Exemple en Laravel 5.5

Laravel 5.5 vient de sortir et pour fêter ça je propose un exemple complet disponible sur Github.

Cet exemple servira de base pour une nouvelle série d'initiation que je compte démarrer bientôt pour cette nouvelle version.

PHP 7 ?

Faut-il passer à PHP 7 (en fait 7.1) ? La question devient pertinente pour les utilisateurs de Laravel parce que la version 5.5 de ce framework imposera cette version de PHP. Alors je vous propose de faire un peu le point de ce que nous apporte ce nouveau PHP…

Déjà on va y gagner en performances puisqu'on nous annonce un gain en 25% et 70%. Bon la marge est grande !

Mais au niveau des fonctionnalités ? Vous pouvez tout trouver dans le manuel.

Pour les allergiques à l'anglais voyons un peu ça en me limitant à ce qui me semble le plus important…

Groupement des « use »

L'utilisation des espaces de noms nous amène à effectuer de nombreuse déclarations. On va pouvoir maintenant les regrouper. prenez par exemple ce code :

```
use App\Http\Controllers\Controller;
use App\Http\Requests\CommentRequest;
use App\Models\Post;
use App\Models\Comment;
```

On a autant de lignes que de déclarations. On va pouvoir simplifier ainsi :

```
use App\Http\{
    Controllers\Controller,
    Requests\CommentRequest
};
use App\Models\{
    Post,
    Comment
};
```

On y gagne en clarté !

Type des paramètres

Paramètres des fonctions

Avec PHP 5 on peut déclarer le type des paramètres pour les interface, les classes et les tableaux. Avec la version 7 on peut enfin aussi déclarer les types scalaires (int, float, string, bool).

Considérez par exemple cette fonction :

```
/**
 * Get the next comments for the specified post.
 *
 * @param  \App\Models\Post  $post
 * @param  integer $page
 * @return array
 */
public function comments(Post $post, $page)
{
    ...
}
```

On a le type du premier paramètre parce que c’est un objet, mais le second, qui est un entier, on ne peut pas le déclarer…

Avec PHP 7 on peut écrire :

```
public function comments(Post $post, int $page)
```

Mais PHP ne va pas vraiment vérifier que le type est bon (allez savoir pourquoi…), il va juste s'efforcer de transformer la valeur pour la rendre conforme au type, s'il y arrive… D'ailleurs il vaut mieux se méfier de la transformation des flottants en entiers qui peuvent réserver des surprises !

Mais on peut lui forcer la main pour qu'il devienne plus pointilleux :

```
declare(strict_types = 1);
```

Obligatoirement sur la première ligne du fichier PHP du code appelant.

Et là il ne laissera rien passer ! Bon, du coup c'est pas mal après d'intercepter l'erreur qui est de type **TypeError**.

Avec la version 7.1 est même apparu un pseudo type : **iterable**. Il arrive souvent qu'on transmettre un paramètre qui implémente l'interface **Transversable**, autrement dit qui acceptent l'utilisation de **foreach**. Dans cette catégorie on a les tableaux et les objets. On peut désormais indiquer à la fonction qu'on va lui transmettre ce genre de chose :

```
function action(iterable $elements)
```

Et rien n'empêche évidement de déclarer une valeur par défaut :

```
function action(iterable $elements = [])
```

Type du retour

La possibilité du typage s'étend à la valeur du retour de la fonction.

Voilà par exemple une fonction qui renvoie un **string** :

```
/**
 * Get .env element.
 *
 * @param string $key
```

```
 * @return string
 */
public function get($key)
{
    return $this->env->get ($key);
}
```

On ne précise pas ici le type de la valeur qu'on renvoie, juste dans les commentaires. Avec PHP 7 on peut écrire :

```
public function get($key) : string
```

Là aussi si on veut un comportement « strict » il faut le préciser de la même manière qu'on a vue ci-dessus.

On peut aussi déclarer le type de retour **iterable**.

Les opérateurs

L'opérateur de coalescence

Je suppose que vous avez souvent l'occasion d'écrire du code dans ce genre :

```
$category = isset($request->category) ? $request->category : 'aucune';
```

PHP 7 nous offre un nouvel opérateur :

```
$category = isset($request->category) ?? 'aucune';
```

Le résultat est le même mais c'est plus concis !

L'opérateur « spaceship »

Prenons ce code qui permet de classer des éléments :

```
usort($element, function ($a, $b) {
  return ($a < $b) ? -1 : (($a > $b) ? 1 : 0);
});
```

On a :

- a < b : -1
- a = b : 0
- a > b : 1

Avec PHP 7 on peut simplifier la syntaxe :

```
usort($element, function ($a, $b) {
  return $a <=> $b;
});
```

Pratique ! Par contre ça me fait pas vraiment penser à un vaisseau spatial mais bon…

Classe anonyme

Avec PHP 5 on a hérité des fonctions anonymes qui sont il faut l'avouer bien pratiques surtout comme fonctions de rappel.

Avec PHP 7 ce sont maintenant les classes qui peuvent devenir anonymes. Autrement dit on n'a pas besoin de faire toutes les déclarations. Voici un exemple :

```
$maClasseAnonyme = new class($parametre) {
   private $parametre;
   public function __construct($parametre) {
     $this->parametre = $parametre;
   }
};
```

A part d'être anonymes on les utilise exactement comme les autres : implémentation d'interfaces, héritages, traits…

J'avoue que je ne vais pas m'en servir tous les jours…

Délégation des générateurs

Si vous n'êtes pas familiarisé avec les générateurs alors commencez par vous renseigner à leur sujet. C'est une façon de fournir des éléments d'un ensemble (itérateur) sans être obligé de construire au départ cet ensemble, donc de ne pas affecter outre

mesure la mémoire.

Ce n’est pas clair ? Voici un exemple :

```
function entiers() {
    for ($i = 0; $i <= 10; ++$i) {
        yield $i;
    }
}

foreach (entiers() as $number) {
    echo "$number ";
}

// Résultat : 0 1 2 3 4 5 6 7 8 9 10
```

Oui c’est un peu idiot comme exemple mais c’est pour le principe !

La délégation c’est le fait pour un générateur d’en appeler un autre :

```
function entiers() {
    for ($i = 0; $i <= 5; ++$i) {
        yield $i;
    }
}

function nombres() {
    yield 100;
    yield from entiers ();
    yield 101;
    yield from entiers ();
}

foreach (nombres() as $number) {
    echo "$number ";
}

// Résultat : 100 0 1 2 3 4 5 101 0 1 2 3 4 5
```

Bon ce n’est pas plus malin comme exemple mais au moins ça donne le principe.

Les exceptions

Les exceptions ont été apportées à PHP à partir de la version 5. Mais ça ne concerne que les objets et toutes les autres erreurs de PHP ne passent pas par là. D'autre part il est en général fait usage d'un gestionnaire par défaut avec la fonction **set_exception_handler**.

PHP 7 apporte l'interface **Throwable** qui concerne les exceptions et, c'est nouveau, les erreurs, parce qu'elle se place en tête de la hiérarchie.

Prenons ce code :

```
function test($object)
{
  return $object->riendutout();
}
test(null);
echo 'Je marche encore !';
```

Si je l'exécute je vais tomber immanquablement sur une erreur fatale : *Call to a member function riendutout() on null…*

On va maintenant intercepter cette erreur :

```
function test($object)
{
  return $object->riendutout();
}
try
{
  test(null);
} catch(Error $e)
{
  echo $e->getMessage(), PHP_EOL;
}
echo 'Je marche encore !';
```

A l'exécution (PHP 7) on obtient :

```
Call to a member function riendutout() on null
Je marche encore !
```

On peut ainsi tout intercepter, même les divisions par zéro !

Syntaxe uniforme des variables

Avec PHP on peut utiliser des variables de variables, même si ce n'est pas très indiqué parce que ça complique la lecture du code et on peut en général facilement s'en passer.

Regardez ce code :

```
$identite = ['nom' => 'Pierre'];
$Pierre = 'Paul';
echo $$identite['nom'];
```

Avec PHP 5 vous obtenez : *Pierre*.

Avec PHP 7 vous obtenez plein d'erreurs : *Array to string conversion…*

Que se passe-t-il ?

PHP 7 apporte la « syntaxe uniforme des variables », un truc qui couvait depuis quelques années avec pas mal d'opposition à cause du cassage du code antérieur. Mais finalement c'est passé dans cette version.

Normalement l'interprétation se fait de gauche à droite, donc on devrait commencer par **$$identite**, puis ensuite prendre l'offset. Mais PHP 5 ne fait pas ça, il commence par traiter **$identite['nom']**. Par contre comme PHP 7 va scrupuleusement de gauche à droite il y a un gros souci. Pour que ça fonctionne on doit écrire **${$identite['nom']}**. Et là maintenant ça fonctionne !

Ce changement n'est pas anodin si vous voulez migrer du code en PHP 7 !

Conclusion

On voit qu'il y a pas mal de nouveautés dans cette version, et je n'ai pas parlé des choses plus ou moins anodines que vous pouvez

trouver dans le manuel…

---

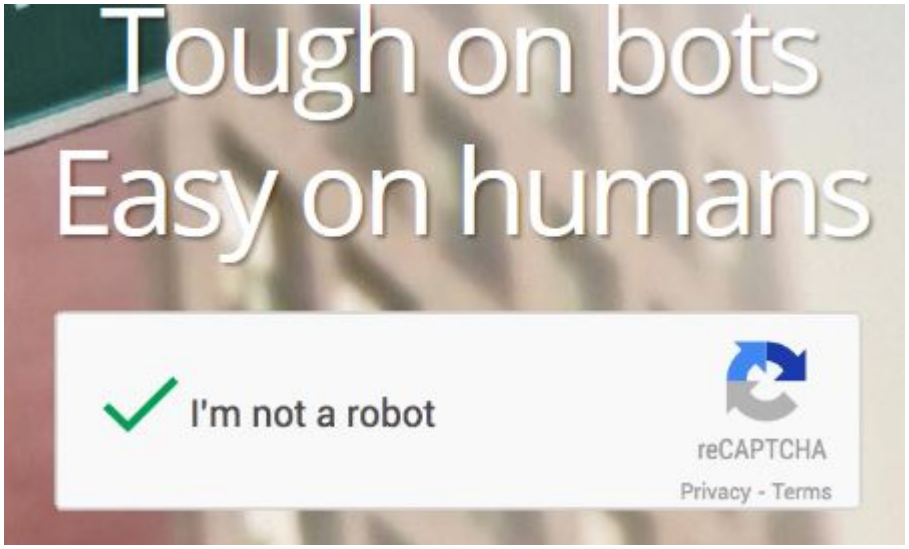
reCAPTCHA2

D'après Wikipedia « Le terme CAPTCHA est une marque commerciale de l'université Carnegie-Mellon désignant une famille de tests de Turing permettant de différencier de manière automatisée un utilisateur humain d'un ordinateur ».

En gros on veut savoir si on a affaire à un humain ou une machine. Sur Internet ce n'est pas si facile à déterminer mais c'est souvent essentiel pour éviter de se faire spammer ou attaquer en brute force.

Les première versions étaient assez laborieuses et finalement pas si efficaces que ça. Mais notre « ami » Google nous propose désormais une version simple, efficace et plutôt facile à mettre en place. L'objet de cet article est de présenter cette nouvelle version. Je vais le faire en PHP de base pour simplifier la démarche.

Imaginons un site où on veut que les utilisateurs puissent s'inscrire à une lettre d'information. On a bien tout mis en place, que ce soir avec du code propre ou par l'intermédiaire de l'une des nombreuses propositions qui existent en ligne. On crée un joli formulaire et là on ne veut pas qu'un méchant robot vienne saboter notre édifice. Alors on décide de placer un reCAPTCHA et évidemment notre choix se porte sur celui de Google :

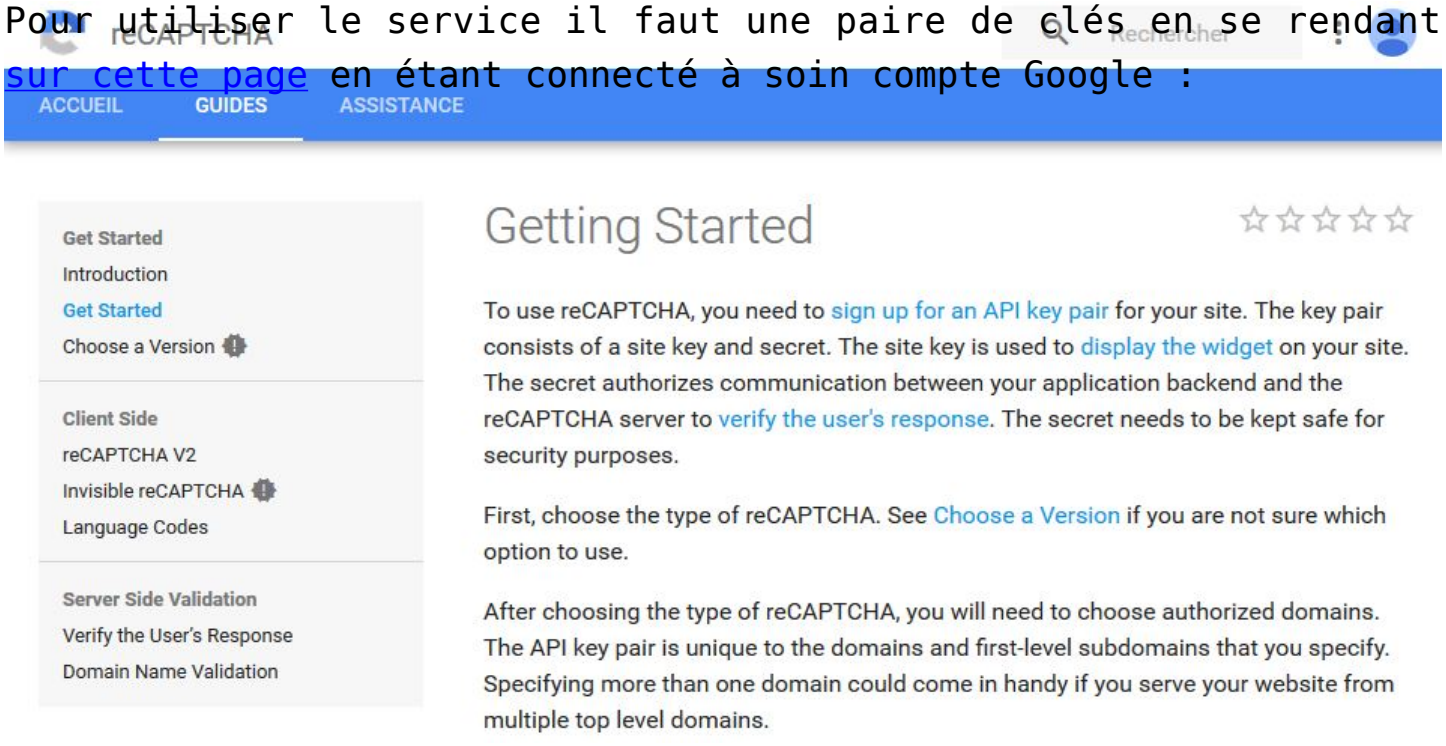


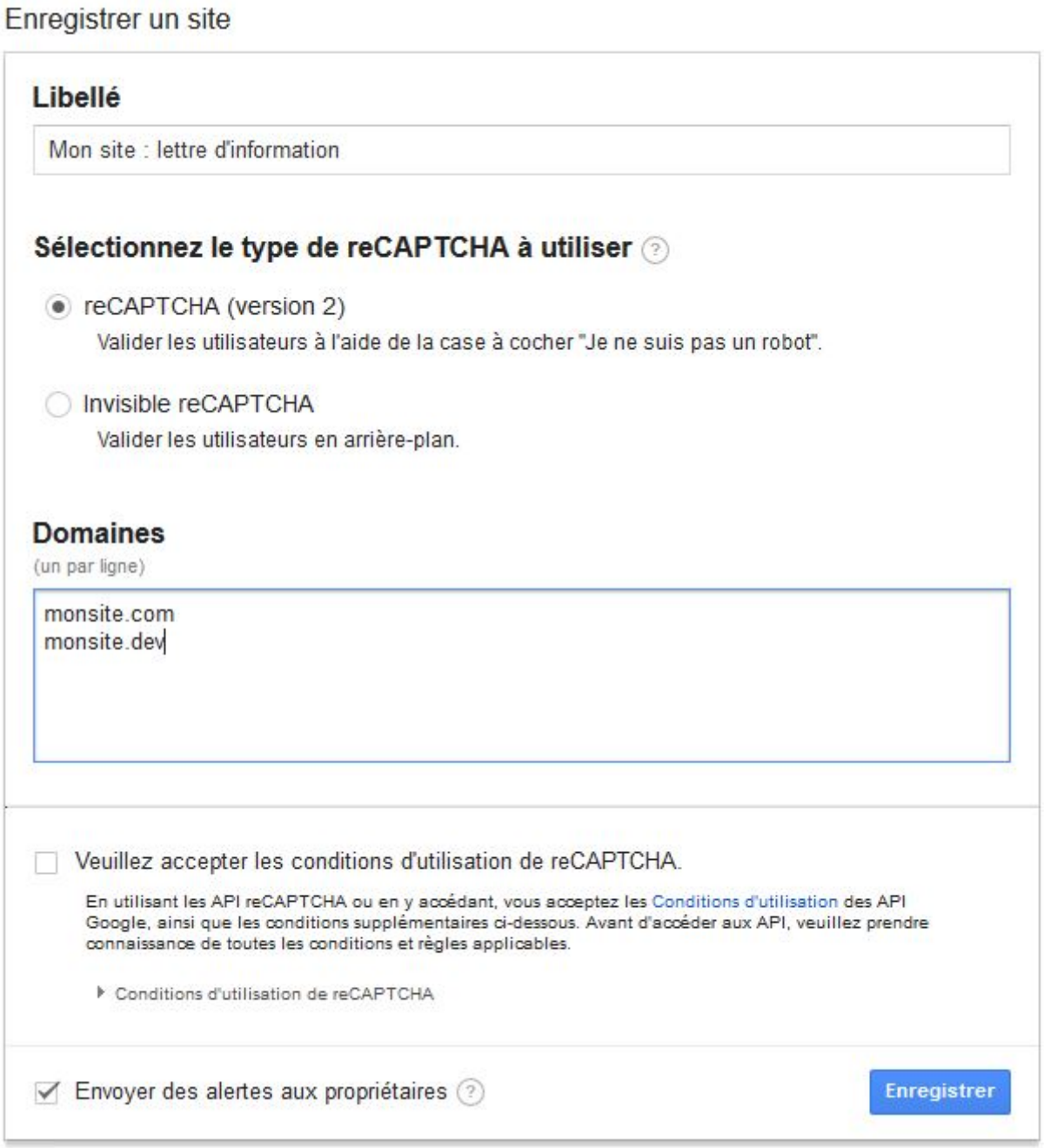
La documentation de Google

On trouve une documentation simple mais suffisante sur le site :

Récupérer les clés

Pour utiliser le service il faut une paire de clés en se rendant sur cette page en étant connecté à soin compte Google :

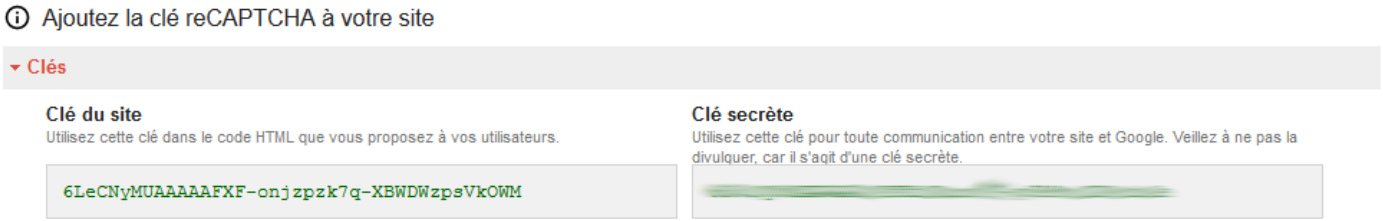




Il faut choisir le type de CAPTCHA qui peut être visible ou invisible :

Pour cet article je vais utiliser le plus classique, le visible.

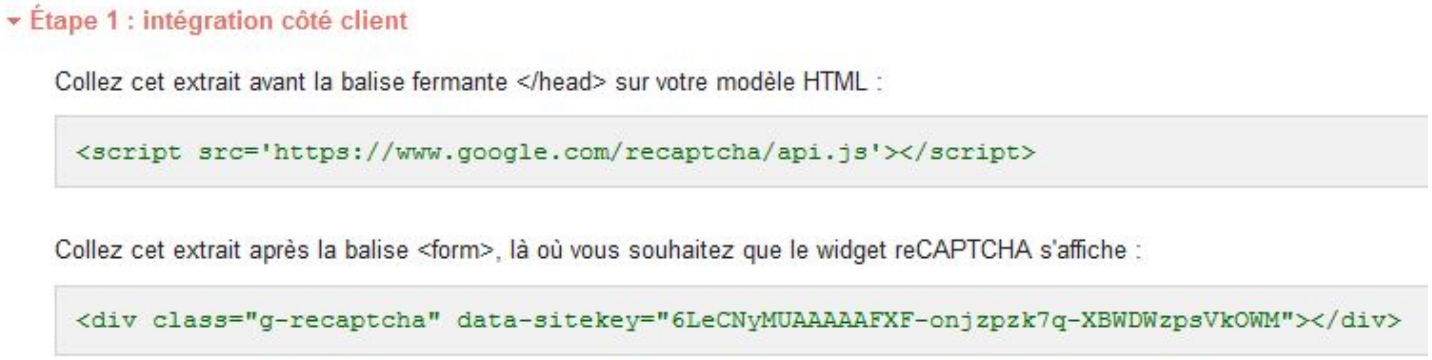
Quand on a validé on se retrouve avec ses clés :



La mise en place côté client

Elle peut se faire automatiquement (le plus simple) ou de façon explicite si on a une situation particulière. Je vais dans cet article utiliser la première version.

Sur la page où on récupère ses clés on a aussi un résumé de l’intégration côté client :



La mise en place côté serveur

La documentation décrit l'API qui est toute simple et qu'on va voir en action bientôt.

Sur la page où on récupère ses clés on a aussi un résumé de l'intégration côté serveur :

La page HTML et le Javascript

Maintenant qu'on a tous les éléments nécessaires voyons notre exemple d'application.

Voici le code complet :

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta content="text/html; charset=UTF-8" http-equiv="content-type">
     <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1.0">
```

```
    <title>Tes du reCAPTCHA2</title>
                                                    <link
href="https://fonts.googleapis.com/icon?family=Material+Icons"
rel="stylesheet">
                                <link       rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/materialize/0.98.2/cs
s/materialize.min.css">
                                                    <link
href="https://fonts.googleapis.com/css?family=PT+Sans+Narrow"
rel="stylesheet">
    <style>
        body {
            PT Sans Narrow', sans-serif;
            font-size: 18px;
            background-color: #5179d0;
        }
        #captcha {
            margin: 30px 0 20px;
        }
    </style>
</head>
<body>

  <div id="newsletter" class="modal">
      <div class="modal-content">
          <h5 class="center">Pour vous abonner (ou désabonner) à
notre lettre d'information :</h5>
          <div class="row">
                    <form id="newsform" class="col s12"
action="mail.php" method="post">
                <div class="row">
                    <div class="input-field col s12">
                        <input placeholder="Votre adresse mail"
id="email" type="email" name="email" class="validate" required>
                    </div>
                    <div class="input-field col s12">
                          <input name="action" type="radio"
value="subscribe" id="subscribe" checked>
                        <label for="subscribe">S'abonner</label>
                          <input name="action" type="radio"
value="unsubscribe" id="unsubscribe">
                                <label for="unsubscribe">Se
désabonner</label>
```

```
                    </div>
                    <div id="captcha" class="input-field col s12
center">
                        <div class="g-recaptcha" data-
sitekey="6LeCNyMUAAAAAFXF-onjzpzk7q-XBWDWzpsVkOWM"></div>
                      <div id="error" class="left red-text text-
darken-2" style="display: none">
                          Veuillez cliquer sur le Captcha,
merci.
                      </div>
                    </div>
                    <button class="btn waves-effect waves-light
right" type="submit">Envoyer
                        <i class="material-icons right">send</i>
                    </button>
                </div>
            </form>
        </div>
    </div>
  </div>

  <div id="erreurmodal" class="modal">
      <div class="modal-content">
          <h5 class="center">Erreur dans la procédure
d'abonnement</h5>
          <p class="center-align">Nous sommes désolés mais une
erreur inattendue est survenue lors de votre demande d'abonnement.
Veuillez s'il vous plait renouveller la procédure dans quelques
minutes.</p><p class="center-align">Merci.</p>
      </div>
  </div>

  <div id="subscribemodal" class="modal">
      <div class="modal-content">
          <h5 class="center">Procédure d'abonnement achevée</h5>
          <p class="center-align">Votre demande a bien été prise
en compte.</p>
          <p class="center-align">Vous allez recevoir un e-mail de
vérification dans un instant. Il contient un lien sur lequel vous
devez cliquer afin de confirmer votre inscription.</p>
          <p class="center-align">Merci.</p>
      </div>
  </div>
```

```
  <div id="unsubscribemodal" class="modal">
      <div class="modal-content">
             <h5 class="center">Procédure de désabonnement
achevée</h5>
          <p class="center-align">Votre demande a bien été prise
en compte.</p>
          <p class="center-align">Nous sommes désolés de vous voir
partir.</p>
          <p class="center-align">Merci.</p>
      </div>
  </div>

   <div id="intro" class="section scrollspy no-pad-bot center
orange-text">
    <div class="container">
      <h1 class="header">Exemple d'utilisation du reCAPTCHA2</h1>
      <div class="row center">
            <p><a id="modalcommand" href="#" class="btn"><i
class="material-icons    right">email</i>La    lettre
d'information</a></p>
       </div>
    </div>
  </div>

                                                    <script
src="https://code.jquery.com/jquery-2.1.4.min.js"></script>
                                                    <script
src="https://cdnjs.cloudflare.com/ajax/libs/materialize/0.98.2/js/
materialize.min.js"></script>
   <script src="https://www.google.com/recaptcha/api.js" async
defer></script>
  <script>
      $(function(){

          $('#newsletter').modal()
          $('#subscribemodal').modal()
          $('#unsubscribemodal').modal()
          $('#erreurmodal').modal()

          $('#newsform').submit (function(event) {
              event.preventDefault()
              $.post(
                  $(this).attr('action'),
```

```
                $(this).serialize()
            )
            .done(function(data) {
                if (data == 200) {
                  $('#newsform').modal('close')
                      if ($('input[name=action]:checked',
'#newsform').val() == 'subscribe') {
                      $('#subscribemodal').modal('open')
                  } else {
                      $('#unsubscribemodal').modal('open')
                  }
                } else if (data == 'nocaptcha') {
                  $('#error').show('slow')
                } else {
                  $('#newsletter').modal('close')
                  $('#erreurmodal').modal('open')
                }
            })
        })

        $('#modalcommand').click(function (event) {
            event.preventDefault()
            $('#newsletter').modal('open')
            $('#error').hide()
        })
    })
  </script>

  </body>
</html>
```

Pour l’esthétique j’ai utilisé Materialize que j’aime bien :

L’intégration du reCAPTCHA se fait en deux emplacements :

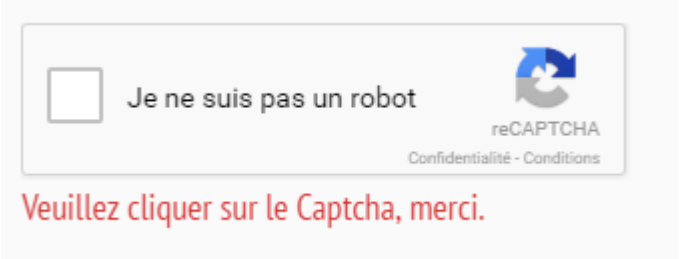
- l’appel de la librairie Javascript :

```
<script src='https://www.google.com/recaptcha/api.js'></script>
```

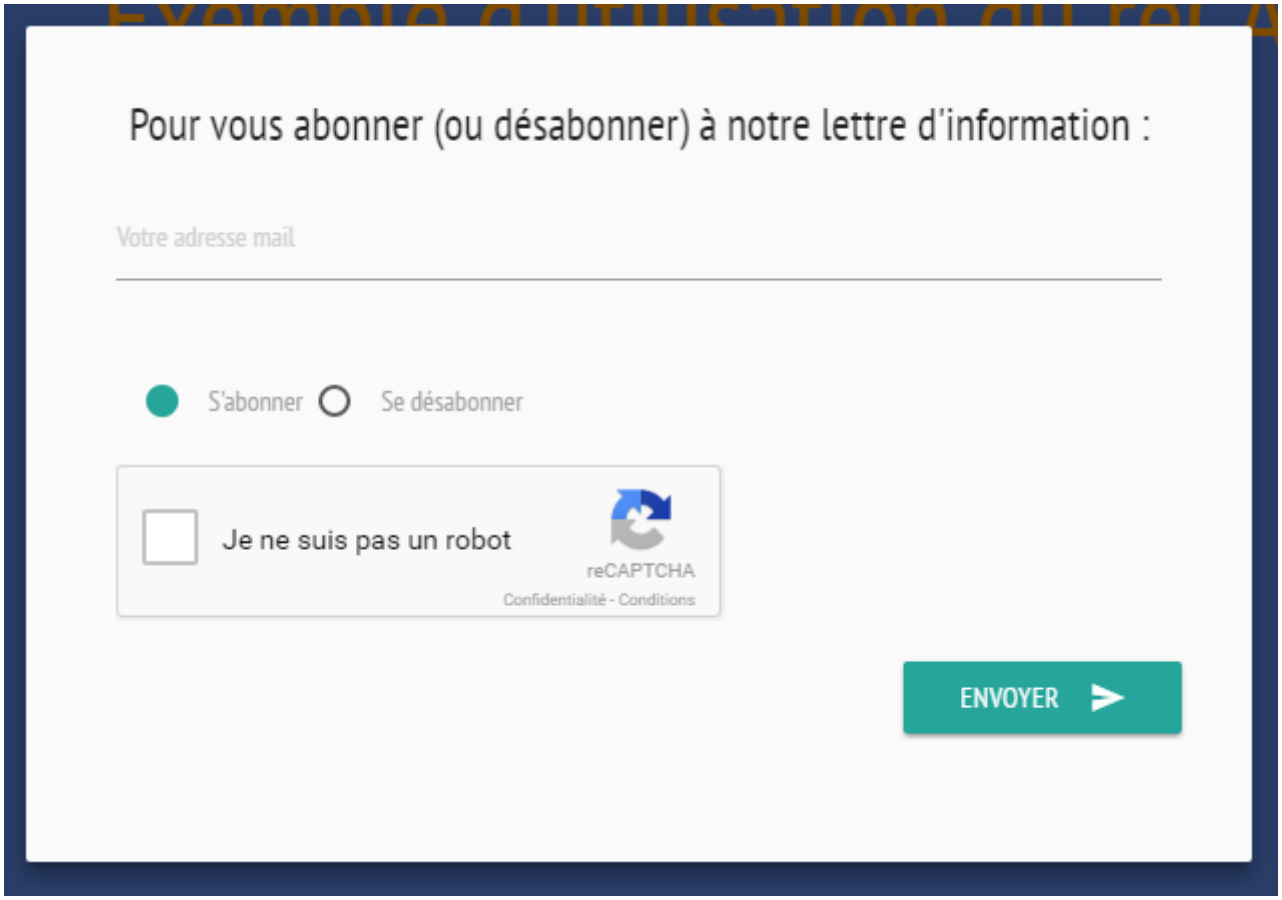
- l’ajout d’un champ dans le formulaire :

```
 <div id="captcha" class="input-field col s12 center">
      <div class="g-recaptcha" data-sitekey="6LeCNyMUAAAAAFXF-onjzpzk7q-XBWDWzpsVkOWM"></div>
       <div id="error" class="left red-text text-darken-2" style="display: none">
         Veuillez cliquer sur le Captcha, merci.
     </div>
 </div>
```

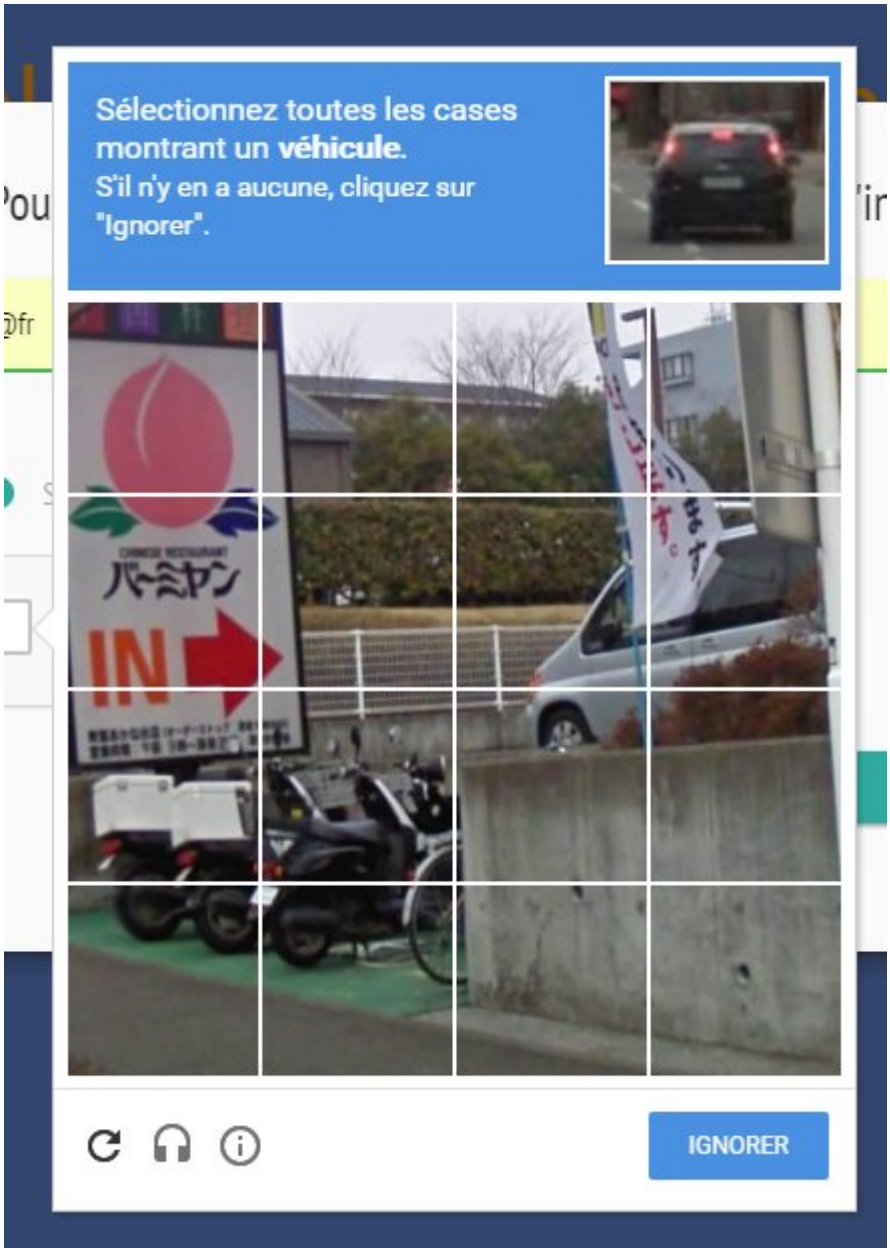
Là je prévoie le signalement de l’erreur au départ invisible si la case n’est pas cochée. Et qui apparaît lorsque ça devient utile :



Lorsqu’on clique sur le bouton le formulaire apparaît dans une fenêtre modale :



Lorsqu’on clique le reCAPTCHA on a évidemment l’apparence classique :

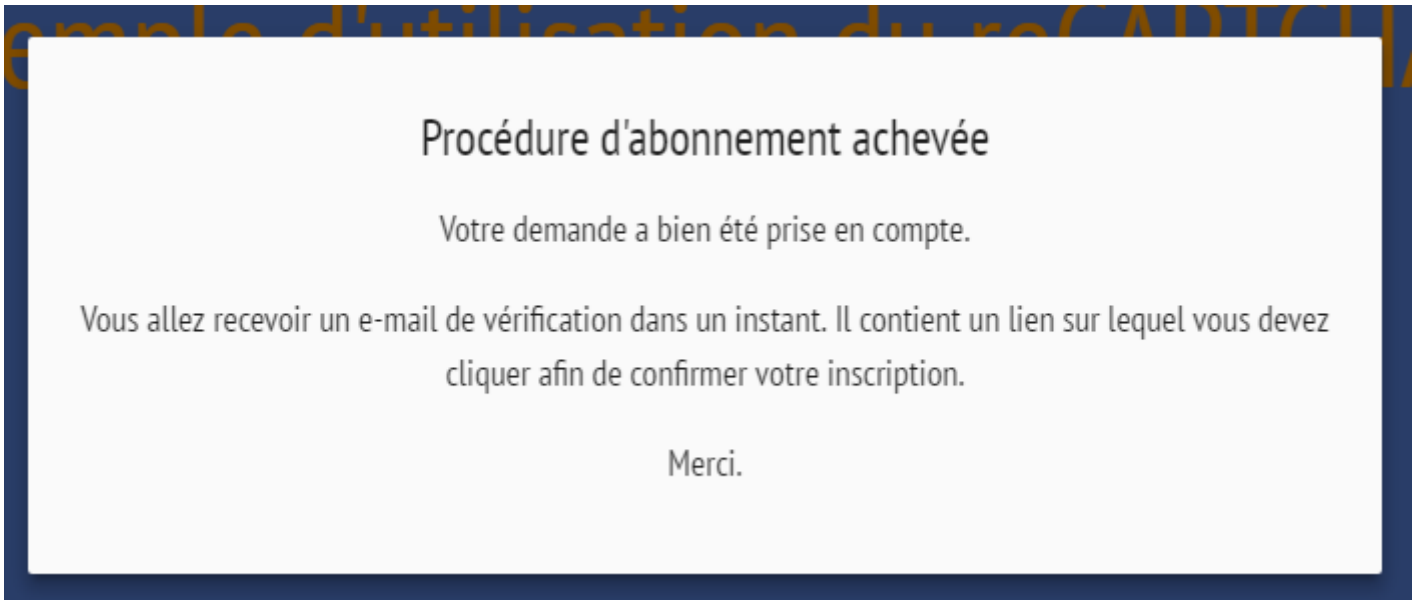


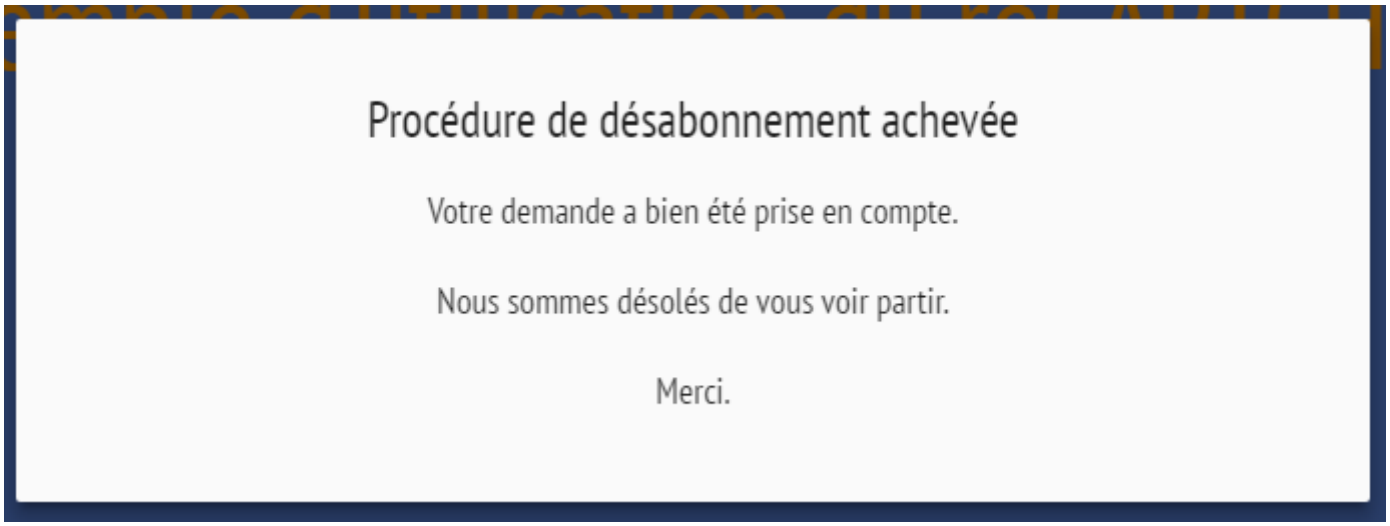
Pour la gestion j'utilise Ajax, donc la soumission et le traitement de la réponse se font avec JQuery :

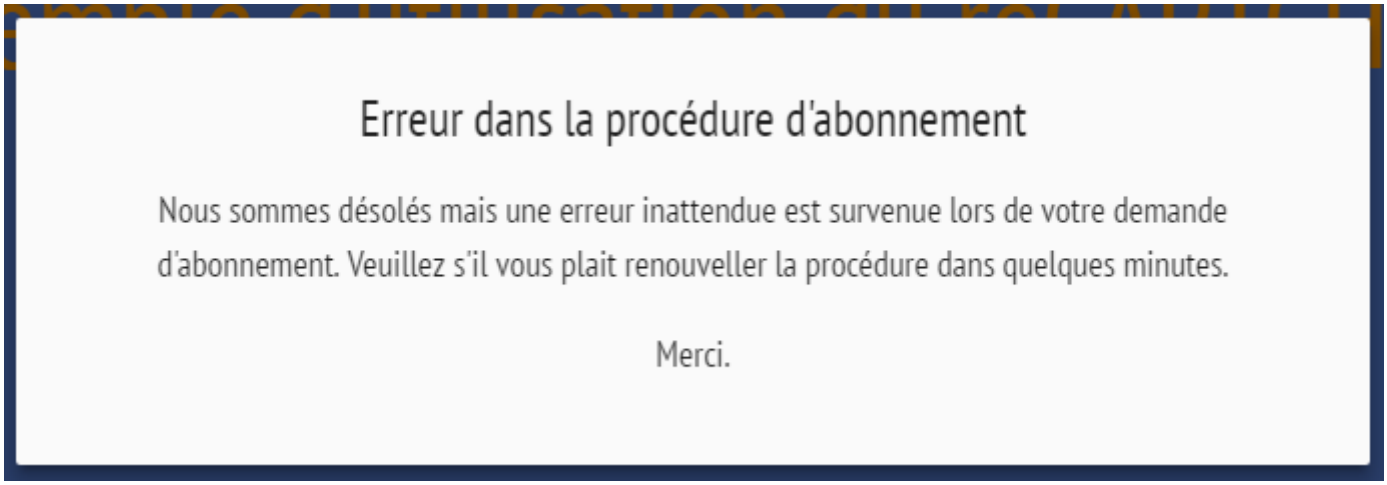
```
$('#newsform').submit (function(event) {
  event.preventDefault()
  $.post(
      $(this).attr('action'),
      $(this).serialize()
  )
  .done(function(data) {
      if (data == 200) {
        $('#newsform').modal('close')
         if ($('input[name=action]:checked', '#newsform').val() ==
'subscribe') {
             $('#subscribemodal').modal('open')
        } else {
```

```
            $('#unsubscribemodal').modal('open')
        }
      } else if (data == 'nocaptcha') {
        $('#error').show('slow')
      } else {
        $('#newsletter').modal('close')
        $('#erreurmodal').modal('open')
      }
  })
})
```

J’ai prévu 3 modales pour l’abonnement, le désabonnement et l’erreur :







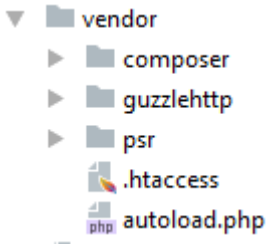
Le traitement côté serveur

Côté serveur on a du simple PHP. Pour la gestion de la requête HTTP j'ai utilisé Guzzle parce que c'est une librairie parfaite pour ne pas se soucier des détails d'implémentation.

Evidemment il faut l'installer :

composer require guzzlehttp/guzzle

Ce qui crée les dossiers avec toutes les dépendances :



Dans notre code il suffira de charger la librairie :

```
require 'vendor/autoload.php';
```

Voici le code PHP complet (fichier nommé **mail.php**) :

```
<?php

if ($_SERVER['REQUEST_METHOD'] === 'POST') {

    require 'vendor/autoload.php';

    $email = null;
    $action = null;
    $captcha = null;
```

```
    if (isset($_POST['g-recaptcha-response'])) {
        $captcha = $_POST['g-recaptcha-response'];
    }
    if (!$captcha) {
        echo 'nocaptcha';
        return;
    }

    if (isset($_POST['email'])) {
        $email = $_POST['email'];
    }
    if (isset($_POST['action'])) {
        $action = $_POST['action'];
    }

    $client = new \GuzzleHttp\Client();

    $response = $client->request (
        'POST',
        'https://www.google.com/recaptcha/api/siteverify', [
        'form_params' => [
            'secret' => 'ma_cle_secrete_cote_serveur',
            'response' => $captcha,
        ]
    ]);

    $responses = json_decode ($response->getBody ());

    if ($responses->success) {
        // Actualisation de l'abonnement sur le serveur
        echo 200;
    } else {
        echo 'fail';
    }
}
```

Dans le fonctionnement on récupère la valeur du captcha issue du formulaire :

```
$captcha = $_POST['g-recaptcha-response'];
```

Si c’est vide on renvoie une erreur :

```
if (!$captcha) {
```

```
     echo 'nocaptcha';
     return;
}
```

Si c’est bon on envoie la requête chez Google :

```
$client = new \GuzzleHttp\Client();
$response = $client->request (
     'POST',
     'https://www.google.com/recaptcha/api/siteverify', [
         'form_params' => [
         'secret' => 'ma_cle_secrete_cote_serveur',
         'response' => $captcha,
     ]
]);

$responses = json_decode ($response->getBody ());

if ($responses->success) {
    // Actualisation de l'abonnement sur le serveur
    echo 200;
} else {
    echo 'fail';
}
```

Il faut évidemment renseigner la clé secrète là.

On récupère la réponse et on adapte le retour en conséquence. Le traitement effectif de l’abonnement se ferait ici.

Conclusion

On voit que l’intégration de reCAPTCHA ne pose aucun problème technique et qu’il serait dommage de ne pas l’utiliser. Mon exemple est assez sommaire côté serveur pour se limiter à l’essentiel de la gestion.

Laravel 5.5 : les nouveautés

A chaque semestre sa nouvelle version importante de Laravel. Certains disent que c'est trop rapide, mais personne n'est obligé de suivre ce rythme ! Alors voici que se profile la version 5.5 qui est annoncée pour le mois de juillet et sera LTS. La copie est assez avancée pour pouvoir faire un peu le point. Si vous aimez la langue anglaise vous avez une description de toutes les nouveautés sur le site officiel. Sinon je vous en fais ici un petit résumé dans notre langue préférée !

PHP 7

La première grande nouveauté est la version minimale de PHP qui est désormais la 7 :

```
"require": {
    "php": ">=7.0.0",
    "laravel/framework": "5.5.*",
    ...
```

Donc si vous voulez passer à cette version de Laravel il faudra peut-être mettre à jour votre serveur !

Le principal intérêt se trouve au niveau des performances parce que cette version de PHP est vraiment plus rapide. Mais ce n'est pas le seul avantage, vous trouvez la liste des nouveautés dans le manuel PHP.

Les erreurs

Les pages d'erreurs

L'aspect des pages d'erreur par défaut (donc pour 404, 419, 500 et 503) ont été relookées. C'est un changement purement cosmétique mais ça fait du bien aux yeux :

Mais on peut évidemment créer son propre aspect comme dans les versions précédentes en créant un dossier **resources/views/errors** et en nommant les fichiers blade avec le numéro de l’erreur.

Les helpers throw_if et throw_unless

Deux nouveaux helpers apparaissent :

```
$foo = false;
throw_if($foo, new MonException('Foo est faux'));
// ou
throw_if($foo, MonException::class, 'Foo est faux');

$foo = true;
throw_unless($foo, new MonException('Foo est faux'));
// ou
throw_unless($foo, MonException::class, 'Foo est faux);
```

Retour des données de la validation

Laravel offre la puissante méthode **validate** pour valider les données :

```
public function store(Request $request)
{
```

```
    $this->validate($request, [
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
    ]);

    return Post::create($request->all());
}
```

La méthode jusque là ne retournait aucune valeur. Désormais elle va retourner les données de la requête. Du coup on pourra écrire :

```
public function store()
{
    $data = $this->validate(request(), [
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
    ]);

    return Post::create($data);
}
```

Il faut évidemment que toutes les données passent par la validation, quitte à ne préciser aucune règle.

Personnellement j'utilise principalement des requêtes de formulaire et cette possibilité me laisse assez indifférent. D'autre part elle ne me paraît pas être d'un très grand intérêt.

La commande migrate:fresh

Voici l'annonce officielle de cette commande.

Vous utilisez sans doute souvent la commande **migrate:refresh**. Elle permet de revenir à zéro pour repartir sur une migration toute fraîche. Mais pour que ça fonctionne il faut prévoir des méthodes **down** dans les migrations pour définir le retour en arrière.

Avec la commande **migrate:fresh** plus besoin de méthode **down**, les tables sont purement et simplement supprimées et les migrations relancées. On peut lui adjoindre l'option **–seed** pour remplir les tables. Voilà une commande que je vais beaucoup utiliser !

La commande vendor:publish

Voilà une commande qui m'a souvent agacé parce que telle qu'elle elle publie tout ce qu'elle trouve et ce n'est pas toujours judicieux !

Désormais lorsque vous utiliserez cette commande sans préciser de provider vous aurez la liste de tous les providers et vous pourrez choisir celui que vous voulez publier !

Voici une animation bien faite pour cette commande :

Les mails

Les thèmes

La gestion des mails a bien progressé dans les dernières versions de Laravel, par exemple avec la possibilité d'utiliser le marquage **Markdown**. Cette fois on va pouvoir utiliser des thèmes. Dans la version 5.4 il y a un thème par défaut. On n'est pas obligé de l'utiliser mais ça oblige à quelques changements : créer évidement son propre CSS :

resources/views/vendor/mail/html/themes/mon-theme-a-moi.css

et aussi informer Laravel (**config/mail.php**) :

```
/*
|--------------------------------------------------------------------------
| Markdown Mail Settings
|--------------------------------------------------------------------------
|
| If you are using Markdown based email rendering, you may configure your
| theme and component paths here, allowing you to customize the design
| of the emails. Or, you may simply stick with the Laravel defaults!
|
*/

'markdown' => [
    'theme' => 'mon-theme-a-moi',

    'paths' => [
        resource_path('views/vendor/mail'),
    ],
],
```

Ce n’est pas bien compliqué mais la version 5.5 nous simplifie un peu plus la vie !

Il faut encore évidemment créer le fichier CSS mais pour sa déclaration ça se passe directement dans la classe Mailable :

```
class MonMailAMoi extends Mailable
{
    protected $theme = 'mon-theme-a-moi';
    ...
}
```

Le rendu dans le navigateur

Lorsqu’on développe un thème ça dure évidemment un moment et on

fait plein de réglage. On ne va pas chaque fois envoyer réellement un mail pour vérifier le rendu. Il existe des solutions mais aucune de vraiment simple et directe.

Avec Laravel 5.5 on peut visualiser le mail directement dans le navigateur. Il suffit d'ajouter une route :

```
Route::get('/demo', function () {
    return new App\Mail\MonMailAMoi();
});
```

Et c'est tout !

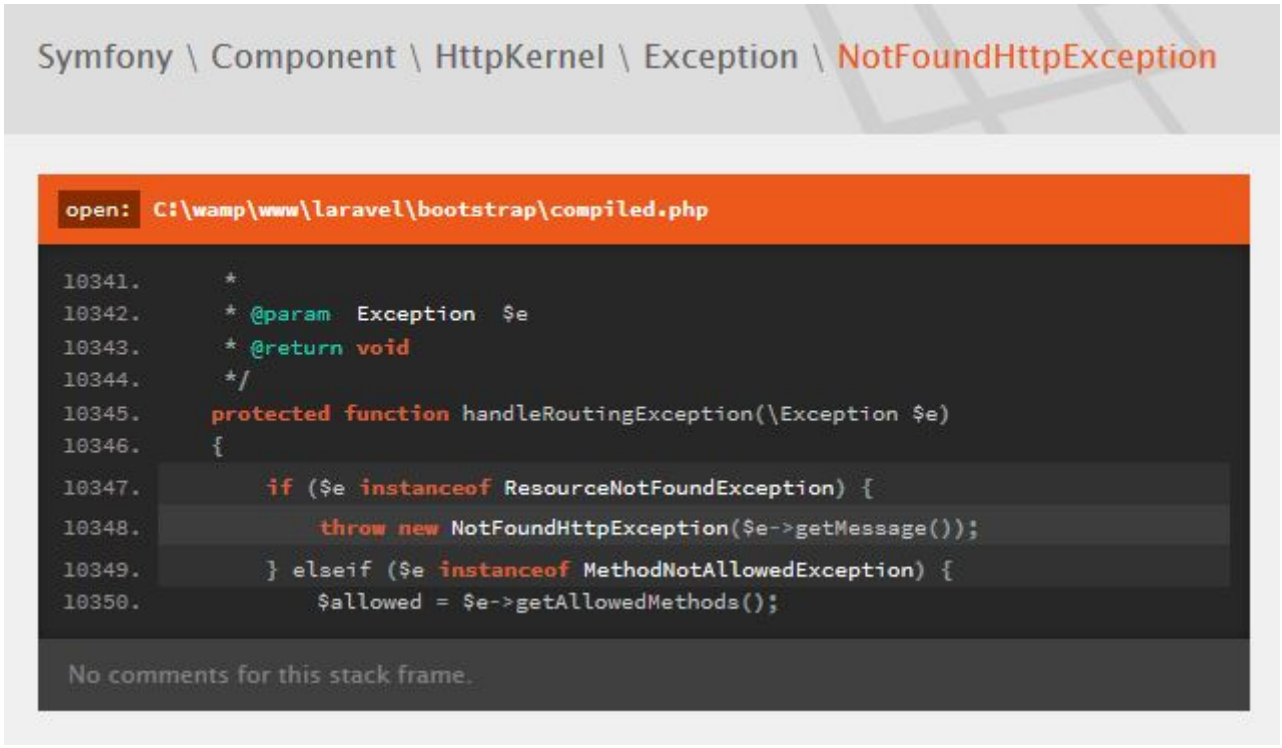
Ca peut d'ailleurs donner d'autres idées d'utilisation…

Le frontend

Le frontend a connu une évolution un peu houleuse au cours des versions. Avec Laravel 5.5 ça devient plus modulable et rien n'est imposé. On a toujours Bootstrap et Vue.js par défaut mais on peut facilement (avec Artisan) changer pour React. Je suppose que ça va encore évoluer avec d'autres propositions.

Whoops

Ceux qui ont utilisé Laravel 4 doivent se rappeler de la fenêtre d'affichage des erreurs Whoops :



Et bien on va retrouver cette sympathique librairie dans la version 5.5 !

Chargement des packages

Taylor a présenté récemment une nouvelle fonctionnalité pour l'installation des packages. On est tous habituer à effectuer toujours les mêmes opérations pour installer un package :

- utilisation de composer
- déclaration du service providers dans la configuration
- déclaration éventuelle de la façade dans la configuration

Avec la version 5.5 on pourra se contenter de composer, le reste sera reconnu automatiquement si le package est configuré pour ça !

Conclusion

Laravel 5.5 ne nous apporte pas des changements très importants mais se présente plutôt comme l'aboutissement de la série 5.

Laravel 5.4 : les nouveautés

La sortie de la vesion 5.4 de Laravel est prévue en janvier 2017 et elle aura besoin de 6 mois pour avoir une élimination des tous les bugs. On peut donc aller voir ce qui nous attend très bientôt dans cette nouvelle version !

La page officielle de présentation des nouveautés est ici. Mais pour ceux qui préfèrent une présentation française je vais détailler un peu tout ça dans cet article.

Pour faire des essais il faut aller chercher la version développement sur Github et l'installer.

Blade : component et slot

La présentation officielle de cette nouveauté est ici. On va avoir plus de possibilités pour organiser nos vues orchestrées par Blade.

Etat actuel

En général on utilise un template (**layout.blade.php**). Voici un exemple standard avec initialisation de bootstrap :

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
     <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Mon joli site</title>
                                <link       rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
```

```
    <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media queries -->
    <!-- WARNING: Respond.js doesn't work if you view the page via file:// -->
    <!--[if lt IE 9]>
                                                    <script src="https://oss.maxcdn.com/html5shiv/3.7.3/html5shiv.min.js"></script>
                                                    <script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
    <![endif]-->
  </head>
  <body>
    <div class="container">
      @yield('contenu')
    </div>
                                                    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
                                                    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
  </body>
</html>
```

On prévoit un emplacement pour gérer le contenu avec :

```
@yield('contenu')
```

Ensuite dans nos pages on référence le template et l’emplacement prévu (**home.blade.php**) :

```
@extends('layout')
@section('contenu')
    <h1>Ma page d'accueil !</h1>
@endsection
```

Tout ça fonctionne très bien et jusque là on s’en est contentés !

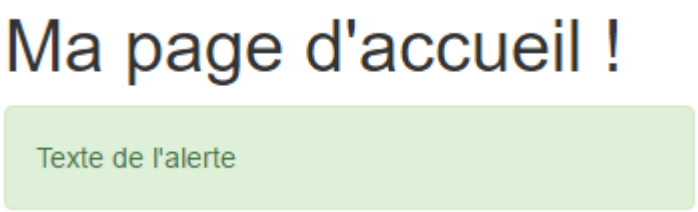
Maintenant si on veut insérer une barre d’alerte dans la vue on

peut utiliser une inclusion avec passage d'un paramètre :

```
@extends('layout')
@section('contenu')
    <h1>Ma page d'accueil !</h1>
    @include('alerte', ['texte' => 'Texte de l\'alerte'])
@endsection
```

Avec ce fichier pour l'alerte (**alerte.blade.php**) :

```
<div class="alert alert-success" role="alert">
    {{ $texte }}
</div>
```



Là aussi ça fonctionne bien…

Un composant

Avec la version 5.4 on dispose pour faire la même chose de la possibilité de créer un composant :

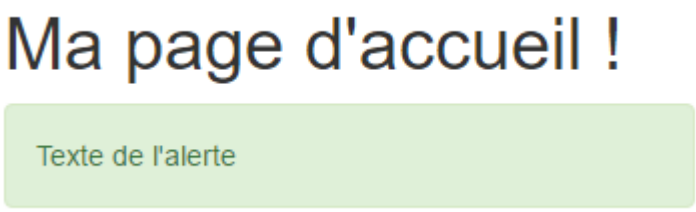
```
@extends('layout')
@section('contenu')
    <h1>Ma page d'accueil !</h1>
    @component('alerte')
        Le message d'alerte ici !
    @endcomponent
@endsection
```

On utilise la nouvelle instruction **@component** et on nomme la vue en relation. On ne nomme pas de paramètre mais on met le texte de l'alerte directement.

Voici le code de l'alerte maintenant :

```
<div class="alert alert-success" role="alert">
    {{ $slot }}
</div>
```

Le texte de l'alerte va occuper la place de **$slot**.



Le résultat est le même mais c'est quand même plus propre et élégant !

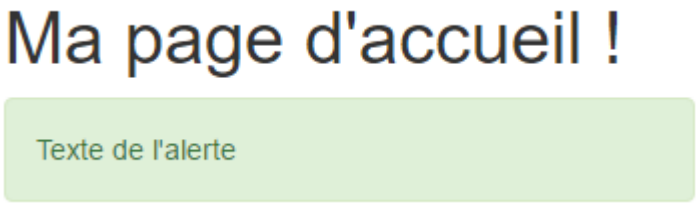
Un autre aspect intéressant de **$slot** c'est que toute variable déclarée dans le composant est utilisable dans la vue principale. C'est pas clair ? Alors un petit exemple… on utilise ce code dans le template :

```
<div class="container">
  <h1>{{ $titre }}</h1>
  {{ $slot }}
</div>
```

On attend une variable **$titre** ainsi qu'un contenu pour le **$slot**. Donc dans notre template on aura forcément un composant qui lui-même peut utiliser d'autres composants.

Voici le code de la page d'accueil :

```
@component('layout')
    @slot('titre')
        Ma page d'accueil !
    @endslot
    @component('alerte')
        Le message d'alerte ici !
    @endcomponent
@endcomponent
```



Le résultat est encore le même. Regardez comment est renseignée la

variable **titre** pour le layout. On peut ainsi créer autant de variables qu'on a besoin.

Ca nous donne une autre façon d'envisager l'organisation de nos vues avec des composants, elle s'inspire de Vue.js.

Façade automatique

La présentation officielle de cette nouveauté est ici.

Etat actuel

Imaginez que vous ayez créé la librairie mathématique du siècle :

```
<?php

namespace App\Services;

class Maths
{
    public function double($i)
    {
        return 2 * $i;
    }
}
```

Pour l'utiliser vous pouvez l'injecter dans un contrôleur :

```
<?php

namespace App\Http\Controllers;

use App\Services\Maths;

class TestController
{
    public function test(Maths $maths, $i)
    {
        echo $maths->double($i);
    }
}
```

Il suffit d’avoir une bonne route :

```
Route::get('test/{i}', 'TestController@test');
```

Et le tour est joué ! Pour :

```
.../test/8
```

Vous allez obtenir le résultat :

```
16
```

Et puis vous vous dites qu’une façade ça serait bien pour une si belle librairie !

Alors vous créez la façade :

```
<?php

namespace App\Services;

use Illuminate\Support\Facades\Facade;

class MathsFacade extends Facade
{
    protected static function getFacadeAccessor()
    {
        return 'maths';
    }
}
```

Vous déclarez votre classe dans **AppServiceProvider** :

```
public function register()
{
    $this->app->bind('maths', function ($app) {
        return new \App\Services\Maths;
    });
}
```

Puis vous déclarez la façade dans **config/app.php** :

```
'Maths' => App\Services\MathsFacade::class,
```

Et là magie de Laravel dans le contrôleur la façade fonctionne :

```php
<?php

namespace App\Http\Controllers;

use Maths;

class TestController
{
    public function test($i)
    {
        echo Maths::double($i);
    }
}
```

La façade automatique

Bon, on peut trouver que la mise en place est un peu laborieuse. Avec la version 5.4 ça sera beaucoup plus facile : oubliées la création et la déclaration de la façade !

Vous avez juste à faire ça dans le contrôleur :

```php
<?php

namespace App\Http\Controllers;

use Facades\ { App\Services\Maths };

class TestController
{
    public function test($i)
    {
        echo Maths::double($i);
    }
}
```

Et ça fonctionne ! Laravel devient de plus en plus magique !

Les routes

La présentation officielle des nouveautés sont ici.

On a une amélioration du cache. d'autre part on peut déclarer le nom d'une route avec une syntaxe simplifiée.

La syntaxe actuelle est celle-ci :

```
Route::get('test/{i}', 'TestController@test')->name('maths');
```

Avec la version 5.4 on pourra écrire :

```
Route::name('maths')->get('test/{i}', 'TestController@test');
```

On peut traiter de la même manière les middlewares.

Bon c'est pas l'évolution la plus géniale il faut l'avouer ! Mais bon, on prend quand même…

Le multilanguage

La présentation officielle des nouveautés sont ici.

Etat actuel

Pour faire du multilangue on dispose de l'helper **trans** et on doit créer autant de fichiers que de langues prévues. Par exemple on a un fichier **en/test.php** pour l'anglais :

```
<?php

return [
    'test' => 'The english version of test',
];
```

Et le même pour le français **fr/test.php** :

```
<?php

return [
    'test' => 'La version française de test',
];
```

Si dans un contrôleur j'utilise :

```
echo trans('test.test');
```

J’aurais selon la locale :

The english version of test

ou

La version française de test

Ca fonctionne plutôt bien mais c’est vrai que ça devient vite laborieux pour un gros site.

D’autre part on a la possibilité d’utiliser des paramètres, par exemple :

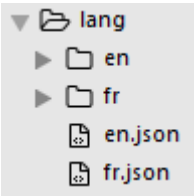
```
<?php

return [
    'test' => 'The english version of :name',
];
```

Et pour l’appel :

```
echo trans('test.test', ['name' => 'Toto']);
```

Utilisation de fichiers JSON

Avec la version 5.4 de Laravel on a une amélioration avec l’utilisation d’un fichier JSON par langue :



Voici celui de l’anglais pour notre exemple (**en.json**) :

```
{"test:": "The english version of test"}
```

Et celui du français (**fr.json**) :

```
{"test:": "La version française de test"}
```

Maintenant la syntaxe appelante est celle-ci :

```
echo __('test:');
```

Le fait de disposer de fichiers en JSON qui est un language utilisé universellement devrait simplifier les traductions et donner lieu à l'émergence d'outils pratiques.

On peut aussi utiliser des paramètres comme dans la version actuelle :

```
{"test:": "The english version of :name"}
```

Avec cet appel :

```
echo __('test:', ['name' => 'Toto']);
```

Higher Order Messaging (HOM)

La présentation officielle est ici.

Il s'agit d'un design pattern destiné à simplifier la syntaxe lorsqu'on doit effectuer des boucles pour des langages orientés objet. La bonne nouvelle c'est qu'il est appliqué aux collections de Laravel dans la version 5.4 !

Par exemple on a cette classe :

```
class Personne
{
    protected $name;
    public function __construct($name)
    {
        $this->name = $name;
    }
    public function afficheNom()
    {
        echo $this->name . '<br>';
    }
}
```

On crée une collection de personnes :

```
$collection = collect([
    new Personne('Dupont'),
```

```
    new Personne('Fremoulle'),
    new Personne('Tartignole')
]);
```

Pour afficher tous les noms on fait classiquement ceci :

```
$collection->each(function($item) {
    $item->afficheNom();
});
```

Dupont
Fremoulle
Tartignole

Avec le nouveau design pattern on pourra écrire avec cette syntaxe simplifiée :

```
$collection->each->afficheNom();
```

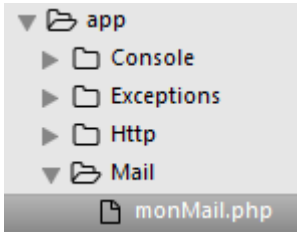
Bon… c'est du détail mais ça participe à l'élégance de Laravel.

Des emails avec Markdown

Avec Laravel 5.3 on dispose de **Mailable** pour créer facilement des emails. Il suffit d'utiliser cette commande d'artisan :

```
php artisan make:mail monMail
```

Et la magie opère, un dossier et un fichier sont créés :



Avec ce code :

```
<?php

namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Mail\Mailable;
```

```
use Illuminate\Queue\SerializesModels;
use Illuminate\Contracts\Queue\ShouldQueue;

class monMail extends Mailable
{
    use Queueable, SerializesModels;

    /**
     * Create a new message instance.
     *
     * @return void
     */
    public function __construct()
    {
        //
    }

    /**
     * Build the message.
     *
     * @return $this
     */
    public function build()
    {
        return $this->view('view.name');
    }
}
```

Il ne reste plus qu'à créer la vue et le tour est joué et au besoin jouer avec quelques méthodes pour savoir qui envoie, transmettre des paramètree… Mais la vue utilise du HTML ou à la rigueur du texte simple.

Avec Laravel 5.4 la commande d'artisan s'enrichit pour le markdown :

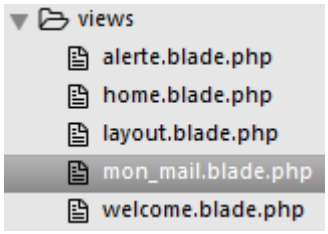
```
php artisan make:mail MonMail --markdown=mon_mail
```

On a encore création du dossier et du fichier comme avant mais la syntaxe change un peu pour la méthode **build** :

```
public function build()
{
    return $this->markdown('mon_mail');
```

}

On ne retroune plus avec **view** mais avec **markdown** et en plus une vue est créée :



Avec ce code de base :

```
@component('mail::message')
# Introduction

The body of your message.

@component('mail::button', ['url' => ''])
Button Text
@endcomponent

Thanks,<br>
{{ config('app.name') }}
@endcomponent
```

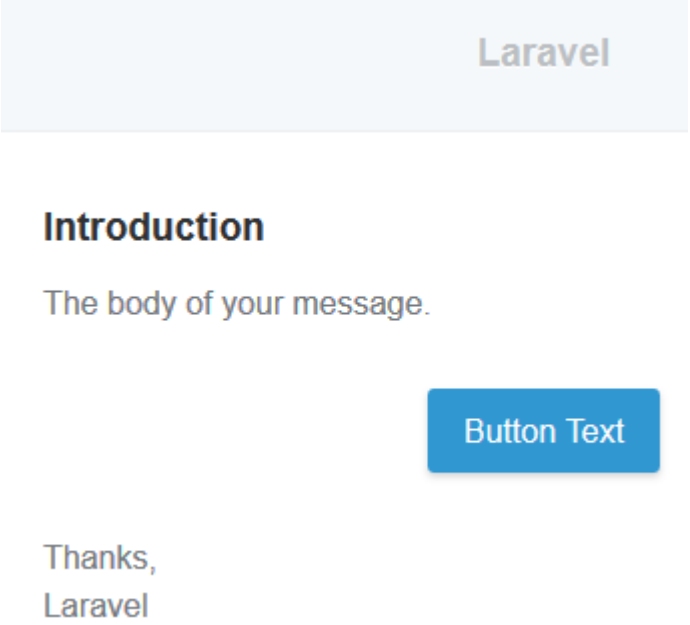
La syntaxe est fondée sur le nouveau composant de Blade dont j’ai parlé ci-dessus.

Il paraît qu’on dispose de ces composants :

- button
- footer
- header
- layout
- message
- panel
- promotion
- subcopy
- table

Ca permet de faire pas mal de choses !

Si on envoie tel quel voici le résultat obtenu :



Pour vérifier si ça marche je tente un tableau :

```
On va tenter un tableau :
@component('mail::table')
Item     | Valeur
-------- | ---
Fraises  | 12 €
Banane   | 3 €
Pipe     | 50 €
@endcomponent
```

Avec ce résultat :

Je n’ai pas essayé les autres composants mais c’est très prometteur !

On a les mêmes possibilités avec les notifications.

Laravel Dusk

Pour terminer on va évoquer les tests. Laravel permet de mettre en place des tests puissants d'application, on peut ainsi remplir un formulaire, cliquer sur un lien… Tout se passe bien tant que Javascript ou encore pire Ajax ne s'en mêlent pas parce que ce qui est utilisé c'est le **BrowserKit** de Symfony.

Dusk arrive à digérer le Javascript, il offre les mêmes possibilités que le BrowserKit et reconnait même les glisser-déposer !

Pour le moment il en est à la phase alpha. Pour se faire une idée il y a un exemple sur Github.

Laravel Mix

Si vous utilisez Laravel vous connaissez forcément Elixir qui facilité l'utilisation de Gulp pour gérer CSS et Javascript. Avec Laravel 5.4 on abandonne gulp en faveur de Webpack et pour l'occasion le nom change aussi pour **Laravel Mix**. En attendant d'avoir plus d'informations on peut déjà se documenter sur Webpack si on le connaît peu ou pas.

Deux nouveaux middlewares

Trim Strings

Avec ce middleware non actif par défaut on n'aura plus de souci concernant la saisie d'espaces parasites surout en fin de chaîne de caractères. Les espaces avant et arrières seront éradiqués.

Convert Empty Strings to Null

On a parfois des soucis avec des colonnes nullables lorsque rien n'est saisit dans un formulaire. Avec ce middleware non actif par défaut on aura une transformation en valeur nulle donc directement envoyable dans la base.

Conclusion

Les nouveautés ne sont pas aussi radicales qu'avec la version 5.3 mais il y a des choses bien intéressantes ! Si vous voulez donner votre avis sur le sujet…

Laravel 5.3 : les nouveautés

La version 5.3 est sortie et il est temps d'aller voir les nouveautés ! Elles sont nombreuses, certaines importantes et d'autres minimes, on va faire un peu le tour de tout ça sans toutefois épuiser le sujet !

J'ai actualisé mon exemple sur github avec un nouveau dépôt. J'en ai profité pour nettoyer le code et le réorganiser, j'ai utilisé les notifications pour l'envoi d'emails et j'ai aussi ajouté des tests !

Mais la première chose à savoir c'est que la version minimale de PHP est désormais la 5.6.4.

Ce passage de PHP 5.5.* à PHP 5.6.* nous donne la possibilité d'utiliser de nouvelles fonctionnalités. Vous en avez la liste sur cette page du manuel. En gros on a comme nouveautés :

- des expressions scalaires dans l'affectation des constantes

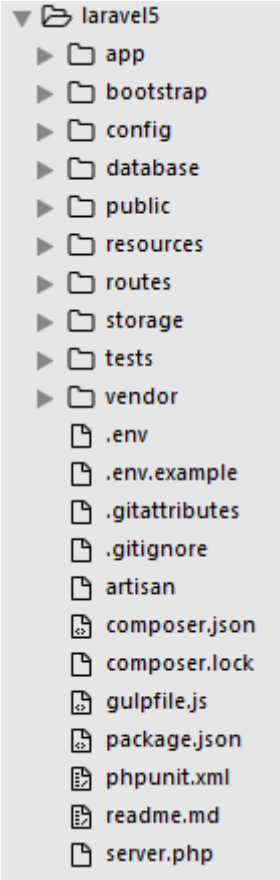
(bon, faut en avoir besoin…),
- l'opérateur … qui va faciliter l'utilisation des fonctions avec un nombre d'arguments variable,
- le nouveau opérateur ** pour les puissances,
- l'utilisation de **use** pour les constantes et les fonctions…

Structure

On va commencer par installer Laravel 5.3 :

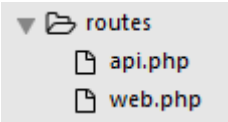
```
composer create-project laravel/laravel laravel5
```

Voici la structure que j'obtiens :

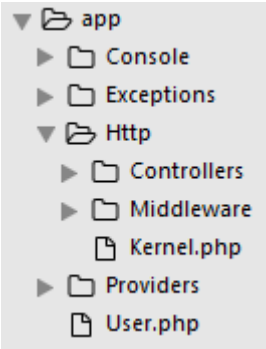


Routes et middlewares

A première vue pas de grand changements, toutefois le dossier **routes** attire mon regard ! Regardons son contenu :



On se retrouve avec deux fichiers : un pour le web et l'autre pour les api. J'en conclus que le fichier des routes dans le dossier **app** a disparu, voyons cela :



Apparemment il n'y a pas que cela qui a disparu, ce dossier a fait une cure d'amaigrissement ! Mais pour le moment on va s'intéresser aux routes.

Voyons le contenu de **web.php** :

```
<?php

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| This file is where you may define all of the routes that are handled
| by your application. Just tell Laravel the URIs it should respond
| to using a Closure or controller method. Build something great!
|
*/

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

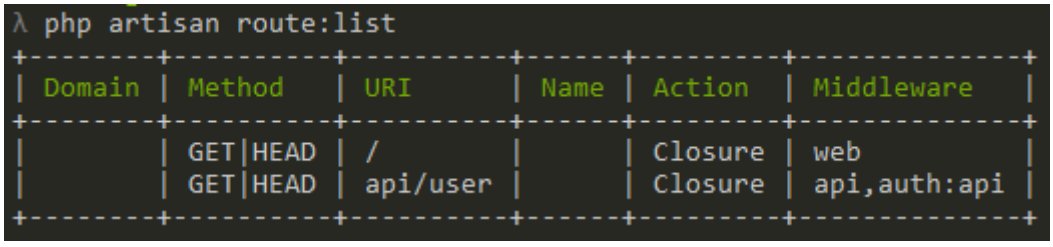
Une simple route pour la racine qui renvoie la vue **welcome**.

Et pour **api.php** :

```
<?php

use Illuminate\Http\Request;

/*
|--------------------------------------------------------------------------
| API Routes
|--------------------------------------------------------------------------
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| is assigned the "api" middleware group. Enjoy building your API!
|
*/

Route::get('/user', function (Request $request) {
    return $request->user();
});
```

Là aussi une simple route pour **user** qui renvoie l'utilisateur connecté.

La partie intéressante va se trouver au niveau des middlewares pour ces deux routes. Voyons un listing :



On voit qu'on a :

- pour la route **web** : le middleware **web**,
- pour la route **api** : les middlewares **api** et **auth:api**. On voit

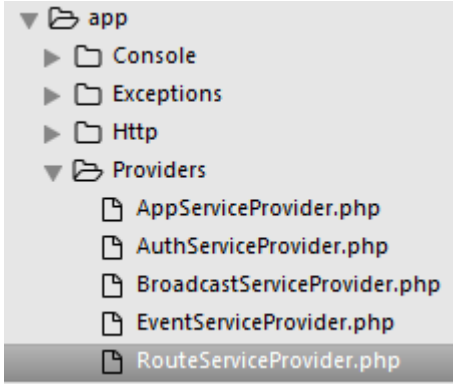
également l'ajout automatique du préfixe **api**.

Voyons dans **app/Http/Kernel.php** de plus près ces middlewares :

```
/**
 * The application's route middleware groups.
 *
 * @var array
 */
protected $middlewareGroups = [
    'web' => [
        \App\Http\Middleware\EncryptCookies::class,
\Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
        \Illuminate\Session\Middleware\StartSession::class,
        \Illuminate\View\Middleware\ShareErrorsFromSession::class,
        \App\Http\Middleware\VerifyCsrfToken::class,
        \Illuminate\Routing\Middleware\SubstituteBindings::class,
    ],

    'api' => [
        'throttle:60,1',
        'bindings',
    ],
];
```

Pour le **web** on a les classiques : cookies, session, protection CSRF… et pour l'**api** un throttle et des bindings.

Mais où se fait l'affectation de ces middlewares pour nos deux fichiers de routes ? Pour ça il faut aller jeter un oeil dans le provider :



On y trouve ce code pour le **web** :

```
/**
```

```
 * Define the "web" routes for the application.
 *
 * These routes all receive session state, CSRF protection, etc.
 *
 * @return void
 */
protected function mapWebRoutes()
{
    Route::group([
        'middleware' => 'web',
        'namespace' => $this->namespace,
    ], function ($router) {
        require base_path('routes/web.php');
    });
}
```

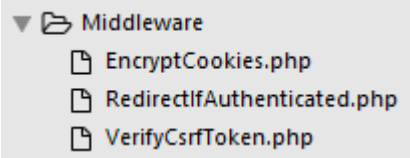
Un groupe avec le middleware **web** et un espace de nom qui est une propriété avec la valeur **App\Http\Controllers**. D'autre part on pointe le fichier **routes/web.php**.

Pour la partie **api** on a :

```
/**
 * Define the "api" routes for the application.
 *
 * These routes are typically stateless.
 *
 * @return void
 */
protected function mapApiRoutes()
{
    Route::group([
        'middleware' => ['api', 'auth:api'],
        'namespace' => $this->namespace,
        'prefix' => 'api',
    ], function ($router) {
        require base_path('routes/api.php');
    });
}
```

On déclare les deux middleware : **api** et **auth:api**. Pour l'espace de nom évidemment c'est le même que pour le web. On a aussi le préfixe **api**. Et évidemment on pointe sur le fichier **routes/api.php**.

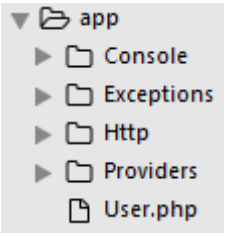
Si on regarde maintenant le dossier des middlewares :



On en trouve plus guère ici ! Il y en a comme **auth** qui ont basculé dans le framework.

Le dossier app

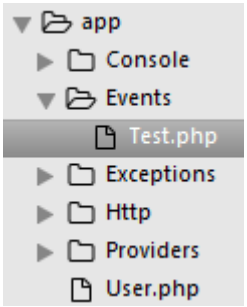
Revenons en à ce dossier **app** amaigri :



On a perdu pas mal de dossiers : Events, Jobs, Listeners, Policies. Mais finalement ces dossiers au départ étaient vides. Le parti pris a été de ne pas les prévoir et de les créer dynamiquement dès qu'on en a besoin. Par exemple si je crée un événement :

```
λ php artisan make:event Test
Event created successfully.
```

Voyons ce qu'il s'est passé :



Le dossier a été créé en même temps que l'événement. Il en est de même pour les autres dossiers. On va dire que c'est un changement cosmétique qui évite d'avoir des dossiers vides.

Routes des ressources

Les paramètres des ressources seront maintenant au singulier par défaut. Autrement dit si vous avez cette ressource :

Route::resource('articles', 'ArticleController');

La route pour le **show** sera :

/articles/{article}

Au lieu de :

/articles/{articles}

Si vous voulez éviter ce comportement, par exemple pour une mise à niveau, il faut le préciser dans **AppServiceProvider** :

Route::singularResourceParameters(false);

Blade

Blade bénéficie d'une nouvelle variable **$loop** pour la directive **@foreach**. Cette variable pointe en fait une classe standard de PHP avec un lot de propriétés :

- **index** : un index pour les éléments (commence à 0)
- **iteration** : un index pour les éléments (commence à 1)
- **remaining** : le nombre d'éléments restants
- **count** : le nombre total d'éléments
- **first** : un booléen qui indique si c'est le premier élément
- **last** : un booléen qui indique si c'est le dernier élément
- **depth** : un entier qui indique la profondeur de la boucle
- **parent** : référence la variable $loop de l'éventuelle boucle englobante, sinon renvoie null

Faisons un petit essai avec une collection :

```
Route::get('/test', function () {
    $collection = collect([['a', 'b', 'c'], ['d', 'e', 'f'], ['g',
```

```
'h', 'i']]);
    return view('test', compact('collection'));
});
```

Et dans la vue :

```
@foreach($collection as $col)
    @foreach($col as $c)
        @if($loop->first)
             {{ 'Boucle de profondeur ' . $loop->depth . ' avec
parent itération ' . $loop->parent->iteration }}
        @endif
        <li>{{ $loop->iteration . '/' . $loop->count . ' et il en
reste ' . $loop->remaining . ' -> ' . $c }}</li>
    @endforeach
@endforeach
```

Ce qui donne :

Boucle de profondeur 2 avec parent itération 1
- 1/3 et il en reste 2 -> a
- 2/3 et il en reste 1 -> b
- 3/3 et il en reste 0 -> c

Boucle de profondeur 2 avec parent itération 2
- 1/3 et il en reste 2 -> d
- 2/3 et il en reste 1 -> e
- 3/3 et il en reste 0 -> f

Boucle de profondeur 2 avec parent itération 3
- 1/3 et il en reste 2 -> g
- 2/3 et il en reste 1 -> h
- 3/3 et il en reste 0 -> i

Je pense que ça va être bien pratique dans les vues !

Validation

Une nouveauté dans les validations : on pouvait déjà vérifier qu'on avait une image avec **image**. On va pouvoir maintenant ajouter des contraintes sur les dimensions de cette image avec la règle **dimensions** qui accepte ces contraintes :

- : largeur minimale en pixels
- **max_width** : largeur maximale en pixels
- **min_height** : hauteur minimale en pixels

- **max_height** : hauteur maximale en pixels
- **width** : largeur précise en pixels
- **height** : hauteur précise en pixels
- **ratio** : rapport largeur/hauteur

On va donc pouvoir écrire ce genre de règle :

```
'paysage' => 'dimensions:min_width=800,max_height=400'
```

Et pour le ratio :

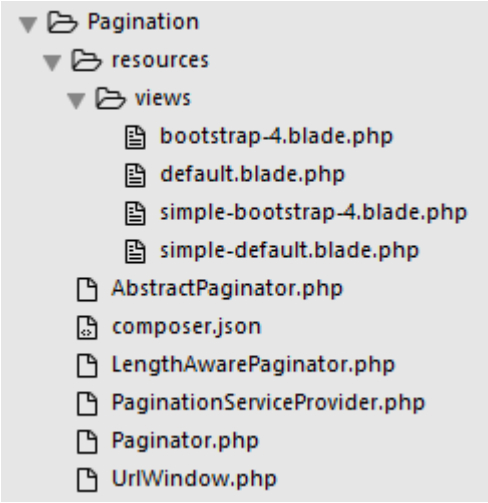
```
'paysage' => 'dimensions:ratio=3/2'
```

Bien pratique tout ça !

Pagination personnalisée

Si vous avez déjà essayé de personnaliser la pagination avec la version 5 vous devez savoir que c'est loin d'être facile mais on va avoir désormais une solution simple !

Regardez dans le framework la partie qui concerne la pagination :



On voit apparaître un dossier **resources/views** qui contient les gabarits pour la pagination. Au passage il est un peu surprenant de voir la référence à bootstrap 4 qui n'en est qu'à la version alpha…

Si on regarde le fichier **bootstrap-4.blade.php** on trouve la pagination classique avec bootstrap :

```
<ul class="pagination">
    <!-- Previous Page Link -->
    @if ($paginator->onFirstPage())
          <li class="page-item disabled"><span class="page-
link">«</span></li>
    @else
         <li class="page-item"><a class="page-link" href="{{
$paginator->previousPageUrl() }}" rel="prev">«</a></li>
    @endif

    <!-- Pagination Elements -->
    @foreach ($elements as $element)
        <!-- "Three Dots" Separator -->
        @if (is_string($element))
             <li class="page-item disabled"><span class="page-
link">{{ $element }}</span></li>
        @endif

        <!-- Array Of Links -->
        @if (is_array($element))
            @foreach ($element as $page => $url)
                @if ($page == $paginator->currentPage())
                        <li class="page-item active"><span
class="page-link">{{ $page }}</span></li>
                @else
                     <li class="page-item"><a class="page-link"
href="{{ $url }}">{{ $page }}</a></li>
                @endif
            @endforeach
        @endif
    @endforeach

    <!-- Next Page Link -->
    @if ($paginator->hasMorePages())
          <li class="page-item"><a class="page-link" href="{{
$paginator->nextPageUrl() }}" rel="next">»</a></li>
    @else
          <li class="page-item disabled"><span class="page-
link">»</span></li>
    @endif
</ul>
```

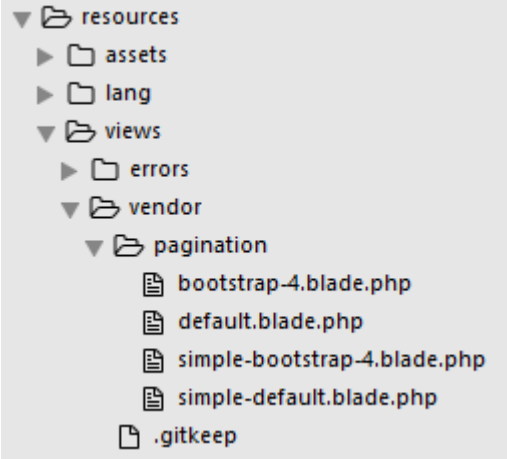
Comme ce fichier est dans le framework on ne va pas aller le

modifier ici !

Par contre on peut le publier :

```
λ php artisan vendor:publish --tag=laravel-pagination
Copied Directory [\vendor\laravel\framework\src\Illuminate\Pagination\resources\views] To [\resources\views\vendor\pagination]
Publishing complete for tag [laravel-pagination]!
```

Et on retrouve les fichiers dans les ressources de l'application :



Là on peut les modifier sans problème et le tour est joué !

Si vous ne voulez pas les publier il reste la solution de renseigner la vue lors de la pagination :

```
{{ $articles->links('view.mapagination') }}
```

Mais pourquoi se compliquer la vie ?

Query Builder

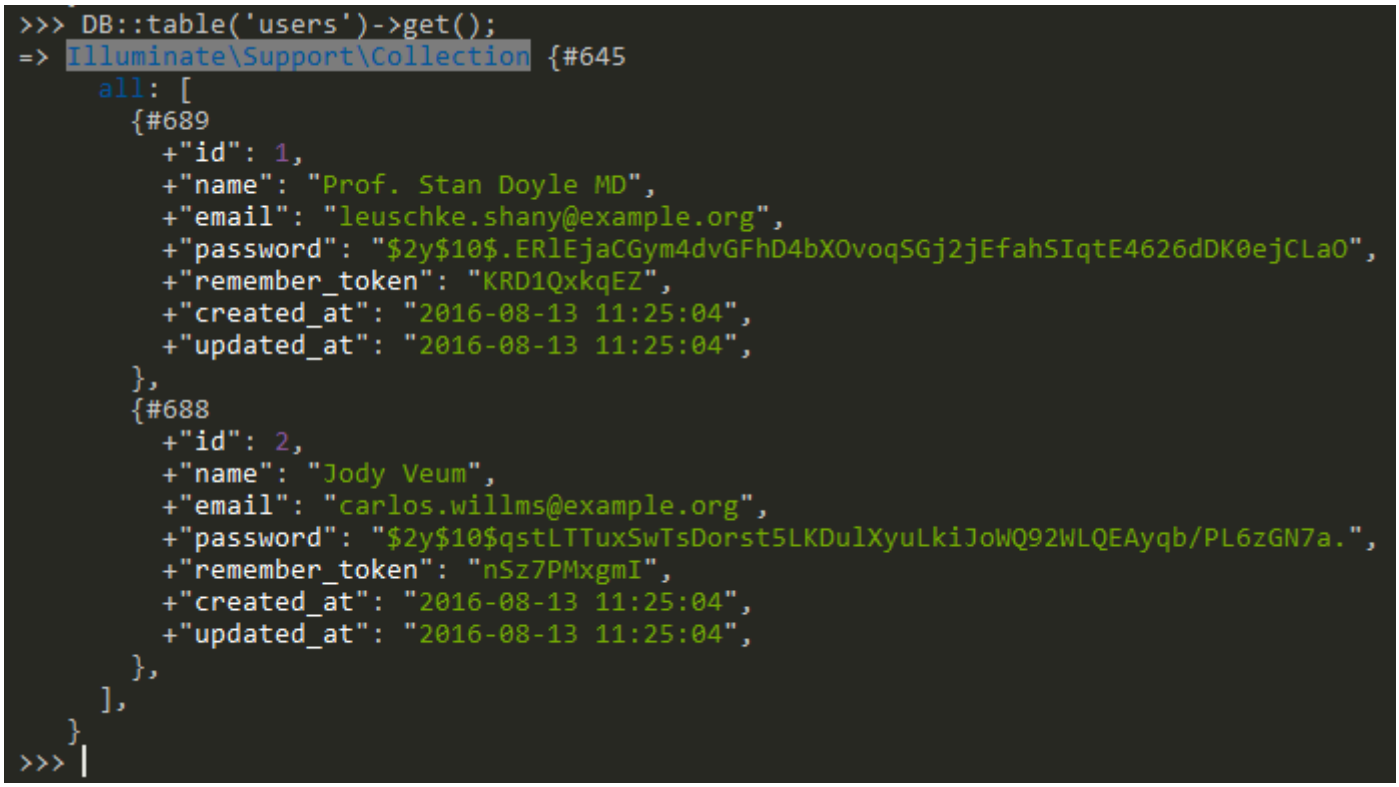
Dans les versions précédentes le query builder retourne un tableau dont chaque élément est une instance de **StdClass**. Ce qui permet d'écrire :

```
foreach ($articles as $article) {
    echo $article->title;
}
```

Désormais le query builder va retourner une instance de **Illuminate\Support\Collection**, donc une collection. On pourra toujours écrire le code ci-dessus mais on pourra aussi utiliser

toute la puissance des collections.

Si par exemple on a deux enregistrements dans la table users et qu'on les récupère on voit bien qu'on obtient une collection :



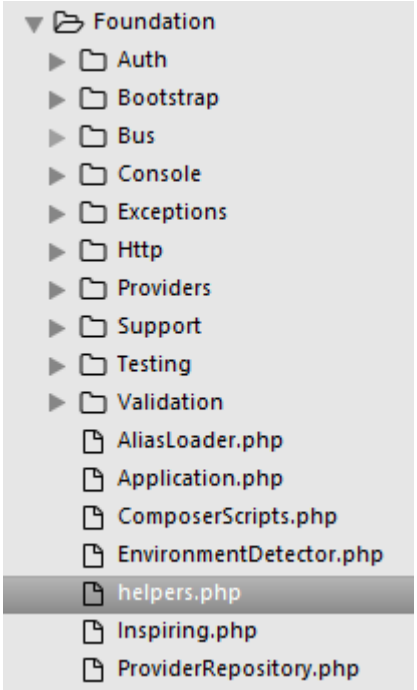
On peut toujours transformer la collection en tableau en utilisant **all()**, mais une collection est quand même plus sympathique et utile, d'autant que ça introduit une homogénéité avec les résultats récupérés par Eloquent.

Si on veut par exemple le premier enregistrement il suffit d'utiliser **first** :

Helpers

Au niveau des helpers on bénéficie d'un nouveau pour le cache. Pour mémoire j'ai rédigé sur ce blog un article concernant le cache mais évidemment sans utiliser l'helper qui n'existait pas encore !

Tous les helpers se trouvent dans le fichier **helpers.php** ici :



On y trouve désormais celui pour le cache :

```
if (! function_exists('cache')) {
    /**
     * Get / set the specified cache value.
     *
     * If an array is passed, we'll assume you want to put to the cache.
     *
     * @param  dynamic  key|key,default|data,expiration|null
     * @return mixed
     *
     * @throws \Exception
     */
    function cache()
    {
        $arguments = func_get_args();
```

```
        if (empty($arguments)) {
            return app('cache');
        }

        if (is_string($arguments[0])) {
                return app('cache')->get($arguments[0],
isset($arguments[1]) ? $arguments[1] : null);
        }

        if (is_array($arguments[0])) {
            if (! isset($arguments[1])) {
                throw new Exception(
                    'You must set an expiration time when putting
to the cache.'
                );
            }

                return app('cache')->put(key($arguments[0]),
reset($arguments[0]), $arguments[1]);
        }
    }
}
```

On retrouve toutes les possibillités du cache mais avec une syntaxe simplifiée :

```
>>> cache()->put('nom', 'Toto', 2)
=> null
>>> cache('nom');
=> "Toto"
>>>
```

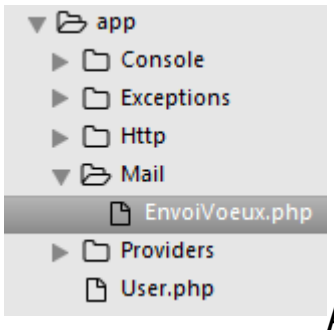
Pour ceux qui utilisent l'helper pour les sessions, ça fonctionne exactement pareil.

Mailable

Il y a aussi du changement pour l'envoi des email avec les classes « mailable » placée dans le dossier **app/Mail**. Pour respecter la cure d'amaigrissement ce dossier n'est pas prévu au départ et est généré dès la création de la première classe :

```
λ php artisan make:mail EnvoiVoeux
Mail created successfully.
```

On trouve la classe dans le dossier créé pour l'occasion :



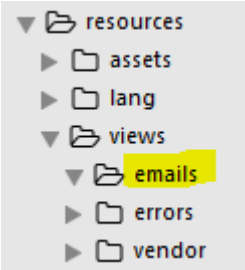
Avec ce code :

```
<?php

namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Mail\Mailable;
use Illuminate\Queue\SerializesModels;
use Illuminate\Contracts\Queue\ShouldQueue;

class EnvoiVoeux extends Mailable
{
    use Queueable, SerializesModels;

    /**
     * Create a new message instance.
     *
     * @return void
     */
    public function __construct()
    {
        //
    }

    /**
     * Build the message.
     *
     * @return $this
     */
    public function build()
    {
```

```
        return $this->view('view.name');
    }
}
```

Tout ce passe dans la méthode **build**. dans laquelle on va pouvoir utiliser **from**, **subject**, **attach** et **view** comme on le voit ci-dessus.

Peut-être qu'une bonne idée est de créer un dossier pour les vues des emails :



On va configurer la classe pour envoyer des voeux à nos amis, ça pourrait donner quelque chose comme ça :

```
<?php

namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Mail\Mailable;
use Illuminate\Queue\SerializesModels;
use Illuminate\Contracts\Queue\ShouldQueue;
use Services\Voeux;

class EnvoiVoeux extends Mailable
{
    use Queueable, SerializesModels;

    /**
     * The voeux instance.
     *
     * @var Voeux
     */
    public $voeux;

    /**
     * Create a new message instance.
     *
```

```
     * @return void
     */
    public function __construct(Voeux $voeux)
    {
        $this->voeux = $voeux;
    }

    /**
     * Build the message.
     *
     * @return $this
     */
    public function build()
    {
         return $this->from('cemoi@coucou.fr')
                ->view('emails.voeux')
                ->attach('/cartes/voeux.jpg');
    }
}
```

On aura automatiquement une variable **$voeux** dans la vue sans avoir besoin de le préciser.

Pour envoyer l'email on utilise la façade :

```
Mail::to($monami)
    ->cc($autreami)
    ->send(new EnvoiVoeux($voeux));
```

Et c'est parti !

Eloquent

Une petite modification : la méthode **save** désormais retourne **false** si le modèle n'a pas changé depuis le dernier accès, ce qui permet d'agir en conséquence.

Pour les tables pivot apparaît la nouvelle méthode **toggle** qui s'ajoute à **attach** et **detach**. Avec ces deux dernières méthodes il faut vérifier l'existence ou la non existence préalable avant de faire la mise à jour, avec **toggle** c'est automatique, on passe d'un état à un autre. Comme paramètre on peut transmettre des id, un

tableau ou une collection.

Notifications

Voilà encore une nouveauté, on va avoir un système complet de notification ! Le système de notification de Laravel permet d'envoyer des informations par différents canaux : mail, SMS (avec Nexmo), ou Slack.Il y a déjà pas mal de drivers, vous pouvez tous les trouver ici.

Mais les notifications peuvent aussi être stockées dans une base en attendant d'être affichées sur le client.

C'est un vaste sujet que je ne vais pas développer dans cet article.

Par exemple pour les mails il y a un template dynamique qui permet d'écrire ce genre de code :

```
$this->line('Merci de votre participation')
    ->action('En savoir plus', 'http://parla.com')
    ->success()
```

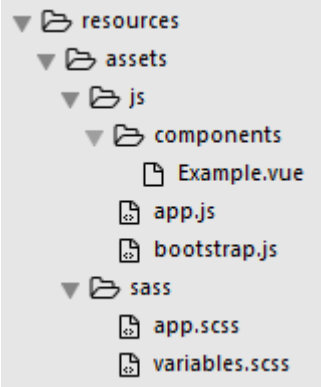
On aura dans le mail le texte et pour l'action un bouton. Si c'est pour une erreur on change l'aspect du bouton avec **error** :

```
$this->line('Il y a apparemment un problème !')
    ->action('En savoir plus', 'http://parla.com')
    ->error()
```

Plutôt sympathique !

FrontEnd

Il y a aussi du nouveau du côté du frontend avec ce qu'on va appeler une forte suggestion d'utiliser **vue.js** (si vous ne connaissez pas cette superbe librairie aller jeter un coup d'oeil sur ma série d'initiation). Si vous regardez les assets vous aller y trouver un peu plus de choses que dans la version précédente :



Et si vous regardez dans **gulpfile.js** :

```
elixir(mix => {
    mix.sass('app.scss')
       .webpack('app.js');
});
```

On voit l'utilisation de **webpack** (mais on pourrait aussi utiliser **Rollup**) pour la compilation du javascript à la mode **elixir** qui simplifie grandement les choses.

Pour faire fonctionner tout ça il faut commencer par installer les dépendances prévues dans **package.json** :

```
"devDependencies": {
  "bootstrap-sass": "^3.3.7",
  "gulp": "^3.9.1",
  "jquery": "^3.1.0",
  "laravel-elixir": "^6.0.0-9",
  "laravel-elixir-vue": "^0.1.4",
  "laravel-elixir-webpack-official": "^1.0.2",
  "lodash": "^4.14.0",
  "vue": "^1.0.26",
  "vue-resource": "^0.9.3"
}
```

On y trouve la version sass de **bootstrap**, mais aussi **jquery**, **elixir**, **vue.js** et son plugin pour les ressources. Bien entendu on n'est pas obligé d'utiliser tout ça, ni même de compiler nos assets mais on y est incité !

On va commencer par installer tout ça :

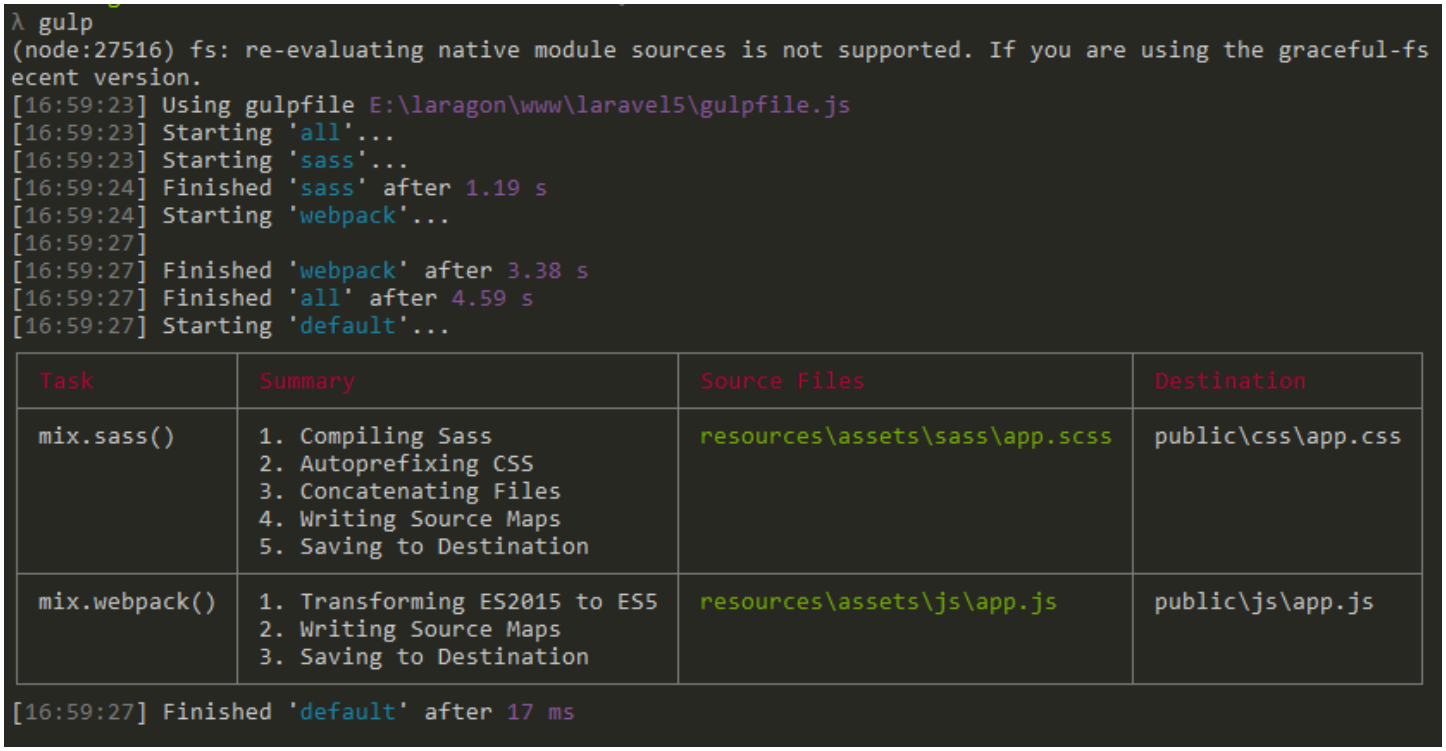
```
npm install
```

Il faut attendre un peu pour que le dossier **node_modules** se remplisse. Lorsque c'est fait il n'y a plus qu'à utiliser gulp :

gulp

Pour avoir aussi la minification il faudrait ajouter l'option **–production**.

Avec ce compte rendu des tâches :



```
λ gulp
(node:27516) fs: re-evaluating native module sources is not supported. If you are using the graceful-fs
ecent version.
[16:59:23] Using gulpfile E:\laragon\www\laravel5\gulpfile.js
[16:59:23] Starting 'all'...
[16:59:23] Starting 'sass'...
[16:59:24] Finished 'sass' after 1.19 s
[16:59:24] Starting 'webpack'...
[16:59:27]
[16:59:27] Finished 'webpack' after 3.38 s
[16:59:27] Finished 'all' after 4.59 s
[16:59:27] Starting 'default'...
```

Task	Summary	Source Files	Destination
mix.sass()	1. Compiling Sass 2. Autoprefixing CSS 3. Concatenating Files 4. Writing Source Maps 5. Saving to Destination	resources\assets\sass\app.scss	public\css\app.css
mix.webpack()	1. Transforming ES2015 to ES5 2. Writing Source Maps 3. Saving to Destination	resources\assets\js\app.js	public\js\app.js

```
[16:59:27] Finished 'default' after 17 ms
```

On voit que le css (bootstrap) est compilé dans **public/css/app.css**.

Pour le JavaScript tout va dans **public/js/app.js**. On a transformation de l'ES2015 en standard ES5. On a ici le JavaScript pour bootstrap et vue.js.

Tout est en place et on peut surveiller les changements avec :

gulp watch

Pour le moment tout ça ne sert à rien parce qu'aucune vue ne l'utilise…

Mais il suffit de mettre en oeuvre l'authentification pour avoir un layout qui l'utilise partiellement :

```
php artisan make:auth
```

Au passage vous remarquerez qu'il y a maintenant 4 contrôleurs plus légers pour l'authentification. D'autre part le verbe de la route du **logout** est passé de **GET** à **POST** pour être cohérent avec les normes. Donc attention aux mises à niveau !

Mais pour utiliser l'exemple de composant de vue.js il faut un peu de code…

On a un composant d'exemple qui se nomme example :

```
Vue.component('example', require('./components/Example.vue'));
```

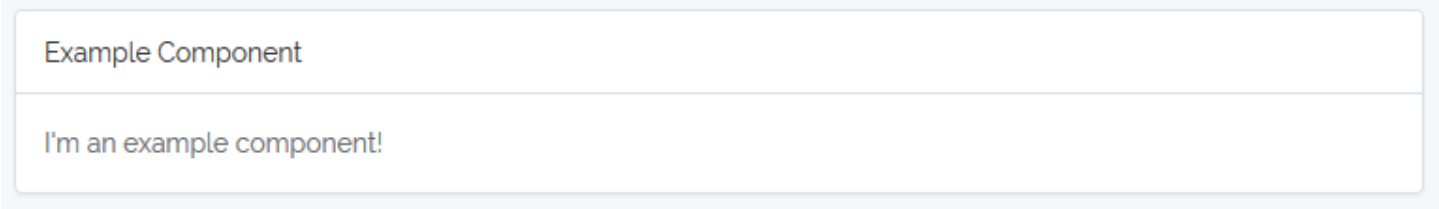
Puisqu'on a un layout qui charge le css et le JavaScript grâce à l'authentification on va faire une simple vue **test.blade.php** :

```
@extends('layouts.app')
<example></example>
```

On ajoute une route :

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

Et on voit appraître le composant au chargement :



Si vous voulez utiliser **vue.js** toute l'infrastructure est déjà en place…

Je n'ai pas fait le tour complet de tous les changements et je n'ai pas vraiment approfondi mais cet article vous donne au moins une idée des nouveautés de cette version qui va bientôt débarquer !