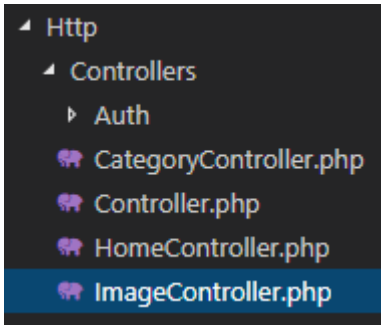
Laravel 5.7 par la pratique – Les images

Notre galerie avance bien. On a désormais des catégories pour classer les photos. On va maintenant voir comment on va ajouter des photos. Pour le faire un utilisateur doit être enregistré ou bien administrateur, et évidemment avec un email vérifié. On va donc créer de nouvelles routes, un contrôleur, un repository pour ranger le code de gestion des données, une vue…

Le contrôleur

Pour gérer les images on va créer un contrôleur :

```
php artisan make:controller ImageController --resource
```



Le fait d'utiliser l'option **–resource** a généré les 7 méthodes de base. On va conserver seulement **create**, **store**, **update** et **destroy**.

Les routes

Pour les routes on va ajouter ça :

```
Route::middleware ('auth', 'verified')->group (function () {
    Route::resource ('image', 'ImageController', [
        'only' => ['create', 'store', 'destroy', 'update']
    ]);
});
```

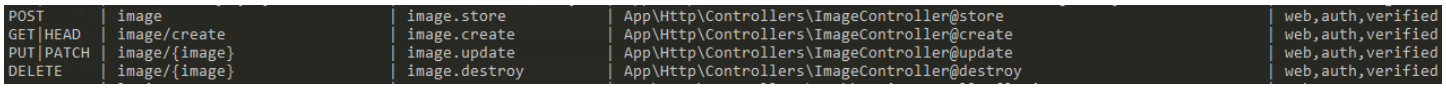
remarquez l'utilisation des middleware :

- **auth** : utilisateur authentifié
- **verified** : email vérifié

Le groupe nous servira plus tard quand on ajoutera d'autres routes.

On vérifie :

php artisan route:list



```
POST      | image                | image.store          | App\Http\Controllers\ImageController@store                | web,auth,verified
GET|HEAD  | image/create         | image.create         | App\Http\Controllers\ImageController@create               | web,auth,verified
PUT|PATCH | image/{image}        | image.update         | App\Http\Controllers\ImageController@update               | web,auth,verified
DELETE    | image/{image}        | image.destroy        | App\Http\Controllers\ImageController@destroy              | web,auth,verified
```

On a bien nos 3 routes pour notre contrôleur.

Le menu

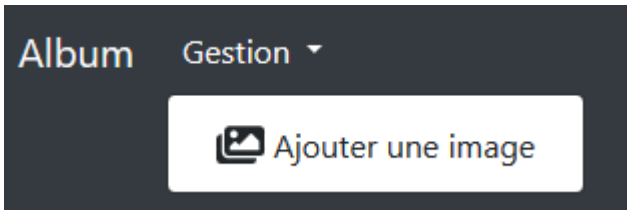
Pour accéder au formulaire d'ajout d'une image on va devoir compléter la barre de navigation dans **views/layouts/app** :

```
@endadmin
@auth
    <li class="nav-item dropdown">
                    <a  class="nav-link  dropdown-toggle{{
currentRoute(route('image.create'))}}"
        href="#" id="navbarDropdownGestAlbum" role="button" data-
toggle="dropdown"
        aria-haspopup="true" aria-expanded="false">
            @lang('Gestion')
        </a>
                        <div  class="dropdown-menu"  aria-
labelledby="navbarDropdownGestAlbum">
                         <a  class="dropdown-item"  href="{{
route('image.create') }}">
                <i class="fas fa-images fa-lg"></i> @lang('Ajouter
une image')
            </a>
        </div>
    </li>
@endauth
```

On utilise la directive **@auth** de Blade pour réserver l'item de menu aux utilisateurs authentifiés. On utilise un menu déroulant

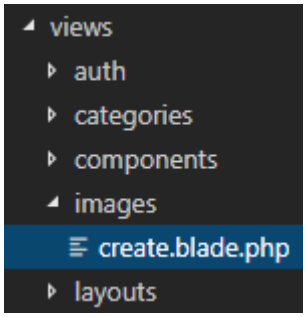
pour anticiper la gestion des albums.

Maintenant un utilisateur authentifié voit ça (un administrateur a en plus le menu d'administration) :



La vue de création

On crée un dossier pour les images et une vue pour la création :



Avec ce code :

```
@extends('layouts.form')

@section('card')

    @component('components.card')

        @slot('title')
            @lang('Ajouter une image')
        @endslot

         <form method="POST" action="{{ route('image.store') }}"
enctype="multipart/form-data">
            @csrf

            <div class="form-group{{ $errors->has('image') ? ' is-
invalid' : '' }}">
                <div class="custom-file">
                    <input type="file" id="image" name="image"
                           class="{{ $errors->has('image') ? ' is-
```

```
invalid ' : '' }}custom-file-input" required>
                        <label class="custom-file-label"
for="image"></label>
                    @if ($errors->has('image'))
                        <div class="invalid-feedback">
                            {{ $errors->first('image') }}
                        </div>
                    @endif
                </div>
                <br>
            </div>

            <div class="form-group">
                  <img id="preview" class="img-fluid" src="#"
alt="">
            </div>

            <div class="form-group">
                                                        <label
for="category_id">@lang('Catégorie')</label>
                  <select id="category_id" name="category_id"
class="form-control">
                    @foreach($categories as $category)
                          <option value="{{ $category->id }}">{{
$category->name }}</option>
                    @endforeach
                </select>
            </div>

            @include('partials.form-group', [
                'title' => __('Description (optionnelle)'),
                'type' => 'text',
                'name' => 'description',
                'required' => false,
                ])

            <div class="form-group">
                <div class="custom-control custom-checkbox">
                     <input type="checkbox" class="custom-control-
input" id="adult" name="adult">
                        <label class="custom-control-label"
for="adult"> @lang('Contenu adulte')</label>
                </div>
```

```
            </div>

            @component('components.button')
                @lang('Envoyer')
            @endcomponent

        </form>

    @endcomponent

@endsection

@section('script')

    <script>
        $(() => {
            $('input[type="file"]').on('change', (e) => {
                let that = e.currentTarget
                if (that.files && that.files[0]) {
                                    $(that).next('.custom-file-
label').html(that.files[0].name)
                    let reader = new FileReader()
                    reader.onload = (e) => {
                        $('#preview').attr('src', e.target.result)
                    }
                    reader.readAsDataURL(that.files[0])
                }
            })
        })
    </script>

@endsection
```

On utilise le contrôle File Browser de Bootstrap 4. On a une liste de choix pour les catégories et un simple contrôle de texte pour la description optionnelle.

On prévoit une visualisation de l’image avant envoi. On va voir le détail plus loin…

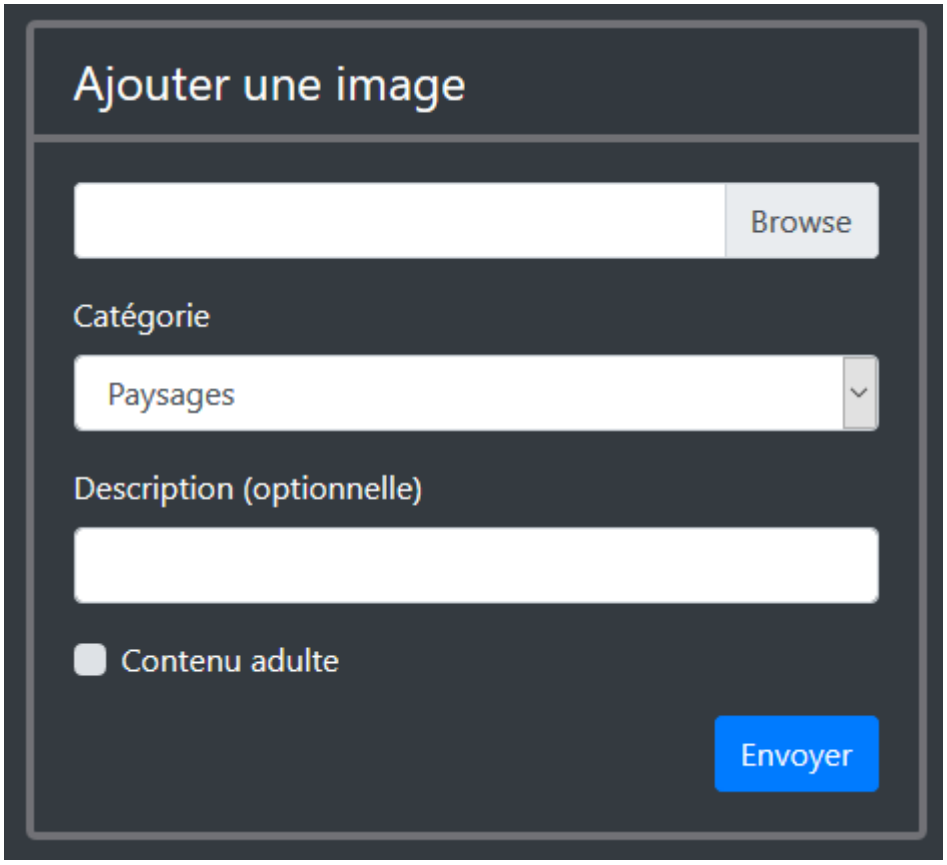
L’affichage du formulaire

Il nous faut maintenant coder la gestion de tout ça dans le contrôleur **ImageController**.

Déjà il faut afficher le formulaire :

```
public function create()
{
    return view('images.create');
}
```

On vérifie avec le menu que ça marche :



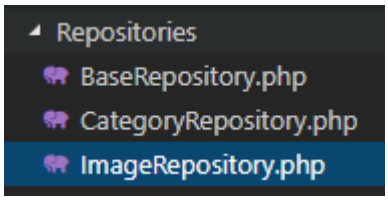
On va juste modifier un peu le CSS dans **ressources/assets/app.css** pour mettre le bouton en français :

```
.custom-file-label::after {
    content: "Parcourir";
}
```

On lance **npm run dev** pour régénérer :

Le repository

Comme on va avoir pas mal de code pour la gestion des images on va créer un repository :



Avec ce code pour le moment :

```
<?php

namespace App\Repositories;

use App\Models\Image;

class ImageRepository
{

}
```

Et on déclare le repository dans le contrôleur **ImageController** :

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Repositories\ImageRepository;

class ImageController extends Controller
{
    protected $repository;

    public function __construct(ImageRepository $repository)
    {
```

```
        $this->repository = $repository;
    }

...
```

Les disques

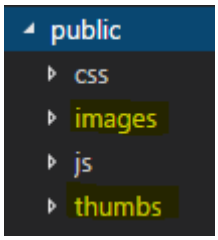
Le système de fichier de Laravel est géré dans **config/filesystems.php** avec ce code par défaut pour les disques :

```
'disks' => [

    'local' => [
        'driver' => 'local',
        'root' => storage_path('app'),
    ],

    'public' => [
        'driver' => 'local',
        'root' => public_path(),
        'visibility' => 'public',
    ],

    's3' => [
        'driver' => 's3',
        'key' => env('AWS_ACCESS_KEY_ID'),
        'secret' => env('AWS_SECRET_ACCESS_KEY'),
        'region' => env('AWS_DEFAULT_REGION'),
        'bucket' => env('AWS_BUCKET'),
        'url' => env('AWS_URL'),
    ],

],
```

Pour chaque image ajoutée on va avoir deux versions :

- une image en haute résolution (mais on limitera quand même la taille à 2MO)
- une image en basse résolution (thumb) pour l’affichage dans les vignettes (on va fixer arbitrairement la largeur à 500 pixels).

Comme les images doivent être accessibles on va créer deux dossiers dans **public** (si vous préférez les simlinks libre à vous d'utiliser le dossier **storage**) :



Comme je veux les fichiers dans **public** et non pas dans le **storage** il faut adapter la configuration en conséquence :

```
'default' => env('FILESYSTEM_DRIVER', 'public'),

...

'disks' => [

    ...
    'public' => [
        'driver' => 'local',
        'root' => public_path(),
        'visibility' => 'public',
    ],

    ...

],
```

On met **public** par défaut pour simplifier la syntaxe.

Manipuler des images

Pour manipuler les images, en particulier créer la version basse résolution on va faire appel au superbe package [Intervention Image](#) :

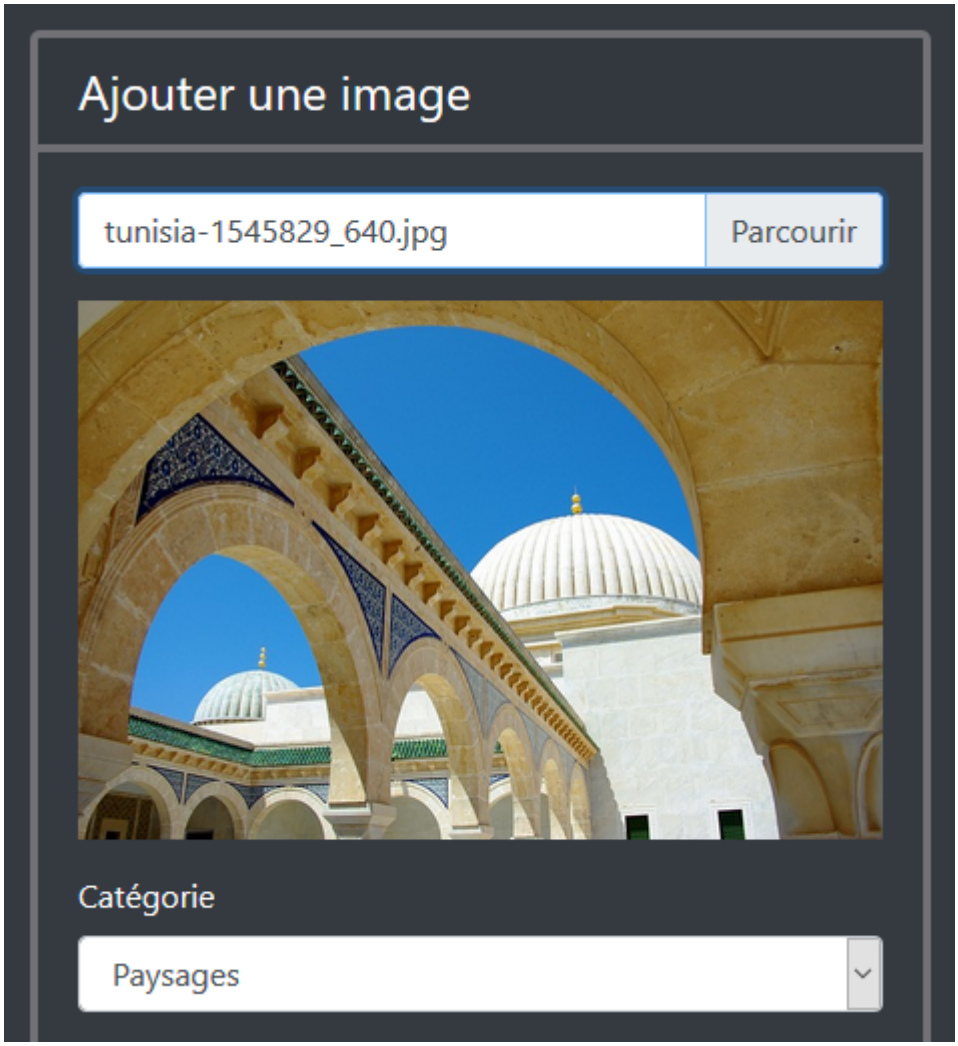
```
composer require intervention/image
```

On va ajouter la référence dans notre **repository**, ainsi que celle du **storage** :

```
<?php

namespace App\Repositories;

use App\Models\Image;
use Illuminate\Support\Facades\Storage;
use Intervention\Image\Facades\Image as InterventionImage;

class ImageRepository
```

L’envoi d’une image

Lorsqu’on sélectionne une image à partir du formulaire on commence par la visualiser :



On réalise ça à l’aide de l’objet FileReader dans ce code Javascript :

```
$(() => {
```

```
    $('input[type="file"]').on('change', (e) => {
        let that = e.currentTarget
        if (that.files && that.files[0]) {
                                $(that).next('.custom-file-
label').html(that.files[0].name)
            let reader = new FileReader()
            reader.onload = (e) => {
                $('#preview').attr('src', e.target.result)
            }
            reader.readAsDataURL(that.files[0])
        }
    })
})
```

A la soumission on arrive dans la méthode **store** du contrôleur. On va la coder ainsi :

```
public function store(Request $request)
{
    $request->validate([
        'image' => 'required|image|max:2000',
        'category_id' => 'required|exists:categories,id',
        'description' => 'nullable|string|max:255',
    ]);

    $this->repository->store($request);

      return back()->with('ok', __("L'image a bien été
enregistrée"));
}
```

On a la validation et ensuite on fait appel au **repository** pour la sauvegarde. Enfin on renvoie la même page (**back**) avec un message de succès.

C’est dans le **repository** qu’on a toute l’intendance :

```
public function store($request)
{
    // Save image
    $path = basename ($request->image->store('images'));

    // Save thumb
     $image = InterventionImage::make ($request->image)->widen
```

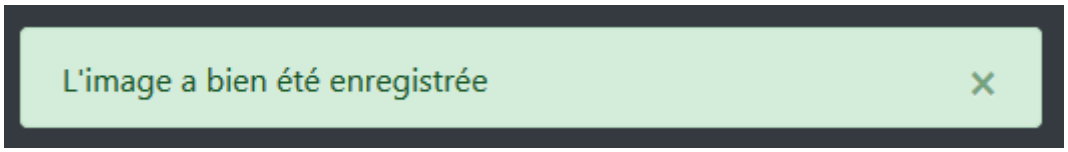
```
(500)->encode ();
    Storage::put ('thumbs/' . $path, $image);

    // Save in base
    $image = new Image;
    $image->description = $request->description;
    $image->category_id = $request->category_id;
    $image->adult = isset($request->adult);
    $image->name = $path;
    $request->user()->images()->save($image);
}
```

On voit que **Intervention Image** nous aide bien ! D'autre par le système de fichiers de Laravel offre une syntaxe concise.

Normalement ça devrait fonctionner. Faite un essai de chargement d'une image et vérifiez dans la table :

id	category_id	user_id	name	description	adult	clicks	created_at	updated_at
1	1	2	hCAlh1eJfEv16gq8ZdSlvPBqHallkHbOBKqpB05Z.jpeg	*NULL*	0	0	2018-09-18 12:48:20	2018-09-18 12:48:20

Et d'affichage du message :



Conclusion

Dans ce chapitre on a :

- complété la barre de navigation
- créé routes, contrôleur la création d'une image
- créé un repository pour les images
- installé le package **Intervention Image**
- écrit tous le code pour le chargement d'une image

Pour vous simplifier la vie vous pouvez charger le projet dans son état à l'issue de ce chapitre. J'ai ajouté un **seeder** pour les images ainsi que toute la collection d'images dans public. Donc si vous faites une migration avec la population vous aurez toutes les

images prêtes, ce qui va nous être utile pour la suite de cette série.