Laravel 5.7 par la pratique – Les catégories 2/2

Dans le précédant chapitre on a commencé à voir la gestion des catégories pour notre galerie photos. On sait maintenant ajouter une catégorie. Maintenant on va voir comment modifier et supprimer une catégorie. C'est encore réservé aux administrateurs évidemment. On va créer deux vues : une qui liste toutes les catégories avec des boutons pour modifier et supprimer, et une pour le formulaire de modification.

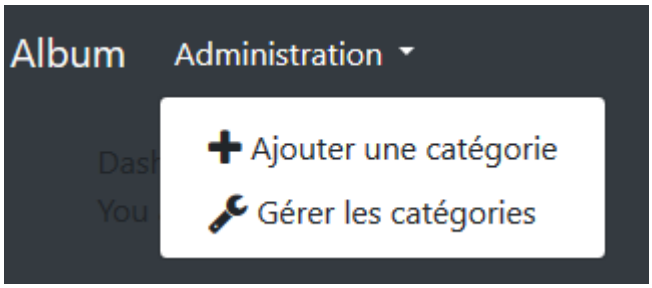
Le menu

On va compléter le menu pour qu'on puisse accéder aux nouvelles vues. Dans dans notre layout (views/layouts/app) dans la partie concernant le menu déroulant pour les administrateurs on va avoir ce code :

```
@admin
<li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle{{ currentRoute(
                            route('category.create'),
                            route('category.index'),
                                        route('category.edit',
request()->category?: 0)
                        )}}" href="#" id="navbarDropdownGestCat"
role="button" data-toggle="dropdown"
        aria-haspopup="true" aria-expanded="false">
        @lang('Administration')
    </a>
                    <div    class="dropdown-menu"    aria-
labelledby="navbarDropdownGestCat">
        <a class="dropdown-item" href="{{ route('category.create')
}}">
            <i class="fas fa-plus fa-lg"></i> @lang('Ajouter une
catégorie')
        </a>
        <a class="dropdown-item" href="{{ route('category.index')
}}">
```

```
            <i class="fas fa-wrench fa-lg"></i> @lang('Gérer les
catégories')
        </a>
    </div>
</li>
@endadmin
```

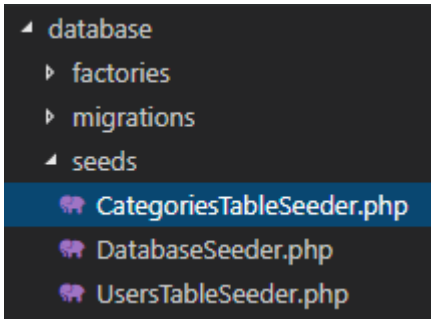
Avec ce résultat :



On crée quelques catégories

Pour avoir un peu de matériel on va créer des catégories. On crée un seeder :

```
php artisan make:seeder CategoriesTableSeeder
```



Avec ce code :

```
<?php

use Illuminate\Database\Seeder;
use App\Models\Category;

class CategoriesTableSeeder extends Seeder
{

    public function run()
    {
```

```
        Category::create([
            'name' => 'Paysages',
        ]);
        Category::create([
            'name' => 'Maisons',
        ]);
        Category::create([
            'name' => 'Personnages',
        ]);
        Category::create([
            'name' => 'Animaux',
        ]);
        Category::create([
            'name' => 'Végétation',
        ]);
    }
}
```

On complète le **DatabaseSeeder** :

```
public function run()
{
    $this->call(UsersTableSeeder::class);
    $this->call(CategoriesTableSeeder::class);
}
```

Et on rafraîchit la base :

```
php artisan migrate:fresh --seed
```

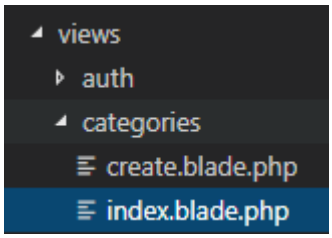
Si tout se passe bien vous devez avoir les 5 catégories :

id	name	slug
1	Paysages	paysages
2	Maisons	maisons
3	Personnages	personnages
4	Animaux	animaux
5	Végétation	vegetation

*Si on vous dit qu'une classe n'existe pas vous pouvez lancer un **composer dumpautoload**.*

La liste des catégories

On va créer maintenant la vue pour lister les catégories et afficher des boutons de commande. On crée donc cette vue ici :

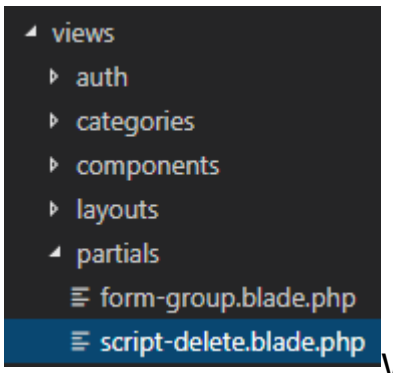


Avec ce code :

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

@section('card')

    @component('components.card')

        @slot('title')
            @lang('Gestion des catégories')
        @endslot


        <table class="table table-dark text-white">
            <tbody>
            @foreach($categories as $category)
                <tr>
                    <td>{{ $category->name }}</td>
                    <td>
                                    <a type="button" href="{{
route('category.destroy', $category->id) }}"
                                class="btn btn-danger btn-sm pull-right
invisible" data-toggle="tooltip"
                                title="@lang('Supprimer la catégorie')
{{ $category->name }}"><i
                                        class="fas fa-trash fa-
lg"></i></a>
                                    <a type="button" href="{{
route('category.edit', $category->id) }}"
                                 class="btn btn-warning btn-sm pull-
right mr-2 invisible" data-toggle="tooltip"
```

```
                                title="@lang('Modifier la catégorie')
{{ $category->name }}"><i
                                                class="fas fa-edit fa-
lg"></i></a>
                        </td>
                    </tr>
                @endforeach
                </tbody>
            </table>

        @endcomponent

@endsection

@section('script')

    <script>
        $(() => {
            $('a').removeClass('invisible')
        })
    </script>

     @include('partials.script-delete', ['text' => __('Vraiment
supprimer cette catégorie ?'), 'return' => 'removeTr'])

@endsection
```

On voit que pour la suppression d’une catégorie on fait appel à une vue partielle parce que le code sera utilisé pour une autre vue :



Voici son code :

```
<script>
    $(() => {
```

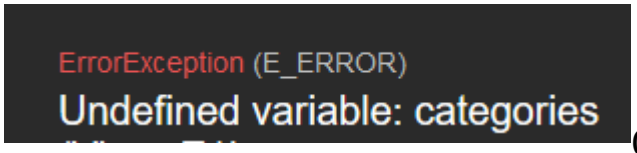
```
        $.ajaxSetup({
              headers: {'X-CSRF-TOKEN': $('meta[name="csrf-
token"]').attr('content')}
        })

        $('[data-toggle="tooltip"]').tooltip()

        $('a.btn-danger').click((e) => {
            let that = $(e.currentTarget)
            e.preventDefault()
            swal({
                title: '{{ $text }}',
                type: 'error',
                showCancelButton: true,
                confirmButtonColor: '#DD6B55',
                confirmButtonText: '@lang('Oui')',
                cancelButtonText: '@lang('Non')'
            }).then((result) => {
                if (result.value) {
                    $.ajax({
                        url: that.attr('href'),
                        type: 'DELETE'
                    })
                    .done(() => {
                        @switch($return)
                            @case('removeTr')
                                that.parents('tr').remove()
                                @break
                            @case('reload')
                                location.reload()
                                @break
                        @endswitch
                    })
                    .fail(() => {
                        swal({
                             title: '@lang('Il semble y avoir une
erreur sur le serveur, veuillez réessayer plus tard...')',
                            type: 'warning'
                        })
                    })
                }
            })
        })
```

```
    })
</script>
```

Pour activer ces vues on va utiliser la fonction **index** du contrôleur **CategoryController** :

```
public function index()
{
    return view('categories.index');
}
```

Comme la vue attend une variable **$categories** et que là on ne lui donne pas on va avoir un problème !



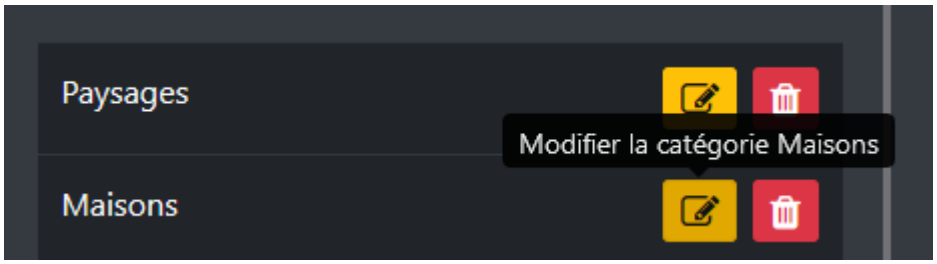
Comme cette variable sera nécessaire pour toutes les vues on va utiliser ce code dans **App\Providers\AppServiceProvider** :

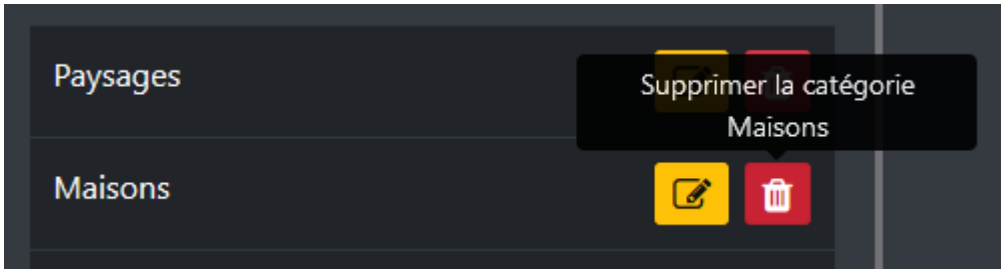
```
use App\Repositories\CategoryRepository;

...

public function boot()
{
    ...

    if (request ()->server ("SCRIPT_NAME") !== 'artisan') {
                        view   ()->share   ('categories',
resolve(CategoryRepository::class)->getAll());
    }
}
```

On teste qu'on est pas avec une commande d'Artisan pour éviter certains conflits. La méthode **share** permet de partager la variable **$categories** avec toutes les vues. La méthode **resolve** permet de demander au conteneur de créer une classe **CategoryRepository** pour pouvoir charger la variable avec toutes les catégories. Remarquez que la méthode **getAll** est en fait dans **BaseRepository**.

Normalement en cliquant maintenant dans le menu vous devez obtenir la liste :

Vérifiez que les popups fonctionnent (la documentation est ici) :





C’est d’ailleurs tout ce qui fonctionne pour le moment !

Supprimer une catégorie

Pour la suppression d’une catégorie j’ai prévu une alerte pour éviter une suppression accidentelle. Plutôt que d’utiliser l’horrible fenêtre de base de Javascript on va utiliser Sweet

Alert.On va l'installer avec **npm** :

```
npm i sweetalert2 -D
```

Il faut le charger dans **resources/sass/app.scss** :

```
// Sweetalert
@import '~sweetalert2/src/sweetalert2.scss';
```

Et pour la partie Javascript dans **resources/js/app.js** :

```
window.swal = require('sweetalert2');
```

Enfin on relance **npm run dev**.

Maintenant quand on clique sur un bouton de suppression on a l'alerte esthétique :

Si on clique sur **Non** ça se referme et rien ne se passe.

Si on clique sur Oui on se rend compte que la catégorie est retirée de la liste mais évidemment la base n'est pas modifiée puisqu'on a pas encore écrit le code correspondant. On obtient aucune erreur parce qu'on a une fonction actuellement vide dans le contrôleur. Si on supprime cette fonction par contre on va avoir une erreur signalée :

On complète le code dans le contrôleur :

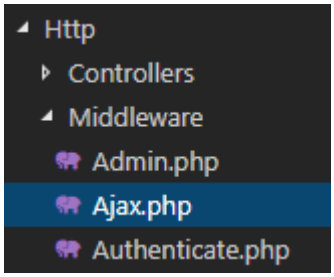
```
public function destroy(Category $category)
{
    $category->delete();

    return response()->json();
}
```

Remarquez la liaison implicite avec le modèle au niveau du paramètre. Maintenant une suppression va être effective.

Mais il serait sans doute judicieux de prévoir un filtre pour être sûr que seules des requêtes Ajax effectuent cette action.

Créons un middleware :

```
php artisan make:middleware Ajax
```



Avec ce code :

```
<?php
```

```
namespace App\Http\Middleware;

use Closure;

class Ajax
{
    public function handle($request, Closure $next)
    {
        if ($request->ajax()) {
            return $next($request);
        }

        abort(404);
    }
}
```

On l’ajoute dans le **Kernel** :

```
protected $routeMiddleware = [
    ...
    'ajax' => \App\Http\Middleware\Ajax::class,
];
```

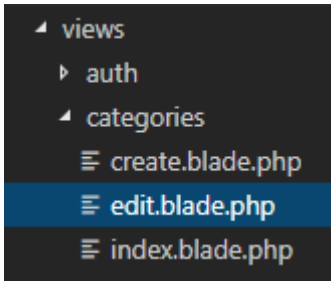
Il n’y a plus qu’à l’ajouter dans le contrôleur **CategoryController** :

```
public function __construct()
{
    $this->middleware('ajax')->only('destroy');
}
```

Évidemment que pour la méthode **destroy**.

Modifier une catégorie

On crée le formulaire pour la modification qui est pratiquement identique à celui pour la création :



Avec ce code :

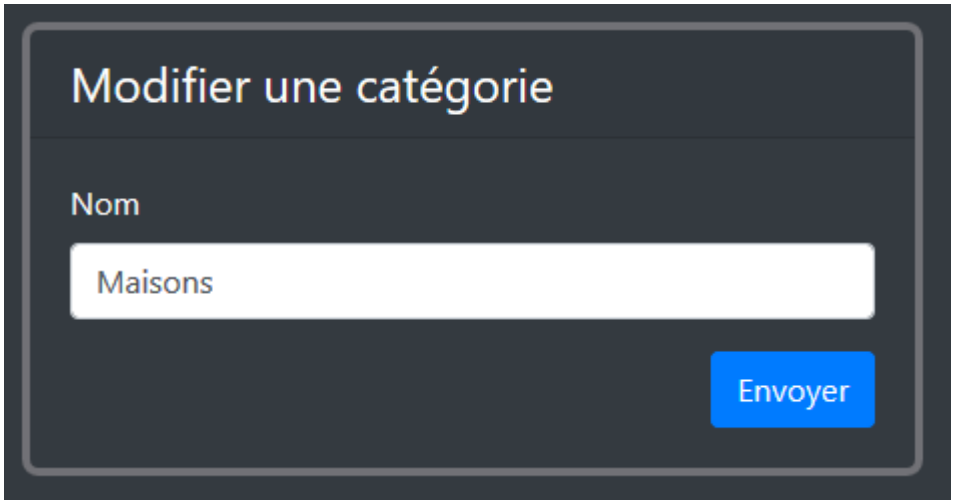
```
@extends('layouts.form')

@section('card')

    @component('components.card')

        @slot('title')
            @lang('Modifier une catégorie')
        @endslot

         <form method="POST" action="{{ route('category.update',
$category->id) }}">
            {{ csrf_field() }}
            {{ method_field('PUT') }}

            @include('partials.form-group', [
                'title' => __('Nom'),
                'type' => 'text',
                'name' => 'name',
                'value' => $category->name,
                'required' => true,
                ])

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

        </form>

    @endcomponent

@endsection
```

On utilise la fonction **edit** du contrôleur **CategoryController** :

```
public function edit(Category $category)
```

```
{
    return view('categories.edit', compact('category'));
}
```

Maintenant quand on clique sur un bouton de modification dans la liste on a bien le formulaire :

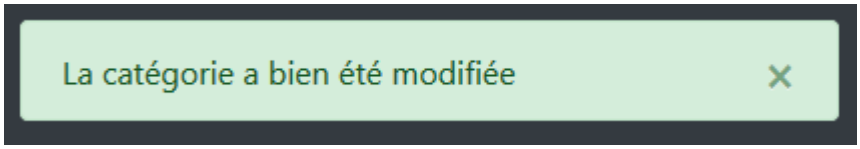


On utilise maintenant la fonction **update** dans le contrôleur :

```
public function update(CategoryRequest $request, Category $category)
{
    $category->update($request->all());

    return redirect()->route('home')->with('ok', __('La catégorie a bien été modifiée'));
}
```

On a dans le précédent chapitre mis en place un événement pour le **slug** qui sera aussi actif dans la modification, on a donc pas à nous en inquiéter ici.

Quand la catégorie a été modifiée on a une alerte (c'est le même code que celui qu'on a vu pour la création au niveau du layout) :



Conclusion

Dans ce chapitre on a :

- modifié le menu de la barre de navigation pour ajouter un item pour l'administration
- créé un seeder pour les catégories
- créé une vue pour lister les catégories avec des boutons pour la modification et la suppression
- ajouté Sweet Alert 2 à l'application
- complété le contrôleur CategoryController pour la gestion des catégories
- créé un middleware pour Ajax
- partagé les données des catégories pour toutes les vues

Pour vous simplifier la vie vous pouvez charger le projet dans son état à l'issue de ce chapitre.