Intégrer AdminLTE

AdminLTE est sans doute le panneau de contrôle gratuit le plus utilisé pour construire la partie administration d'un site. En ce qui concerne Laravel il existe des packages pour l'intégrer comme celui-ci ou encore celui-là mais il semble aux questions sur les forums que ces approches ne soient pas très intuitives.

Alors je vous propose dans cet article une approche manuelle (il en existe bien d'autres possibles évidemment) et vous allez voir que ce n'est pas bien compliqué parce qu'après tout il s'agit juste d'une page HTML accompagnée de CSS et de Javascript…

Cet article se situe en marge de mon cours sur Laravel 5.5.

Installation de Laravel

On va partir d'un Laravel 5.5 tout neuf :

```
composer create-project laravel/laravel adminlte --prefer-dist
```

Et on va ajouter l'arsenal de l'authentification pour compléter :

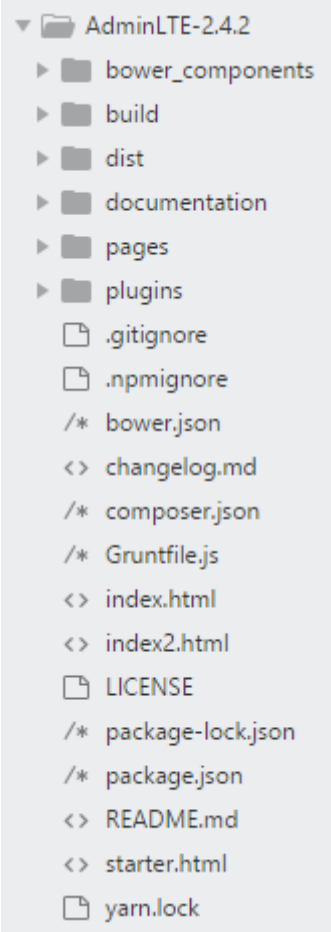
```
php artisan make:auth
```

Vérifiez que tout fonctionne correctement.

AdminLTE

On trouve AdminLTE sur ce site :

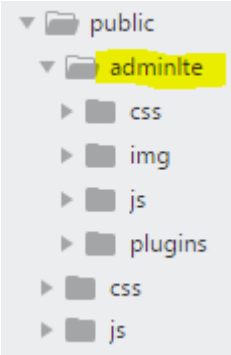
On va le télécharger avec le bouton **DOWNLOAD**. On se retrouve avec un fichier compressé et après décompression on obtient tout ça :



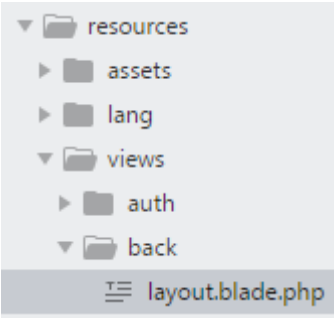
On va utiliser les éléments suivants :

- **dist** : la distribution pour les assets (css, img et js)
- **starter.html** : la page de base
- **plugins** : les plugins

On va commencer par copier les assets et plugins dans un nouveau dossier dans **public** :



On va prendre le code de la page **starter.html**, changer son nom pour **layout.blade.php** et ranger le fichier dans un dossier **back** des vues :



On a déjà bien avancé, on va maintenant intégrer tout ça…

Route et contrôleur

Routes

Pour les routes le mieux est de créer un groupe pour l'administration avec un préfixe **admin** et un espace de nom **Back** :

```
Route::prefix('admin')->namespace('Back')->group(function () {

    Route::name('admin')->get('/', 'AdminController@index');
```

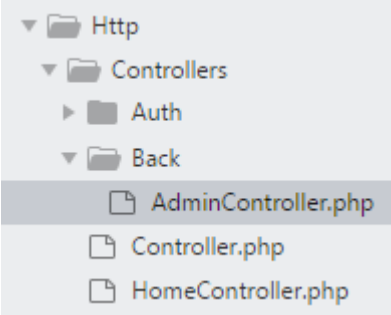
```
    ...

});
```

Évidemment dans votre application il faudra ajouter un middleware pour protéger l'administration.

Contrôleur

On va créer maintenant le contrôleur général de l'administration :

```
php artisan make:controller Back\AdminController
```



On va juste changer ainsi le code par défaut :

```
<?php

namespace App\Http\Controllers\Back;

use Illuminate\Http\Request;
use App\Http\Controllers\Controller;

class AdminController extends Controller
{
    public function index()
    {
        return view('back.index');
    }
}
```

Les vues

Le layout

Pour que notre layout fonctionne on va devoir un peu le bricoler.

Les assets

Déjà les références des assets ne sont pas bonnes, on va donc les changer :

```
...

<link                                          rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstra
p.min.css"                                  integrity="sha384-
BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
crossorigin="anonymous">
<!-- Font Awesome -->
<link                                          rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/font-
awesome.min.css">
<!-- Ionicons -->
<link                                          rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/ionicons/2.0.1/css/io
nicons.min.css">
<!-- Theme style -->
<link rel="stylesheet" href="adminlte/css/AdminLTE.min.css">
<!-- AdminLTE Skins. We have chosen the skin-blue for this starter
      page. However, you can choose any other skin. Make sure you
       apply the skin class to the body tag so the changes take
effect. -->
<link   rel="stylesheet"   href="adminlte/css/skins/skin-
blue.min.css">

...

  <!-- The user image in the navbar-->
  <img  src="adminlte/img/user2-160x160.jpg"  class="user-image"
alt="User Image">
 <!-- hidden-xs hides the username on small devices so only the
image appears. -->
 <span class="hidden-xs">Alexander Pierce</span>
</a>
<ul class="dropdown-menu">
```

```
 <!-- The user image in the menu -->
 <li class="user-header">
  <img  src="adminlte/img/user2-160x160.jpg"  class="img-circle"
alt="User Image">

...

<div class="pull-left image">
  <img  src="adminlte/img/user2-160x160.jpg"  class="img-circle"
alt="User Image">
</div>

...
<!-- jQuery 3 -->
<script
src="https://code.jquery.com/jquery-3.2.0.min.js"></script>
<!-- Bootstrap 3.3.7 -->
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.
min.js"                                  integrity="sha384-
Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa"
crossorigin="anonymous"></script>
<!-- AdminLTE App -->
<script src="adminlte/js/adminlte.min.js"></script>
```

Les sections

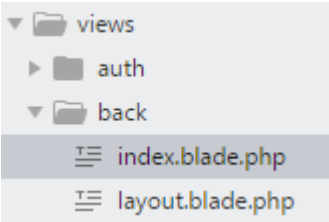
D’autre part on va prévoir les sections nécessaires pour ensuite construire nos pages :

```
<!-- AdminLTE Skins. -->
<link   rel="stylesheet"   href="/adminlte/css/skins/skin-
blue.min.css">

@yield('css')

...

<!-- Main content -->
<section class="content container-fluid">
  @yield('main')
</section>
```

```
...

<!-- AdminLTE App -->
<script src="adminlte/js/adminlte.min.js"></script>

@yield('js')
```

La page d’accueil

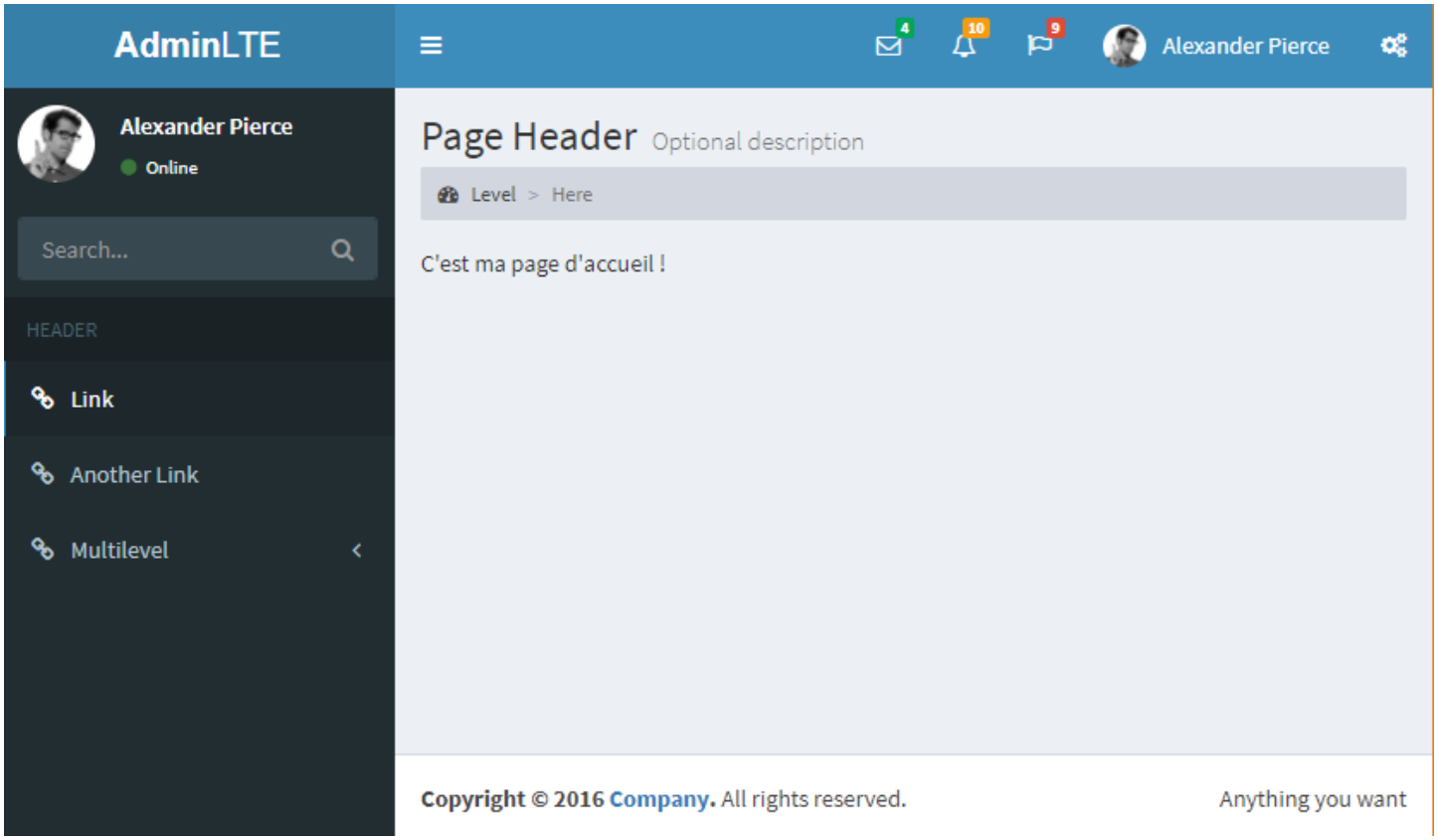
Maintenant qu’on a un layout créons une page d’accueil pour notre administration :



Avec ce simple code :

```
@extends('back.layout')

@section('main')
    C'est ma page d'accueil !
@endsection
```

Normalement avec l’url …/admin vous devriez obtenir ceci :



Le plus gros a été fait, maintenant il va falloir penser aux détails. Qu'est-ce que vous allez garder ou ajouter ? Comment gérer le menu, le breadcrumb, les titres, les images…

Le menu latéral

Gestion dans la vue

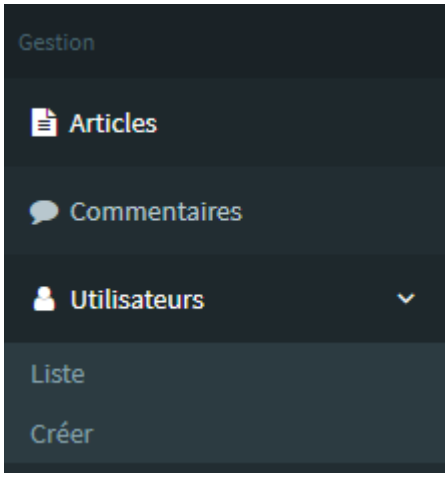
Le menu latéral est géré par ce code :

```
<!-- Sidebar Menu -->
<ul class="sidebar-menu" data-widget="tree">
  <li class="header">HEADER</li>
  <!-- Optionally, you can add icons to the links -->
   <li class="active"><a href="#"><i class="fa fa-link"></i>
<span>Link</span></a></li>
   <li><a href="#"><i class="fa fa-link"></i> <span>Another
Link</span></a></li>
  <li class="treeview">
    <a href="#"><i class="fa fa-link"></i> <span>Multilevel</span>
      <span class="pull-right-container">
          <i class="fa fa-angle-left pull-right"></i>
```

```
        </span>
    </a>
    <ul class="treeview-menu">
      <li><a href="#">Link in level 2</a></li>
      <li><a href="#">Link in level 2</a></li>
    </ul>
  </li>
</ul>
<!-- /.sidebar-menu -->
```

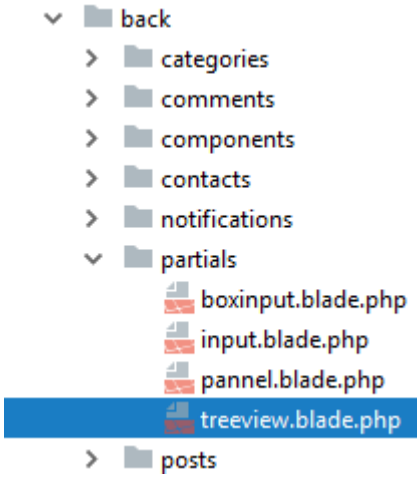
Vous pouvez directement construire le menu dans ce code, par exemple :

```
<!-- Sidebar Menu -->
<ul class="sidebar-menu" data-widget="tree">
  <li class="header">Gestion</li>
  <!-- Optionally, you can add icons to the links -->
   <li class="active"><a href="#"><i class="fa fa-fw fa-file-
text"></i> <span>Articles</span></a></li>
    <li><a  href="#"><i  class="fa  fa-fw  fa-comment"></i>
<span>Commentaires</span></a></li>
  <li class="treeview">
        <a  href="#"><i  class="fa  fa-fw  fa-user"></i>
<span>Utilisateurs</span>
      <span class="pull-right-container">
          <i class="fa fa-angle-left pull-right"></i>
        </span>
    </a>
    <ul class="treeview-menu">
      <li><a href="#">Liste</a></li>
      <li><a href="#">Créer</a></li>
    </ul>
  </li>
</ul>
<!-- /.sidebar-menu -->
```



Inclusion d’une vue partielle

Dans mon application d’exemple j’ai opté pour une vue partielle :



Avec ce code :

```
<li class="treeview">
      <a href="#"><i class="fa fa-fw fa-{{ $icon }}"></i>
<span>@lang('admin.menu.' . $type . 's')</span>
        <span class="pull-right-container">
            <span class="fa fa-angle-left pull-right"></span>
        </span>
    </a>
    <ul class="treeview-menu">
        @foreach ($items as $item)
            <li><a href="{{ $item['route'] }}"><span class="fa fa-
fw fa-circle-o text-{{ $item['color'] }}"></span>
<span>@lang('admin.menu.' . $item['command'])</span></a></li>
        @endforeach
    </ul>
```

```
</li>
```

Du coup au niveau du layout on appelle la vue partielle en transmettant les informations. Par exemple pour les articles :

```
@include('back.partials.treeview', [
  'icon' => 'file-text',
  'type' => 'post',
  'items' => [
    [
      'route' => route('posts.index'),
      'command' => 'list',
      'color' => 'blue',
    ],
    [
      'route' => route('posts.index', ['new' => 'on']),
      'command' => 'new',
      'color' => 'yellow',
    ],
    [
      'route' => route('posts.create'),
      'command' => 'create',
      'color' => 'green',
    ],
  ],
])
```

C’est un choix d’organisation, ça rend le layout plus lisible et plus facile à modifier.

Une approche encore plus globale consisterait à créer une configuration et à l’utiliser comme source d’information et on pourrait imaginer une interface de gestion, mais ça se justifierait uniquement si le menu devait évoluer souvent.

Les images des utilisateurs

Les images des utilisateurs ajoutent une note esthétique mais vous pouvez les supprimer. Pour leur gestion vous avez le choix. Soit vous permettez que les administrateurs gèrent leur profil et puissent ajouter une image à leur goût, soit opter pour un fonctionnement plus automatisé. C’est cette deuxième option que

j’ai choisie pour l’application d’exemple avec le Gravatar.

J’ai opté pour un package pour gérer cet aspect.

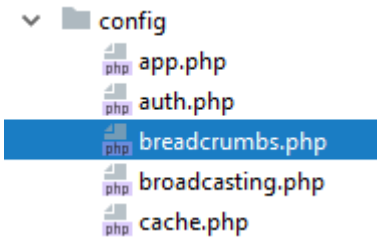
Une fois installé tout est automatisé et dans le layout ça devient tout simple :

```
<img src="{{ Gravatar::get(auth()->user()->email) }}" class="img-circle" alt="User Image">
```

Le breadcrumb

Pour le breadcrumb c’est un peu plus délicat à gérer. Vous pouvez utiliser ce package. Franchement il ne m’a personnellement pas convaincu et j’ai préféré coder cette partie.

J’ai créé une configuration pour stocker les informations :

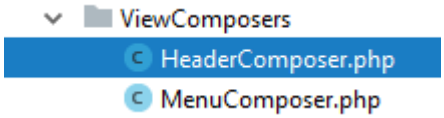


Avec ce code :

```
return [

    /*
    |--------------------------------------------------------------------------
    | Names, icons and urls for urls segments
    |--------------------------------------------------------------------------
    |
    | Set names, icons and urls for each admin url segment
    */

    'admin' => [
        'name' => 'dashboard',
        'icon' => 'dashboard',
        'url' => '/admin',
```

```
    ],
    'posts' =>
    [
        'name' => 'posts',
        'icon' => 'file-text',
        'url' => '/admin/posts',
    ],

   ...
```

On dispose là des noms, des icônes et des urls. Pour envoyer ces informations j'utilise un composeur de vue (pour centraliser le code) :



Voilà la partie de code pour le breadcrumb :

```
public function compose(View $view)
{
    // Breadcrumb
    $elements = config ('breadcrumbs');
    $segments = request()->segments();

    foreach ($segments as $segment) {
        if (!is_numeric($segment)) {
            $elements[$segment]['name'] = __('admin.breadcrumbs.'
. $elements[$segment]['name'] . '-name');
            if($segment === end($segments)) {
                $elements[$segment]['url'] = '#';
            }
            $breadcrumbs[] = $elements[$segment];
        }
    }

   ...

$view->with(compact('breadcrumbs', 'title', 'countNotifications'))
;
}
```

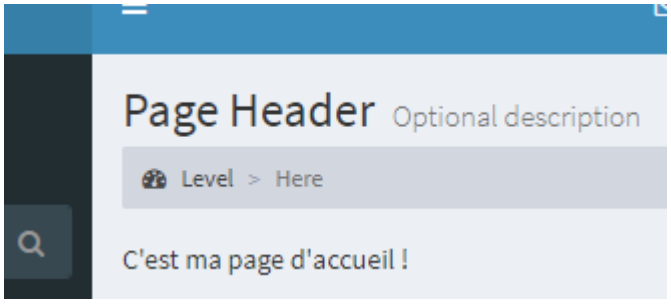
Et dans la vue on utilise la variable **$breadcrumbs** transmise :

```
<ol class="breadcrumb">
  @foreach ($breadcrumbs as $item)
    <li @if ($loop->last && $item['url'] === '#') class="active"
@endif>
      @if ($item['url'] !== '#')
        <a href="{{ $item['url'] }}">
      @endif
      @isset($item['icon'])
        <span class="fa fa-{{ $item['icon'] }}"></span>
      @endisset
      {{ $item['name'] }}
      @if ($item['url'] !== '#')
        </a>
      @endif
    </li>
  @endforeach
</ol>
```

C’est simple et efficace et on peut se passer du package un peu lourd et pas vraiment intuitif. En plus je n’arrivais pas à faire ce que je voulais avec lui…

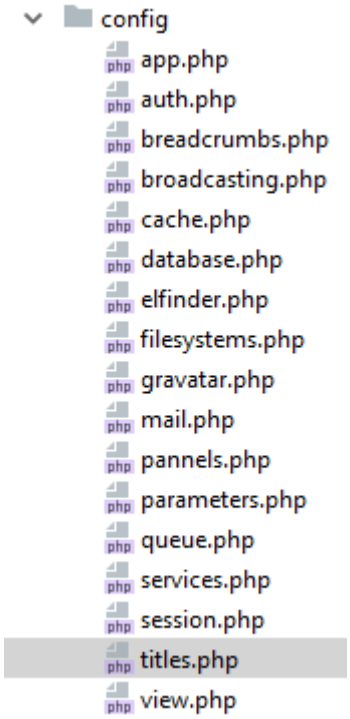
Les titres

Chaque page comporte un titre et une description optionnelle :



Vous pouvez pour chaque vue envoyer un titre et une description et gérer ça dans chaque méthode de vos contrôleurs. mais il est plus efficace de centraliser le traitement.

Dans l’application d’exemple j’ai créé une configuration :



Avec ce code :

```
return [

    /*
    |--------------------------------------------------------------------------
    | Titles for routes names
    |--------------------------------------------------------------------------
    |
    | Set Titles for each admin routes names
    */

    'admin' => 'dashboard',
    'users' => [
        'index' => 'usersGestion',
        'edit' => 'userEdit',
    ],
    'contacts' => [
        'index' => 'contactsGestion',
    ],

   ...
```

On ne trouve pas les noms réels parce qu’il faut tenir compte de la localisation.

Dans le même composeur de vues qu'on a vu ci-dessus pour le breadcrumb on trouve le code pour la gestion des titres :

```
public function compose(View $view)
{
    ...

    // Title
    $title = config('titles.' . Route::currentRouteName());
    $title = __('admin.titles.' . $title);

    ...

          $view->with(compact('breadcrumbs',  'title',
'countNotifications'));
}
```

C'est ici que les noms des titres récupèrent leur texte selon la locale.

Et dans la layout évidemment ça devient simple :

```
{{ $title }}
```

Les notifications

Pour les notifications dans l'application d'exemple je suis aussi passé par le composeur de vues :

```
public function compose(View $view)
{
    ...

    // Notifications
                                $countNotifications       =
auth()->user()->unreadNotifications()->count();

          $view->with(compact('breadcrumbs',  'title',
'countNotifications'));
}
```

On transmet le compte des notifications éventuelles dans la variable **$countNotifications**. Dans le layout on a :

```
<!-- Notifications Menu -->
@if ($countNotifications)
  <li class="dropdown notifications-menu">
    <!-- Menu toggle button -->
    <a href="#" class="dropdown-toggle" data-toggle="dropdown">
      <i class="fa fa-bell-o"></i>

         <span class="label label-warning">{{ $countNotifications
}}</span>

    </a>
    <ul class="dropdown-menu">
      <li class="header">@lang('New notifications')</li>
      <li>
        <!-- Inner Menu: contains the notifications -->
        <ul class="menu">
          <li><!-- start notification -->
            <a href="#">
                  <i class="fa fa-users text-aqua"></i> {{
$countNotifications       }}       @lang('new')       {{
trans_choice(__('comment|comments'), $countNotifications) }}
            </a>
          </li>
          <!-- end notification -->
        </ul>
      </li>
       <li class="footer"><a href="{{ route('notifications.index',
[auth()->id()]) }}">@lang('View')</a></li>
    </ul>
  </li>
@endif
```

On affiche l'icône et nombre de notifications s'il y en a :

En conclusion

Il y aurait encore à dire sur la question mais vous avez là une bonne base de départ et après les solutions évoluent beaucoup par rapport aux différents besoins et il faut traiter au cas par cas…