Laravel 5.7 par la pratique – Les langues

Dans ce chapitre on va s'intéresser à l'aspect multi-langage. Pour le moment notre galerie est en français mais on a fait en sorte que les textes soient faciles à traduire en utilisant dans le code les helpers de Laravel. On va donc ajouter maintenant l'anglais à notre galerie. Ça ne concernera évidemment que l'interface et pas les données, ce qui serait une autre histoire…

La configuration

Dans le fichier **config/app.php** on a des réglages pour les langues :

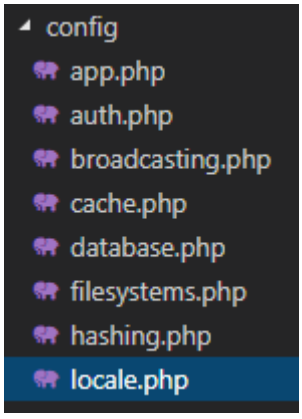
```
'locale' => 'fr',

'fallback_locale' => 'en',
```

On a fixé la locale au français (**fr**) et la langue par défaut en cas d'absence de traduction à l'anglais (**en**).

On va ajouter un réglage avec les langues qu'on va mettre en œuvre et qui devront avoir les traductions présentes :

```
'locales' => ['fr', 'en',],
```

On va ajouter un fichier de configuration pour les locales :



Ce fichier est issu de [ce dépôt Github](). Il permet d'avoir par pays

le code pour **Carbon** et celui pour **setLocale()**.

Route, contrôleur et middleware

Route

On ajoute la route pour le changement de la langue :

```
Route::name('language')->get('language/{lang}', 'HomeController@language');
```

Contrôleur

Et on ajoute la fonction dans **HomeController** :

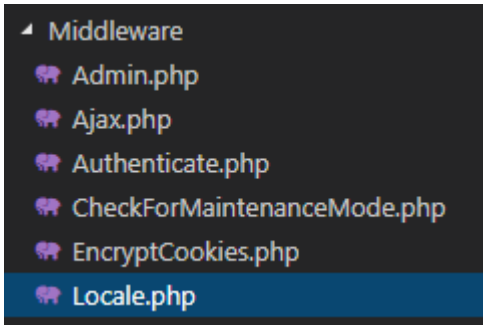
```
public function language(String $locale)
{
    $locale = in_array($locale, config('app.locales')) ? $locale : config('app.fallback_locale');

    session(['locale' => $locale]);

    return back();
}
```

On reçoit une locale en paramètre. Si elle est présente dans le tableau de la configuration on fixe cette locale en session, sinon on se rabat sur la locale par défaut.

Middleware

Il nous faut maintenant un middleware qui va vérifier si on a une locale en session pour réellement l’affecter. Si ce n’est pas le cas essayer de définir la langue de l’utilisateur. On va aussi fixer la locale pour les dates.

```
php artisan make:middleware Locale
```



Et on code ainsi :

```
public function handle($request, Closure $next)
{
    if (!session ()->has ('locale')) {
          session (['locale' => $request->getPreferredLanguage
(config ('app.locales'))]);
    }
    $locale = session ('locale');
    app ()->setLocale ($locale);
     setlocale (LC_TIME, app()->environment('local') ? $locale :
config('locale.languages')[$locale][1]);
    return $next ($request);
}
```

Et on le référence dans **app/Http/Kernel** :

```
protected $middlewareGroups = [
    'web' => [
        ...

        \App\Http\Middleware\Locale::class,
        \App\Http\Middleware\Settings::class,
    ],

    ...
];
```

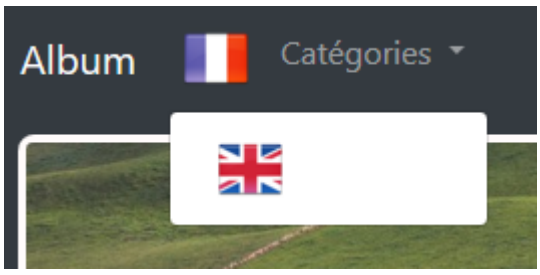
Le menu

On va ajouter un menu déroulant pour le choix de la locale dans **views/layouts/app** :

```
<ul class="navbar-nav mr-auto">
    <li class="nav-item dropdown">
```

```
          <a class="nav-link" href="#" id="navbarDropdownFlag"
role="button" data-toggle="dropdown"
            aria-haspopup="true" aria-expanded="false">
            <img width="32" height="32" alt="{{ session('locale')
}}"
                          src="{!! asset('images/flags/' .
session('locale') . '-flag.png') !!}"/>
        </a>
            <div id="flags" class="dropdown-menu" aria-
labelledby="navbarDropdownFlag">
            @foreach(config('app.locales') as $locale)
                @if($locale != session('locale'))
                        <a class="dropdown-item" href="{{
route('language', $locale) }}">
                          <img width="32" height="32" alt="{{
session('locale') }}"
                                    src="{!! asset('images/flags/' .
$locale . '-flag.png') !!}"/>
                    </a>
                @endif
            @endforeach
        </div>
    </li>
```

Et on va voir le résultat :



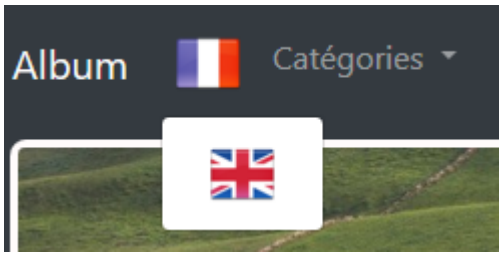
Je ne trouve pas trop élégant la largueur de la zone pour le drapeau.On va arranger ça dans **resources/sass/_variables.scss**. On va en profiter pour changer la typographie et un peu la pagination :

```
// Body
$body-bg: #343a40;

// Menu
$dropdown-min-width: 5rem;
```

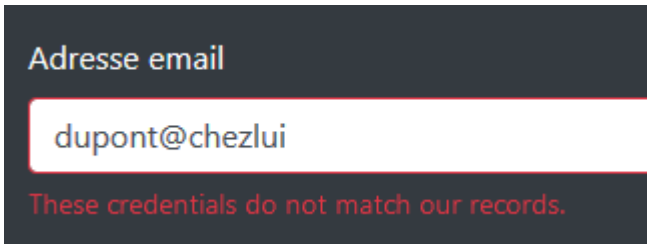
```
// Typography
$Raleway", sans-serif;
$font-size-base: 0.9rem;
$line-height-base: 1.6;
$text-color: #636b6f;

// Card
$card-border-width: 4px;
$card-border-radius: .3rem;
$card-border-color: rgba(255, 242, 242, .3);

// Pagination
$pagination-active-bg: grey;
$pagination-active-border-color: grey;
```

On lance npm…

Et c’est maintenant plus équilibré :



Pour le moment le changement de langue ne se voit que dans les validations :



Un package

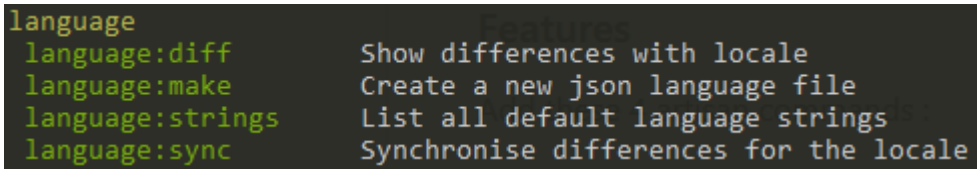
Il nous faut créer un fichier JSON avec toutes les traductions pour l’anglais. On pourrait faire ça en explorant tout le code avec des copier/coller, ça serait vraiment laborieux !

On va plutôt utiliser un package pour nous aider. Comme je n’en ai

pas vraiment trouvé un qui me plaise j'en ai créé un. On va commencer par l'installer :

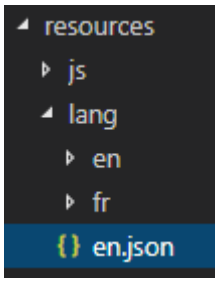
composer require bestmomo/laravel5-artisan-language --dev

On a maintenant 4 commandes de plus dans artisan :



On va utiliser la deuxième :

php artisan language:make en

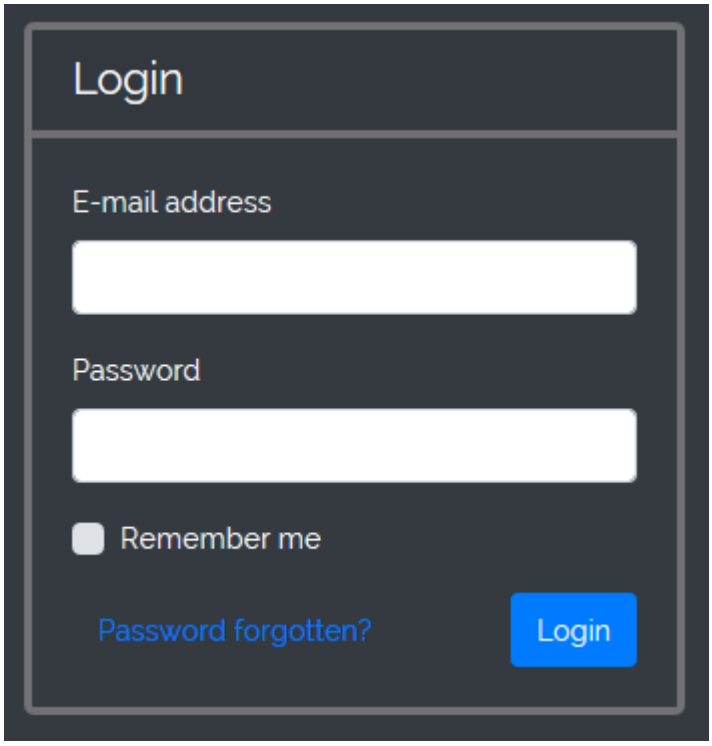


Le fichier JSON a été créé et on a tous les textes par ordre alphabétique qui attendent leur traduction :

```
{
    "A l'url :": "",
    "A propos": "",
    "Administration": "",
    "Adresse email": "",
    "Adresse web :": "",
    "Adulte": ""

    ...
```

On ajoute donc les traductions. Comme le fichier est assez gros vous pouvez le récupérer sur github.

Maintenant si on passe à l'anglais on a bien les textes dans cette langue :



Vous pourrez aussi remarquer que les dates sont maintenant correctes.

C’est le dernier chapitre de cette série. Il reste quelques éléments dont je n’ai pas parlé, comme les pages d’information, mais que vous pouvez retrouver dans le dépôt Github.

Conclusion

Dans ce chapitre on a :

- prévu la configuration pour les locales
- ajouté la route, la fonction du contrôleur et un middleware pour le changement de locale
- ajouté un menu déroulant avec les drapeaux des langues disponibles
- installé un package pour créer le fichier de la nouvelle langue et ajouté ainsi les traductions