Créer une application avec Laravel 5.5 – 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.

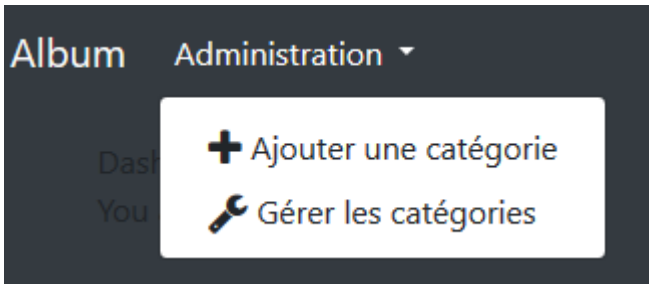
Le menu

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)
         )}}" 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 :

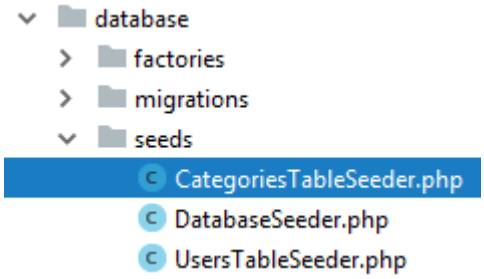


On a maintenant un item dans le menu…

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 :

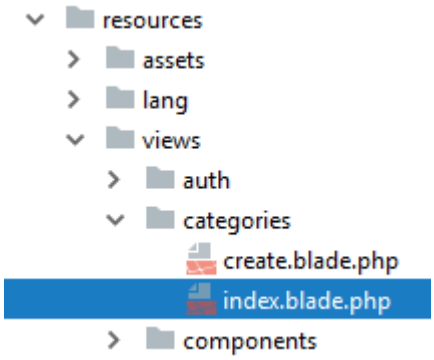
```
php artisan migrate:fresh --seed
```

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

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">
            <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" 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" 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>
        $(function() {

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

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

            $('a.btn-danger').click(function(e) {
                let that = $(this)
                e.preventDefault()
                swal({
                      title: '@lang('Vraiment supprimer cette
catégorie ?')',
                    type: 'warning',
                    showCancelButton: true,
                    confirmButtonColor: '#DD6B55',
                    confirmButtonText: '@lang('Oui')',
                    cancelButtonText: '@lang('Non')'
                }).then(function () {
                    $('[data-toggle="tooltip"]').tooltip('hide')
                    $.ajax({
                        url: that.attr('href'),
                        type: 'DELETE'
                    })
                        .done(function () {
                            that.parents('tr').remove()
                        })
                        .fail(function () {
```

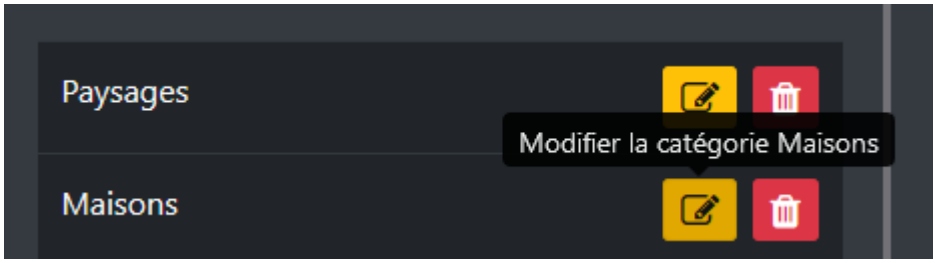
```
                                swal({
                                     title: '@lang('Il semble y avoir
une erreur sur le serveur, veuillez réessayer plus tard...')',
                                     type: 'warning'
                                })
                            }
                        )
                    })
                })
            })
        </script>

@endsection
```

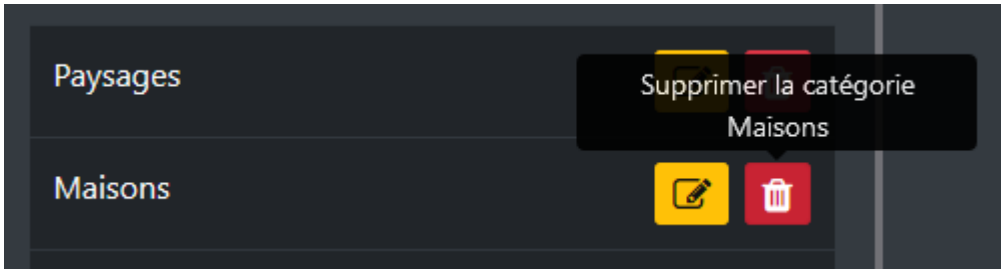
Pour l'activer on va compléter la fonction **index** du contrôleur **CategoryController** :

```
public function index()
{
    $categories = Category::all();

    return view('categories.index', compact ('categories'));
}
```

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

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

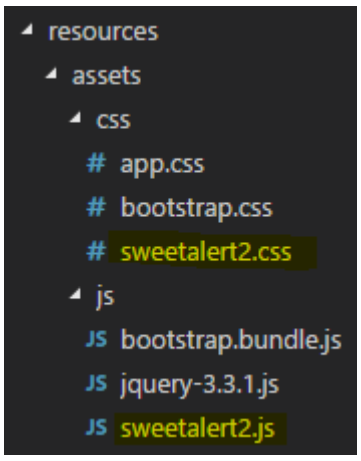




C’est d’ailleurs tout ce qui fonctionne pour le moment !

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. Il nous faut le CSS et le JS :



On met à jour **webpack.mix.js** :

```
mix.styles([
    'resources/assets/css/bootstrap.css',
    'resources/assets/css/app.css',
    'resources/assets/css/sweetalert2.css'
], 'public/css/app.css')
```

```
.scripts([
    'resources/assets/js/jquery-3.3.1.js',
    'resources/assets/js/bootstrap.bundle.js',
    'resources/assets/js/sweetalert2.js'
], 'public/js/app.js');
```

Et on relance npm (ou alors on le met en watch)…

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 :

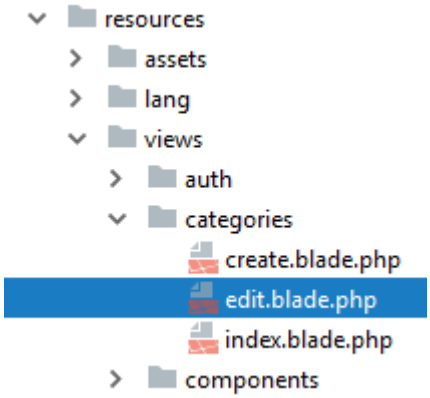
On complète le code dans le contrôleur :

```
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.

Modifier une catégorie

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



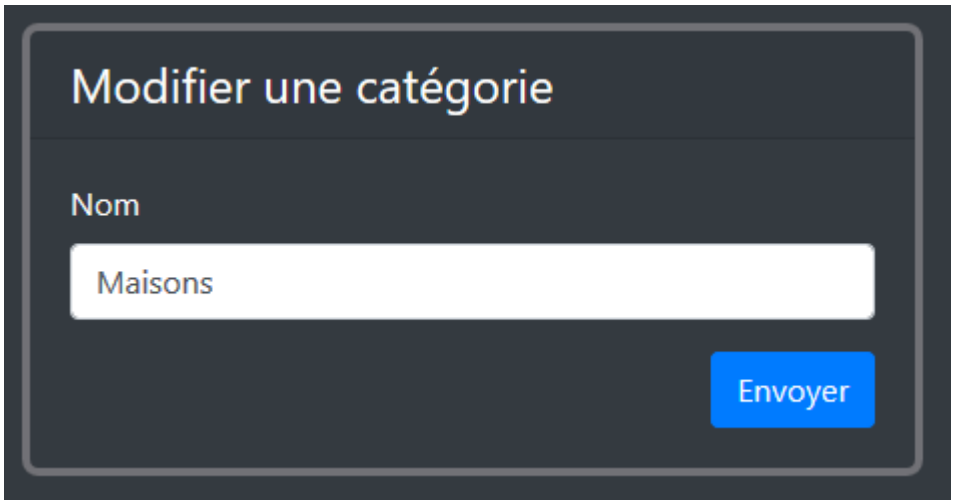
Avec ce code :

```
@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 complète 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 :

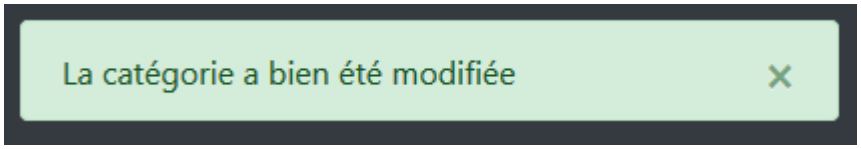


On complète maintenant la fonction **update** dans le contrôleur :

```
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 quo'n a vu pour la création au niveau du layout) :



Des données pour toutes les vues

On a vu ci-dessus qu'on envoie les catégories dans la vue pour afficher la liste. Comme on aura besoin de ces catégories dans pratiquement toutes les vues on va écrire ce code dans **AppServiceProvider** :

```
use App\Models\Category;

...

public function boot()
{
    ...

    if(request()->server("SCRIPT_NAME") !== 'artisan') {
        view ()->share ('categories', Category::all ());
    }
}
```

La méthode **share** permet le partage de données pour toutes les vues. J'inclus ce code dans une condition pour vérifier qu'on est pas avec une commande artisan. En effet si vous lancez une commande de migration vous n'avez pas encore les catégories et vous aurez forcément une erreur.

*Par la suite j'ai changé l'emplacement de cette instruction justement pour éviter de devoir faire un test d'origine et aussi pour rendre les tests éventuels plus sereins. J'ai créé ce commit sur github pour la modification. J'ai opté pour un middleware (même si la documentation indique de le mettre dans **AppServiceprovider**) dans le groupe **web**, comme ça une commande artisan (ou les tests) ne l'atteint pas. Je ne modifie pas cet article parce que l'enchaînement des fichiers à télécharger est trop important et ce n'est pas une modification critique. D'autre part ça ne change rien au principe de la mise en commun de données entre les vues.*

Du coup on peut simplifier la fonction **index** du contrôleur :

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

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
- 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.