer worden de error messages meegegeven. Deze zullen worden weergegeven in de view, ook zullen de fields die niet correct gevalideerd zijn een rode border krijgen zodat de user makkelijk kan zien welke fields niet correct zijn ingevuld.

We hebben 1 probleem gehad met de validatie, voor enkele textarea's hadden we een validatieregel voor max aantal characters en tegelijk hadden we wat javascript code die toonde hoeveel characters de user nog kan invoeren. Maar bij de validatie van laravel telde een enter als meerdere characters en bij onze js validatie voor 1 char. dus hebben we de laravel validatie regel verwijderd. Dit is niet super ideaal want als bij de browser javascript staat uitgeschakeld zal men meer characters kunnen invoeren als toegelaten. Maar anders zou men bijna geen enters kunnen gebruiken in textarea's omdat deze tellen voor meerdere characters bij laravel validatie.

Views

Wij hebben in onze views gebruik gemaakt van de built-in templating engine genaamd blade. 2 grote voordelen die we hiermee hebben zijn het gebruik van template overerving en het gebruik van secties. Wij hebben voor onze applicatie een aantal standaard layouts gemaakt waarvan andere pagina's zullen overerven.

- **Master layout** Elke pagina erft hiervan over => deze pagina bevat meta properties, link naar css file en link naar js file.
- **Admin layout** Bevat de standaard layout van het admin gedeelte van de applicatie.

- **Simple layout** Bevat de standaard layout die gebruikt word bij het editeer scherm van Employees en Clients en bij de support en legal agreement pagina.
- **step layout** bevat de standaard layout voor de verschillende stappen van het registratieproces.

In deze layout files hebben we sections gecreeërd aan de hand van `@yield` directives. Deze kunnen we in de child pages (pages die overerven) vullen aan de hand van `@section` directives.

data weergeven

Telkens er een pagina word geladen door de controller zal de controller data meesturen die weergegeven moet worden in de view. Deze data sturen we mee in de vorm van een multidimensionele array genaamd de viewbag. Deze viewbag word uitgelezen in de view zodat alle data op de juiste plek komt te staan. Data weergeven in de view word gedaan door curly braces te plaatsen. Als we nu bv alle employees willen weergeven zullen we met een foreach loop door de array met index employees lopen.

```
@foreach($viewbag['employees'] as $employee)
    @include('includes.previews.employee')
@endforeach
```

Dit wil zeggen dat voor elke employee in de viewbag men de partial employee moet laden en in deze partial laden we de specifieke data in voor elke employee. De naam geven we bv weer op deze manier: `geef naam weer NIET VERGETE EH !!!!` Omdat de content die weergegeven moet worden is ingevoerd door een user gebruiken we double curly braces zodat de HTML entiteiten geëscaped worden door de engine. Anders zou er aan cross-site scripting kunnen gedaan worden. We hebben ook een rich text editing functie moeten toevoegen waarbij we de content die weergegeven werd niet mocht geëscaped worden omdat de opmaak zichtbaar moest zijn. Hiervoor konden we dus niet de templating engine gebruiken omdat deze alle tags escaped, dus om dit op te lossen maken we gebruik van een package genaamd purifier. Deze package zal alle html tags die toegelaten worden in de config file (is dus instelbaar) niet escaperen, alle andere tags worden geëscaped. Omdat dit op de serverside niet goed werkte (cleanen voordat opgeslagen word) word de content nog eens gecleaned in de view.

De statische content van de pagina renderen we door het gebruik van language files. Op deze manier is het heel makkelijk om meerdere talen toe te voegen aan de applicatie, momenteel was dit nog niet noodzakelijk maar als het toch ooit eens geïmplementeerd moet worden is dit niet zoveel extra werk. Een language file bestaat uit een return met een array in met de statische content. voorbeeld van lang file:

```
return [

    'edit' => [
        'title' => 'Edit Client',

        'b1' => 'Save',
        'b2' => 'Cancel',

        'b3' => 'Upload logo',
        'b3a' => 'Select logo',

        'e1' => 'Only .jpg, .jpeg, .gif and .png formats are supported.',
    ],
];
```

De titel kan men dan weergeven in de view op deze manier: `@lang('client.edit.title')`

Data opslaan

Data opslaan doen we aan de hand van forms. Om forms te gebruiken in het laravel framework moeten we een dependency toevoegen via composer omdat dit niet meer in de core van het framework zit sinds versie 5.0. Omdat heel het registratieproces geëditeerd moet kunnen worden hebben we gebruik gemaakt van een model form. Een model form word geopend op deze manier: `{!! Form::model($agency, array('files' => true, 'id'=>'new__agency', 'class'=>'form--top-page')) !!}` Als hierna een form element word gebruikt bv een text input veld voor een woonplaats, dan zal de naam van dit input veld hetzelfde moeten zijn als de attribuutnaam in het bijhorende model. Als men form elementen op deze manier benaamt dan zal tijdens het renderen de form gevuld worden met data uit de database. Maar als er nu bijvoorbeeld een form word gesubmit en er zijn enkele validatie fouten dan zal er in de sessie data een item zijn met de zelfde naam als het input veld. Als dit het geval is dan zal de form gevuld worden met de data uit de sessie. Zodat form elementen terug gefilled worden met de data die gevalideerd werd en niet de data uit de db.

Javascript files

In dit project maken we gebruik van Browserify. Dit helpt ons om beter onderhoudbare code te schrijven door het opsplitsen van verschillende javascript files die uiteindelijk gecompileerd worden naar 1 main javascript bestand. Dit laat ons toe om op een modulaire manier code te schrijven. Hoe gebeurt dit nu juist? We hebben een main.js bestand in de root van de javascript folder, hierin requiren we andere javascript files. Als we nu gulp runnen dan zal browserify al deze javascript files compileren in 1 groot javascript bestand. Welke verschillende javascript files hebben we allemaal opgesplitst?

- **awards.js** animatie voor het weergeven van het awards form voor een award toe te voegen.
- **bootstrap-dropdown.js** code voor de multiselect die gebruikt word voor het selecteren van services voor klanten van een agency.
- **checkUrl.js** word gebruikt voor het formateren van een url input.
- **clients.js** animatie voor het weergeven van het clients form om een client toe te voegen + het toevoegen van een has-errors klasse om een red border weer te geven bij validatie fout, hier moest dit gedaan worden met javascript omdat dit niet werkte bij de multiselect op de gewone manier (via de templating engine).
- **dashboard.js** enkele animaties voor het tonen/hiden van buttons + een check-all functie voor de search filters in het dashboard.
- **disableButton.js** word gebruikt om buttons op een disabled state te zetten.
- **employee.js** animatie voor het weergeven van het employee form voor een employee toe te voegen.
- **enableEditor.js** Het enablen van een rich text editing functie op sommige textarea fields.
- **expertises.js** bevat de logica om expertises te selecteren met een bepaalde level, eenmaal deze is geselecteerd kan er een percentage aan deze expertise worden meegegeven. In deze js file staat ook een teller die controleerd of het totaal van de geselecteerde percentages gelijk is aan 100%. Als dit het geval is zal de save button actief worden.
- **fallbackDatepicker.js** Fallback voor elke browser die geen html datepicker ondersteunt. Deze js file zal detecteren of de browser het datepicker element ondersteunt, zoniet dan zal deze een jquery ui datepicker laden.
- **imagePreview.js** Zorgt ervoor dat als de user een image selecteerd dat deze instant gepreviewd word op de pagina.
- **jquery-ui.min.js** jquery-ui file voor datepicker functie.
- **limitInput.js** word gebruikt om inputs te limiteren zodat er op een bepaalde char count geen chars meer kunnen worden ingegeven + de gebruiker zal bij het typen van de laatste 10 chars een counter te zien krijgen met de resterende chars die nog kunnen getyped worden.
- **markets.js** bevat de logica om percentages te tellen van verschillende markets. als het totaal gelijk is aan 100% dan zal de save button actief worden.
- **mobile-nav.js** animatie en weergave van de navbar als deze mobiel word weergegeven.
- **modernizr-custom.js** Tool die gebruikt word om te checken of een bepaald html element ondersteunt word door de browser, een custom versie zodat het js bestand zo klein mogelijk blijft.

- **numericInput.js** zorgt ervoor dat op sommige velden enkel numerieke characters kunnen worden ingegeven. Zorgt ook voor het formateren van telefoon nummers en geld bedragen.
- **print.js** zorgt ervoor dat de admin kan aanklikken wat er mee moet afgeprint worden en wat niet.
- **references.js** animatie voor het weergeven van het references form om referenties toe te voegen.
- **RichTextEditor.js** bevat de Rich text editor.
- **saveWarning.js** zorgt ervoor dat als men form aanpassingen heeft gedaan en als men ergens naar wilt navigeren dat de user een melding krijgt die zegt dat niet alles is opgeslagen met dan als keuze om door te gaan of op deze pagina te blijven.
- **smoothScroll.js** zorgt voor een smooth scroll effect.
- **vat.js** formateren van het vat veld.

Elk Web-platform heeft enkele basis functies die altijd geïmplementeerd moeten worden , een web-framework zoals laravel maakt het mogelijk om dit niet elke keer opnieuw te coderen (en dus niet elke keer opnieuw het wiel te moeten uitvinden.

enkele features die later zeker nog besproken worden:


- Bundles (Composer as dependency manager)
- Eloquent ORM (object-relational mapping) vs query builder
- application logic
- sql database
- models
- artisan
- views
- restful controllers
- blade templating engine
- migrations en seeders voor de databank
- unit testing
- pagination
- form
- Filesystems
- Laravel Elixir

Bespreking ontwikkeling

Model

Voor dat de seeders uitgevoerd kunnen worden moet het model aangemaakt worden. Models worden aangemaakt door dit command : `php artisan make:model modelname` . Het aangemaakt model zal geplaatst worden in de map `App` . In dit model worden er een aantal accessors gedefinieerd zoals :

- welke fields er mass-assignable moeten zijn
 - Als een field als fillable word gedefinieerd dan zal deze mass-assignable zijn. Dit wil zeggen dat als er een input array word verstuurd als de gebruiker zijn form submit. De fields die in deze array zitten en fillable zijn zullen in 1 keer kunnen worden opgeslagen. En niet allemaal apart , als deze fields niet worden aangeduid als fillable dan zou alles mass assignable zijn en dan kan men iets meesturen met de input dat normaal niet verandert zou mogen worden bv: `is_admin=true`
- welke fields er gecast moeten kunnen worden
 - Booleans worden in de databank opgeslagen als TINYINT (=0 of 1) maar als we deze waardes uit de databank halen , is het handig dat we verder met het boolean type kunnen werken , dit is mogelijk door in het model deze fields te definiëren en te zeggen naar welke data-type deze gecast moeten worden.
- welke fields hidden zijn.
 - als er bv. users uit de databank moeten gehaald worden dan zal het password meegegeven worden maar als er een hidden attribute password word gedefinieerd dan zal dit niet meer gebeuren.

Laravel maakt gebruik van een ORM(object-relational-mapper) genaamd Eloquent. Eloquent voorziet een object georiënteerde query taal zodat de ontwikkelaar zich kan concentreren op het object model in plaats van de database structuur. Eloquent zal deze taal vertalen in de juiste syntax voor de database. Elke Table heeft een bijhorend model dat gebruikt word om data uit de database te halen en toe te voegen. Om models aan te maken maakt men gebruik van de command line tool genaamd artisan , deze tool zit ingebakken in het framework. Door het command `php artisan make:model User` uit te voeren zal er een model genaamd User worden aangemaakt. Als er nergens expliciet word gedefinieerd welke table er gebruikt moet worden voor het model zal eloquent het meervoud van de modelnaam + enkel kleine letters uit de db gebruiken als table. bv. model User zal standaard de users table uit de db gebruiken. In deze models zullen ook de relaties tussen elkaar gedeclareerd moeten worden want de User zal bv geassocieerd worden met een agency en een agency zal dan weer geassocieerd worden met enkele klanten. Zo zijn er wel meerdere relaties die gedefinieerd moesten worden , hieronder een overzicht van ons databank schema waar je makkelijk kan zien welke relaties er bestaan in onze applicatie.  Om deze relaties te definiëren zal men eerst moeten bepalen welk type relatie er gelegd moet worden. Zo bestaan er een aantal types (die wij gebruikt hebben):

1. One To One relationship Bv: one Agency has one User account => om deze relatie te definiëren moeten we in het Agency model een methode user plaatsen met hierin de functie :

```
public function user()
{
    return $this->hasOne('App\User');
}
```

2. One To Many relationship Bv: one Agency has many Employees om deze relatie te definiëren moeten we in het Agency model een methode user plaatsen met hierin de functie :

```
public function employees()
{
    return $this->hasMany('App\Employee', 'agency_id')->get();
}
```

De relatie moet ook andersom gedefinieerd worden zodat we bv de agency kunnen achterhalen aan de hand van een employee. Dit word gedaan aan de hand van deze methode toe te voegen.

```
public function agency()
{
    return $this->belongsTo('App\Agency');
}
```

Nu kunnen we beginnen met de controllers te gebruiken zodat we de juiste data in de view kunnen krijgen , maar eerst wat info over hoe de gebruiker navigeert tussen verschillende pagina's.

Routes

De gebruiker kan naar verschillende pagina's navigeren met behulp van routes. Deze routes dienen zelf gedefinieerd te worden in de file `app/Http/routes.php`. Als er nu een request word gedaan door de client en deze is gelijk aan de gespecificeerde route uri, dan zal de server weten welke controller er moet worden aangesproken om een bepaalde functie uit te voeren en/of de juiste pagina terugsturen naar de client. Bv: `Route::get('/{id}/step1', 'AgencyController@showStep1');` deze route verteld ons dat als je naar `/2/step1` surft dan zal de `id=2` meegegeven worden als parameter in de `showStep1` functie in de `AgencyController` en dan zal de eerste stap terug gestuurd worden.

resourceful routes

We hebben ook gebruik gemaakt van resourceful routes , dit zijn routes die geregistreerd worden voor een resource controller. Als men een controller aanmaakt dat bv requests verwerkt aangaande werknemers kan men gebruik maken van resource controllers. Deze kunnen aangemaakt worden via artisan door het command : `php artisan make:controller EmployeeController --resource` uit te voeren. Hierna zal er een resource route worden gedefinieerd die geassocieerd word met deze controller. Deze resource route zal meerdere routes bevatten die we nodig hebben om data op te slaan , data te updaten , data te verwijderen en data weer te geven. Deze routes zullen naar methodes linken in de controller. Al deze methodes worden ook al gegenereerd bij het aanmaken van de controller. Zo hebben we enkele resourceful routes gecreeërd samen met hun bijhorende controller:

- voor werknemers van een agency
- voor awards van een agency
- voor referenties van een agency
- voor klanten van een agency
- voor commentaar over een agency
- voor sterke punten van een agency

HTTP Middleware

HTTP Middleware gebruiken we als filter om op bepaald requests een gepaste response te geven. Bv. middleware kan controleren of de persoon die de request stuurt geauthenticeerd is of niet. Als deze persoon ingelogd is dan zal hij naar register step 1 worden gestuurd en anders naar de login page. Welke middleware gebruiken wij in onze applicatie ?

- **web group middleware** Word gebruikt voor cookies , sessies en Cross site request forgery protection. Deze middleware word toegepast op elke route in onze applicatie omdat overal sessies en cookies en CSRF protection nodig zijn deze middleware zit ingebakken in het framework.
- **Auth middleware** Word gebruikt bij routes die enkel toegankelijk zijn voor geauthenticeerde users. Als user niet geauthenticeerd is dan zal deze een pagina krijgen met een 'unauthorized' message. Deze middleware komt ook out of the box bij het laravel framework.

- **Admin middleware** Word gebruikt bij routes die enkel toegankelijk zijn voor geauthenticeerde users met admin rechten. Deze middleware hebben we zelf geschreven.

Middleware kan aangemaakt worden via artisan door het command `php artisan make:middleware Middlename` uit te voeren. Om deze middleware te gebruiken dienen we te declareren op welke routes deze middleware moet uitgevoerd worden. Dit gebeurt in de route.php file waar alle routes staan gedefinieerd. Men kan ook groepen maken van routes waar deze middleware moet worden uitgevoerd. Elke keer als er nu een request verstuurd wordt naar een bepaalde route met middleware zal eerst de code van de middleware worden uitgevoerd voordat de controller aangesproken zal worden.

Controllers

Controllers worden gebruikt om gerelateerde http requests te verwerken in een klasse. Zo kunnen we, zoals bij routes vermeld, een route aanmaken die deze controller actie zal triggeren. Wij hebben 2 "hoofd" controllers :

Admin controller

In de admin controller worden (bijna) alle requests verwerkt die uit het admin gedeelte van de webapplicatie komen. Als er wordt ingelogd met een admin account zal de `index()` functie uitgevoerd worden.

- **index()** Deze functie zal data uit de databank halen om een overzicht te kunnen weergeven van alle geregistreerde agencies en de user sturen naar de dashboard pagina.
- **show(\$id)** als er nu op een agency wordt geklikt in het dashboard dan zal deze functie worden aangesproken in de controller en dan zal de controller een nieuwe view terug sturen namelijk het overzicht van het geklikte agency met alle data die nodig is om dit weer te geven. Het `$id` vertelt de controller op welke agency er is geklikt.
- **store(\$id , Request \$request)** In het agency overzicht zal de admin commentaar, sterke punten en een laatste contactdatum kunnen toevoegen. Dit opslaan naar de databank zal gebeuren door deze functie. In de request parameter zit de data die opgeslagen moet worden en het `$id` vertelt de controller naar welk agency er moet worden opgeslagen.

Agency controller Deze controller zal zich bezig houden met het verwerken van requests aangaande het registratie proces.

- **showStep1(\$id)** deze functie zal uitgevoerd worden wanneer een user op de registreer knop drukt. De controller zal hierna de view van de eerste registratiestap terug sturen naar de client.
- **show(\$id , \$stepId)** Deze functie zal alle verschillende stappen van de registratie weergeven met de correcte data , want alle data die al ingevuld is geweest moet zich terug vullen in het form zodat geëditeerd kan worden. Als parameters gebruiken we `$id` en `$stepId` , `$id` gebruiken we voor de correcte data van het juiste agency terug te sturen en `$stepId` gebruiken we zodat de controller weet welke stap er moet weergegeven worden. telkens dat deze functie wordt uitgevoerd zal deze ook updaten tot welke stap van het registratieproces de gebruiker alles al heeft ingevuld , dit wordt allemaal opgeslagen in de database. Deze waarde wordt gebruikt om een afbakening te maken tot waar de gebruiker allemaal kan navigeren : de gebruiker kan navigeren tot de stap na de laatst ingevulde stap , zodat er geen stappen kunnen overgeslagen worden. Deze functie zal ook een mail sturen naar de gebruiker als deze de registratie volledig heeft doorlopen. Deze mail bevat een bedankje voor de registratie en een link waar het account kan geëditeerd worden. Ook zal er een mail gestuurd worden naar pitchpoint zelf als een nieuw agency zich heeft geregistreerd , deze mail bevat de naam van het geregistreerde agency en een link naar de admin pagina van pitchpoint.
- **store(\$id , \$stepId , Request \$request)** Deze functie zal alle ingevoerde data van het registratie proces verwerken en opslaan in de databank. Telkens er een submit wordt gedaan in het registratieproces zal deze functie checken op welke stap er juist gesubmit is , hierna zal hij de juiste validatie regels in de validator steken. De data zal gevalideerd worden , als de data niet voldoet aan de regels dan zal de gebruiker geredirect worden naar dezelfde stap maar deze keer zullen er error messages meegestuurd worden zodat de gebruiker

bovenaan de pagina een message te zien krijgt over oncorrect ingevulde data. Als de data wel correct gevalideerd is dan zal deze opgeslagen worden en zal de user geredirect worden naar de volgende stap.

- **downloadWorkProcess(\$id)** bevat code om een file te downloaden die geupload is door een user.
- **Redirect** redirect gebruikers die inloggen naar de laatst ingevulde stap van de registratie.
- **registerStep(\$stepId, \$user)** slaagt op in de database welke stap de laatst ingevulde stap is van de gebruiker.
- **sendSupportMail(Request \$request)** als een user gebruik maakt van het support formulier (om een vraag te stellen) zal deze functie een mail sturen naar pitchpoint met de desbetreffende vraag.

We hebben ook nog aparte controllers voor de resource routes (employees, awards, references, clients, comments, strongpoints). In employees en clients zijn er functies voor het aanmaken, updaten, verwijderen, en editeren. Bij awards, references, comments, strongpoints kan er enkel aangemaakt en verwijderd worden.

Validation

Zoals hierboven vermeld zal de controller de binnenkomende data valideren aan de hand van een aantal regels. Hoe werkt dit nu juist? Wel, de eerste stap is het definiëren van de validatie regels. Laravel heeft enkele out of the box validatie regels die men kan gebruiken. Hieronder een kort lijstje van de validatie regels die wij gebruiken in het project.

- **required** Deze field moet worden ingevuld.
- **min:#** Deze field moet min x aantal characters hebben.
- **max:#** Deze field mag max x aantal characters hebben.
- **email** Deze field moet een geldig email adress bevatten.
- **numeric** Deze field mag enkel numerieke characters bevatten.
- **unique** Deze field moet uniek zijn in een bepaalde database table en kolom. dus bv: email adress in de users table.
- **between:#,#** Deze field moet tussen de 2 ingevulde waardes liggen.
- **integer** Deze field moet een integer bevatten.
- **confirmed** Als deze field validatie regel op bv: field password word toegepast dan zal er eenzelfde waarde in password_confirmation field moeten staan.

Deze regels worden allemaal opgeslagen in een array die mee word gegeven aan de validate methode. De validate methode accepteert 2 parameters, de http request met alle ingevoerde data en de array met de validatie regels. Als de validatie gelukt is zal de controller verder de functie uit voeren, maar als de validatie mislukt zal de controller de user terugsturen naar de huidige pagina maar deze keer worden de error messages meegegeven. Deze zullen worden weergegeven in de view, ook zullen de fields die niet correct gevalideerd zijn een rode border krijgen zodat de user makkelijk kan zien welke fields niet correct zijn ingevuld.

We hebben 1 probleem gehad met de validatie, voor enkele textarea's hadden we een validatieregel voor max aantal characters en tegelijk hadden we wat javascript code die toonde hoeveel characters de user nog kan invoeren. Maar bij de validatie van laravel telde een enter als meerdere characters en bij onze js validatie voor 1 char. dus hebben we de laravel validatie regel verwijderd. Dit is niet super ideaal want als bij de browser javascript staat uitgeschakeld zal men meer characters kunnen invoeren als toegelaten. Maar anders zou men bijna geen enters kunnen gebruiken in textarea's omdat deze tellen voor meerdere characters bij laravel validatie.

Views

Wij hebben in onze views gebruik gemaakt van de built-in templating engine genaamd blade. 2 grote voordelen die we hiermee hebben zijn het gebruik van template overerving en het gebruik van secties. Wij hebben voor onze applicatie een aantal standaard layouts gemaakt waarvan andere pagina's zullen overerven.

- **Master layout** Elke pagina erft hiervan over => deze pagina bevat meta properties, link naar css file en link naar js file.
- **Admin layout** Bevat de standaard layout van het admin gedeelte van de applicatie.

- **Simple layout** Bevat de standaard layout die gebruikt word bij het editeer scherm van Employees en Clients en bij de support en legal agreement pagina.
- **step layout** bevat de standaard layout voor de verschillende stappen van het registratieproces.

In deze layout files hebben we sections gecreeërd aan de hand van `@yield` directives. Deze kunnen we in de child pages (pages die overerven) vullen aan de hand van `@section` directives.

data weergeven

Telkens er een pagina word geladen door de controller zal de controller data meesturen die weergegeven moet worden in de view. Deze data sturen we mee in de vorm van een multidimensionele array genaamd de viewbag. Deze viewbag word uitgelezen in de view zodat alle data op de juiste plek komt te staan. Data weergeven in de view word gedaan door curly braces te plaatsen. Als we nu bv alle employees willen weergeven zullen we met een foreach loop door de array met index employees lopen.

```
@foreach($viewbag['employees'] as $employee)
    @include('includes.previews.employee')
@endforeach
```

Dit wil zeggen dat voor elke employee in de viewbag men de partial employee moet laden en in deze partial laden we de specifieke data in voor elke employee. De naam geven we bv weer op deze manier: `geef naam weer NIET VERGETE EH !!!!` Omdat de content die weergegeven moet worden is ingevoerd door een user gebruiken we double curly braces zodat de HTML entiteiten geëscaped worden door de engine. Anders zou er aan cross-site scripting kunnen gedaan worden. We hebben ook een rich text editing functie moeten toevoegen waarbij we de content die weergegeven werd niet mocht geëscaped worden omdat de opmaak zichtbaar moest zijn. Hiervoor konden we dus niet de templating engine gebruiken omdat deze alle tags escaped, dus om dit op te lossen maken we gebruik van een package genaamd purifier. Deze package zal alle html tags die toegelaten worden in de config file (is dus instelbaar) niet escaperen, alle andere tags worden geëscaped. Omdat dit op de serverside niet goed werkte (cleanen voordat opgeslagen word) word de content nog eens gecleaned in de view.

De statische content van de pagina renderen we door het gebruik van language files. Op deze manier is het heel makkelijk om meerdere talen toe te voegen aan de applicatie, momenteel was dit nog niet noodzakelijk maar als het toch ooit eens geïmplementeerd moet worden is dit niet zoveel extra werk. Een language file bestaat uit een return met een array in met de statische content. voorbeeld van lang file:

```
return [

    'edit' => [
        'title' => 'Edit Client',

        'b1' => 'Save',
        'b2' => 'Cancel',

        'b3' => 'Upload logo',
        'b3a' => 'Select logo',

        'e1' => 'Only .jpg, .jpeg, .gif and .png formats are supported.',
    ],
];
```

De titel kan men dan weergeven in de view op deze manier: `@lang('client.edit.title')`

Data opslaan

Data opslaan doen we aan de hand van forms. Om forms te gebruiken in het laravel framework moeten we een dependency toevoegen via composer omdat dit niet meer in de core van het framework zit sinds versie 5.0. Omdat heel het registratieproces geëditeerd moet kunnen worden hebben we gebruik gemaakt van een model form. Een model form word geopend op deze manier: `{!! Form::model($agency, array('files' => true, 'id'=>'new__agency', 'class'=>'form--top-page')) !!}` Als hierna een form element word gebruikt bv een text input veld voor een woonplaats, dan zal de naam van dit input veld hetzelfde moeten zijn als de attribuutnaam in het bijhorende model. Als men form elementen op deze manier benaamt dan zal tijdens het renderen de form gevuld worden met data uit de database. Maar als er nu bijvoorbeeld een form word gesubmit en er zijn enkele validatie fouten dan zal er in de sessie data een item zijn met de zelfde naam als het input veld. Als dit het geval is dan zal de form gevuld worden met de data uit de sessie. Zodat form elementen terug gefilled worden met de data die gevalideerd werd en niet de data uit de db.

Javascript files

In dit project maken we gebruik van Browserify. Dit helpt ons om beter onderhoudbare code te schrijven door het opsplitsen van verschillende javascript files die uiteindelijk gecompileerd worden naar 1 main javascript bestand. Dit laat ons toe om op een modulaire manier code te schrijven. Hoe gebeurt dit nu juist? We hebben een main.js bestand in de root van de javascript folder, hierin requiren we andere javascript files. Als we nu gulp runnen dan zal browserify al deze javascript files compileren in 1 groot javascript bestand. Welke verschillende javascript files hebben we allemaal opgesplitst?

- **awards.js** animatie voor het weergeven van het awards form voor een award toe te voegen.
- **bootstrap-dropdown.js** code voor de multiselect die gebruikt word voor het selecteren van services voor klanten van een agency.
- **checkUrl.js** word gebruikt voor het formateren van een url input.
- **clients.js** animatie voor het weergeven van het clients form om een client toe te voegen + het toevoegen van een has-errors klasse om een red border weer te geven bij validatie fout, hier moest dit gedaan worden met javascript omdat dit niet werkte bij de multiselect op de gewone manier (via de templating engine).
- **dashboard.js** enkele animaties voor het tonen/hiden van buttons + een check-all functie voor de search filters in het dashboard.
- **disableButton.js** word gebruikt om buttons op een disabled state te zetten.
- **employee.js** animatie voor het weergeven van het employee form voor een employee toe te voegen.
- **enableEditor.js** Het enablen van een rich text editing functie op sommige textarea fields.
- **expertises.js** bevat de logica om expertises te selecteren met een bepaalde level, eenmaal deze is geselecteerd kan er een percentage aan deze expertise worden meegegeven. In deze js file staat ook een teller die controleert of het totaal van de geselecteerde percentages gelijk is aan 100%. Als dit het geval is zal de save button actief worden.
- **fallbackDatepicker.js** Fallback voor elke browser die geen html datepicker ondersteunt. Deze js file zal detecteren of de browser het datepicker element ondersteunt, zoniet dan zal deze een jquery ui datepicker laden.
- **imagePreview.js** Zorgt ervoor dat als de user een image selecteerd dat deze instant gepreviewd word op de pagina.
- **jquery-ui.min.js** jquery-ui file voor datepicker functie.
- **limitInput.js** word gebruikt om inputs te limiteren zodat er op een bepaalde char count geen chars meer kunnen worden ingegeven + de gebruiker zal bij het typen van de laatste 10 chars een counter te zien krijgen met de resterende chars die nog kunnen getyped worden.
- **markets.js** bevat de logica om percentages te tellen van verschillende markets. als het totaal gelijk is aan 100% dan zal de save button actief worden.
- **mobile-nav.js** animatie en weergave van de navbar als deze mobiel word weergegeven.
- **modernizr-custom.js** Tool die gebruikt word om te checken of een bepaald html element ondersteunt word door de browser, een custom versie zodat het js bestand zo klein mogelijk blijft.

- **numericInput.js** zorgt ervoor dat op sommige velden enkel numerieke characters kunnen worden ingegeven. Zorgt ook voor het formateren van telefoon nummers en geld bedragen.
- **print.js** zorgt ervoor dat de admin kan aanklikken wat er mee moet afgeprint worden en wat niet.
- **references.js** animatie voor het weergeven van het references form om referenties toe te voegen.
- **RichTextEditor.js** bevat de Rich text editor.
- **saveWarning.js** zorgt ervoor dat als men form aanpassingen heeft gedaan en als men ergens naar wilt navigeren dat de user een melding krijgt die zegt dat niet alles is opgeslagen met dan als keuze om door te gaan of op deze pagina te blijven.
- **smoothScroll.js** zorgt voor een smooth scroll effect.
- **vat.js** formateren van het vat veld.

Elk Web-platform heeft enkele basis functies die altijd geïmplementeerd moeten worden , een web-framework zoals laravel maakt het mogelijk om dit niet elke keer opnieuw te coderen (en dus niet elke keer opnieuw het wiel te moeten uitvinden.

enkele features die later zeker nog besproken worden:

- Bundles (Composer as dependency manager)
- Eloquent ORM (object-relational mapping) vs query builder
- application logic
- sql database
- models
- artisan
- views
- restful controllers
- blade templating engine
- migrations en seeders voor de databank
- unit testing
- pagination
- form
- Filesystems
- Laravel Elixir

Conclusie en samenvatting

Uitbreidingen/future work

Html mail

De eerste week bij prophets moest ik met html/css een mail maken voor Carrefour, die deze dan doorstuurt naar zijn klanten (reclame dus). Er werd mij verteld dat deze mail puur oefening was zodat ik er wat kon inkomen.

Ik kreeg bij deze opdracht begeleiding van Elien, ze stuurde me een html template waar de structuur al grootendeels gelegd was. De structuur bestond uit tables , tables en nog eens tables. Waarom niet op de traditionele manier van websites maken ? Het verschil tussen het opstellen van een mail en een website zit in het aantal mailclients en internetbrowsers. Er zijn heel veel mailclients tegenover internetbrowsers en al deze mailclients ondersteunen een verschillende subset van html en css . Bovendien gebruiken desktop, webmail en mobiele email clients allemaal verschillende rendering engines om de content van de mail te displayen. Om dit te illustreren kunt u een voorbeeld hieronder zien van enkele mail clients en welke css properties deze ondersteunt.

font-family	✓	✓	✓	✓	✓	✓	✓	✓	✓
font-size	✓	✓	✓	✓	✓	✓	✓	✓	✓
font-style	✓	✓	✓	✓	✓	✓	✓	✓	✓
font-variant	✓	✓	✓	✓	✓	✓	✓	✓	✗
font-weight	✓	✓	✓	✓	✓	✓	✓	✓	✓
height	✗	✓	✓	✓	✓	✓	✓	✓	✗
left	✗	✓	✓	✓	✓	✓	✓	✓	✗
letter-spacing	✓	✓	✓	✓	✓	✓	✓	✓	✗
line-height	✓	✓	✓	✓	✓	✓	✓	✓	✗
list-style-image	✗	✓	✓	✓	✓	✓	✓	✓	✗
list-style-position	✗	✓	✓	✓	✓	✓	✓	✓	✗
list-style-type	✗	✓	✓	✓	✓	✓	✓	✓	✓
margin	✓	✓	✓	✓	✓	✗	✓	✓	✗
opacity	✗	✗	✓	✓	✓	✓	✓	✓	✗
overflow	✗	✓	✓	✓	✓	✓	✓	✓	✗
padding	✓	✓	✓	✓	✓	✓	✓	✓	✗
position	✗	✓	✓	✓	✓	✓	✓	✓	✗
right	✗	✓	✓	✓	✓	✓	✓	✓	✗
table-layout	✓	✓	✓	✓	✓	✓	✓	✓	✗
text-align	✓	✓	✓	✓	✓	✓	✓	✓	✓
text-decoration	✓	✓	✓	✓	✓	✓	✓	✓	✓
text-indent	✓	✓	✓	✓	✓	✓	✓	✓	✗
text-transform	✓	✓	✓	✓	✓	✓	✓	✓	✗
top	✗	✓	✓	✓	✓	✓	✓	✓	✗
vertical-align	✗	✓	✓	✓	✓	✓	✓	✓	✗
visibility	✗	✓	✓	✓	✓	✓	✓	✓	✗
white-space	✓	✗	✓	✓	✓	✗	✗	✓	✗
width	✗	✓	✓	✓	✓	✓	✓	✓	✗
word-spacing	✗	✓	✓	✓	✓	✓	✓	✓	✗
z-index	✓	✓	✓	✓	✓	✓	✓	✓	✗
	Outlook '07	Windows Mail	Mac Mail	Entourage 2008	Thunder- bird 2	AOL 9	AOL 10	AOL Mac	Notes 6

Mijn taken:

- Target blank zetten op alle hyperlinks.
- Controleren of er geen lege href="#" meer zijn .
- Alle afbeeldingen voorzien van een ALT tag.
- Title tag invullen.
- Images linken maar links mogen niet absoluut zijn.
- Images slicen uit de voorziene psd.
- Ervoor zorgen dat de images correct weergegeven worden in de mail.
- Testen in Litmus : dit is een tool waarbij je de mail kunt testen in 40 van de populairste email clients.
- Dit alles moest gedaan worden voor zowel de franse als de nederlandse versie.

Uit deze opdracht heb ik geleerd dat zoveel tables moeilijk zijn om mee te werken , ik had een tag ergens vergeten te sluiten met als gevolg dat heel de pagina versprong vanaf een bepaald punt. Het is niet zo evident om een niet gesloten tag te zoeken in een pagina met 1000-2000 van deze tags. Gelukkig zijn er online tools beschikbaar die dit kunnen detecteren. Ik vond het verbazingwekkend dat een 'simpele' mail van carrefour zoveel lijnen code kon bevatten.

website Fukushima

Mijn 2de opdracht was al wat groter als de eerste. Ik moest een website maken voor het nucleaire forum over Fukushima.

Gebruikte technologieën en tools.

Bij het ontwikkelen van deze website heb ik gebruik gemaakt van verscheidene technologieën en tools.

Gulp

Gulp is een build systeem die de ontwikkeling van websites kan verbeteren door het automatiseren van algemene taken zoals : compilen van preprocessed CSS (sass) , het minifyen van JavaScript , reloaden van de browser,...

De installatie gaat als volgt:

- Installeren van Node.js.
- Gulp installeren aan de hand van de package manager van Node.js genaamd npm.

Bij het gebruik van gulp hebben we een src (=source) folder en een build folder nodig. De source folder gebruiken we om alles in te coderen en in de build folder word alle gecompileerde code gezet door middel van gulp zijn streams.

Gulp doet niet veel op zich maar maakt gebruik van plug-ins die men kan installeren om specifieke taken uit te voeren. Deze plugins kan men installeren via de npm package manager. Hierna kan men taken declareren in de file 'gulpfile.js' die in de root van de website folder staat. De plug-ins die men wil gebruiken moeten toegevoegd worden bovenaan deze file , daarna worden de taken gedeclareerd. Een taak word opgebouwd in 3 delen :

1. Source files selecteren waarop taken moeten worden uitgevoerd.
2. Files die geselecteerd zijn worden door een pipe stream gestuurd die alle taken zal uitvoeren die gedeclareerd werden.
3. De aangepaste files zullen geplaatst worden in de build map.

In onze gulpfile hadden we verschillende taken :

- templates
 - Content nemen uit json file en in html file toevoegen zodat we 2 json files kunnen gebruiken één voor de franse en nederlandse versie van de content. Er worden nu 2 index.html files gegenereerd in de build map 1 fr en 1 nl.
 - Alle js files toevoegen aan 1 main.js.
- styles
 - Alle partials bundelen in 1 css file.
 - Sass omzetten naar css.
 - Autoprefixer, produceert code zodat alles ook ondersteunt is door oudere browsers
- watch
 - voert styles en template taak uit als er iets verandert in de projectmap.

Sass

Sass is een extentie van CSS3 , omdat web applicaties en websites uitgebreider worden met de jaren kunnen we gebruiken maken van deze css preprocessor om onze stylesheets onderhoudbaarder en gestructureerder te maken. Waarom ? Omdat Sass enkele nieuwe features aanbied die men momenteel niet kan gebruiken bij

normale CSS , zoals :

- **Variabelen:** Door deze feature kunnen we een aantal variabelen aanmaken die we zullen gebruiken doorheen het maken van de website zoals : kleuren , fonts , margins , line-heights , breakpoints . Het handige aan deze feature is dat als de klant wilt dat bv. een kleur of bv de margins wilt veranderen dan moet dit maar op 1 plaats aangepast worden.
- **Partials** Deze feature staat ons toe om onze stylesheets op te delen in verschillende delen zodat we onze css op een heel modulaire manier konden schrijven. Dit is mogelijk door partials aan te maken die allemaal worden toegevoegd aan een main.scss bestand op deze manier : `@import('filepath')` . De partials worden aangeduid met een `_` voor hun naam zodat sass weet dat dit partials zijn en niet mee moeten gecompileerd worden. Wat er nu wel gecompileerd word is de main.scss file waarin alle partials worden toegevoegd. Zo maakten we voor alles dat logisch gescheiden kon worden van elkaar een partial , dit zorgt voor een overzichtelijker project. Het voordeel van alles te compilen in 1 file is dat het aantal HTTP requests enorm zal dalen tegenover al deze verschillende partials apart.
- **Mixins** Omdat sommige css declaraties veel terugkomen kan men gebruik maken van mixins , dit zijn groepjes van css declaraties die men kan hergebruiken doorheen de website.
- **Extends** Met extend kan je de css declaraties van een klasse extenden naar een andere klasse. bv als je een button maakt dan kan je deze extenden en zo een groene en rode knop klasse maken die de button klasse extenden. Op deze manier moet je veel minder code schrijven.

Susy

Susy is een set van Sass Mixins die dienen om een responsive grid op te stellen. Het leuke hieraan is dat je volledig zelf je grid definieert en dat susy al de nodige berekeningen doet. De sass compiler zal kijken naar de mixin definities in de susy bestanden en zal deze omzetten naar css. De mixin die het meeste word gebruikt is de span mixin :

```
.picblock {  
  
    @include span(4 of 12);  
}
```

deze mixin word gebruikt om de breedte van een kolom te bepalen en word berekend aan de hand van de container mixin (wrapper). Susy staat het toe om een aantal settings in te stellen zodat je het grid helemaal kunt afstellen naar jou wensen , hieronder enkele settings die aan te passen zijn:

```
$susy: (  
  flow: ltr,  
  math: fluid,  
  output: float,  
  gutter-position: after,  
  container: auto,  
  container-position: center,  
  columns: 4,  
  gutters: .25,  
  column-width: false,  
  global-box-sizing: content-box,  
  last-flow: to,  
  debug: (  
    image: hide,  
    color: rgba(#66f, .25),  
    output: background,  
    toggle: top right,  
  ),  
);
```


Tweenmax.js

Er waren enkele DOM elementen die geanimeerd moesten worden. Hierbij hebben we gebruik gemaakt van Tweenmax.js Tweenmax.js is een javascript animatie library die onderdeel is van GSAP (Greensock Animation Platform) , Tweenmax is een uitbreiding van tweenlite.js. Om deze library te gebruiken steken we deze gewoon mee in de package.json file zodat deze mee word geïnstalleerd met het npm-install command , hierna hoeven we deze enkel nog te requiren in de main javascript file en alles is gereed om gebruikt te worden. Waarom GSAP ? Omdat deze library goed gedocumenteerd en dus makkelijk in gebruik is en bovendien zijn de animaties ook heel soepel (hoge framerate) dit kan je makkelijk zien met deze tool [SpeedTest](#)

Hoe werkt dit nu juist ? Een Tweenmax instantie zal 1 of meerdere properties van eender welk DOM object aanpassen over een bepaalde tijdsperiode. Bv. 2 markers die op de map moeten 'miegen'

```
TweenMax.fromTo([".map__pin",".map__pin"],0.7 , {top:"40%", opacity:"0", visibility:"hidden"} , {top:"54%", opacity:"1", visibility:"visible"} );
```

Dit stukje code zal het object aanspreken met het ID map__pin , en de properties :top:40% , opacity:0 en visibility:hidden veranderen naar top:54% , opacity:1 en visibility:visible over de tijd van 0.7 seconden. Zoals je ziet is hier helemaal niet veel code voor nodig , waardoor ik tweenmax.js heel aangenaam vind om mee te werken.

Al de animaties die gebeuren op de website zijn getriggerd op een bepaalde scroll hoogte die word gemeten vanaf de bovenzijde van de pagina.

Verloop

Hier vertel ik hoe het verloop is gegaan bij het maken van deze website , van begin tot einde (wat heb ik eerst gedaan ? en wat daarna ?) + moeilijkheden/problemen

term1

Een term die belangrijk is. Hieronder verschijnen alle pagina's waar deze term te vinden zijn. Volgende term als voorbeeld.

gitbook

Verwijder uiteraard ook deze term in je glossary, maar aanschouw hier toch het resultaat.