Laravel 5.7 par la pratique – Les albums 2/2

Dans le précédant chapitre on a commencé à voir la gestion des albums pour notre galerie photos. On sait maintenant ajouter un album. Maintenant on va voir comment modifier et supprimer un album. 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. Enfin il faudra aussi prévoir dans la barre un menu pour ces albums !

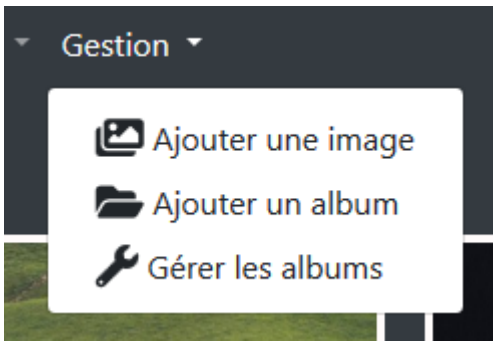
Le menu

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 utilisateurs authentifiés on va avoir ce code :

```
@auth
    <li class="nav-item dropdown">
        <a class="nav-link dropdown-toggle{{ currentRoute(
                            route('album.create'),
                            route('image.create'),
                            route('album.index')
                        )}}"
        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>
                        <a  class="dropdown-item"  href="{{
route('album.create') }}">
```

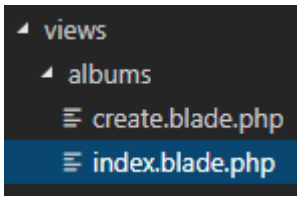
```
                    <i class="fas fa-folder-open fa-lg"></i>
@lang('Ajouter un album')
            </a>
            <a class="dropdown-item" href="{{ route('album.index')
}}">
                <i class="fas fa-wrench fa-lg"></i> @lang('Gérer
les albums')
            </a>
        </div>
    </li>
@endauth
```

Avec ce résultat :



La liste des albums

On va créer maintenant la vue pour lister les albums 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 albums')
```

```
        @endslot


        <table class="table table-dark text-white">
            <tbody>
            @if($userAlbums->isEmpty())
                 <p class="text-center">@lang("Vous n'avez aucun
album pour le moment")</p>
            @else
                @foreach($userAlbums as $album)
                    <tr>
                        <td>{{ $album->name }}</td>
                        <td>
                                    <a type="button" href="{{
route('album.destroy', $album->id) }}"
                                class="btn btn-danger btn-sm pull-
right invisible" data-toggle="tooltip"
                                 title="@lang("Supprimer l'album")
{{ $album->name }}"><i
                                        class="fas fa-trash fa-
lg"></i></a>
                                    <a type="button" href="{{
route('album.edit', $album->id) }}"
                                class="btn btn-warning btn-sm pull-
right mr-2 invisible" data-toggle="tooltip"
                                title="@lang("Modifier l'album") {{
$album->name }}"><i
                                         class="fas fa-edit fa-
lg"></i></a>
                        </td>
                    </tr>
                @endforeach
            @endif
            </tbody>
        </table>

    @endcomponent

@endsection

@section('script')

    <script>
```

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

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

@endsection
```

On a un code équivalent à celui vu pour les catégories.

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

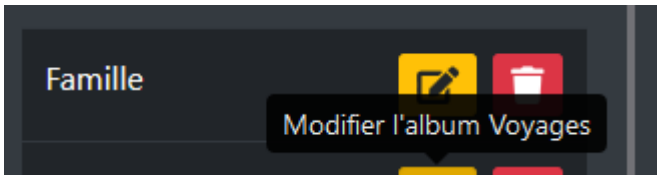
```
public function index(Request $request)
{
     $userAlbums = $this->repository->getAlbums ($request->user ());
    return view ('albums.index', compact('userAlbums'));
}
```

Il nous faut évidemment les albums de l'utilisateur connecté. On délègue cette tâche au repository **AlbumRepository** :

```
public function getAlbums($user)
{
    return $user->albums()->get();
}
```

Normalement en cliquant maintenant dans le menu vous devez obtenir la liste des albums (à condition évidemment d'en avoir créé !) :

Vérifiez que les popups fonctionnent :



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

Supprimer un album

Pour la suppression d’un album j’ai prévu une alerte pour éviter une suppression accidentelle, comme pour les catégories :

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

On complète le code dans le contrôleur :

```
public function destroy(Album $album)
{
    $this->authorize('manage', $album);
    $album->delete ();
    return response ()->json ();
}
```

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

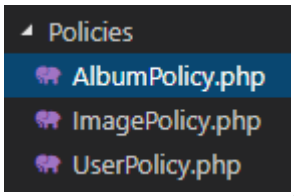
Mais il serait sans doute judicieux de prévoir un filtre pour être sûr que seules des requêtes Ajax effectuent cette action. On complète **AlbumController** :

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

Évidemment que pour la méthode **destroy**.

On autorise que le propriétaire de l'album (et els administrateurs) à faire cette action, on va créer cette autorisation :

```
php artisan make:policy AlbumPolicy
```



Avec ce code :

```
<?php

namespace App\Policies;

use App\Models\ { User, Album };
use Illuminate\Auth\Access\HandlesAuthorization;
```

```
class AlbumPolicy
{
    use HandlesAuthorization;

    public function before(User $user)
    {
        if ($user->admin) {
            return true;
        }
    }

    public function manage(User $user, Album $album)
    {
        return $user->id === $album->user_id;
    }
}
```

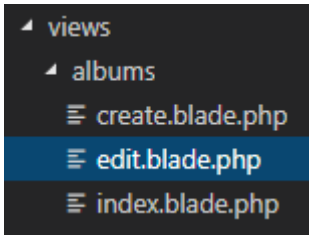
On déclare ça dans **AuthServiceProvider** :

```
use App\Policies\{ AlbumPolicy, ImagePolicy, UserPolicy };
use App\Models\ { Image, User, Album };

    ...

protected $policies = [
    Image::class => ImagePolicy::class,
    User::class => UserPolicy::class,
    Album::class => AlbumPolicy::class,
];
```

Et maintenant la suppression devrait bien se passer !

Modifier un album

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



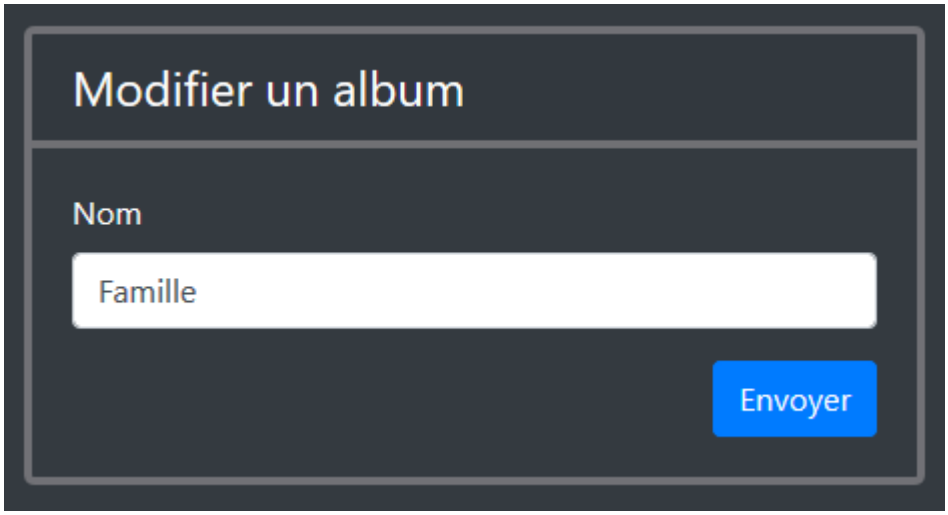
Avec ce code :

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

@section('card')

    @component('components.card')

        @slot('title')
            @lang('Modifier un album')
        @endslot

          <form method="POST" action="{{ route('album.update',
$album->id) }}">
            @csrf
            @method('PUT')

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

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

        </form>

    @endcomponent

@endsection
```

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

```
public function edit(Album $album)
{
    return view ('albums.edit', compact ('album'));
}
```

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

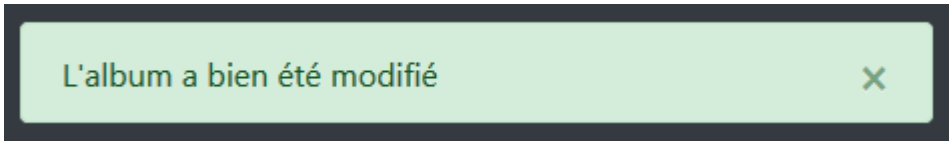


On utilise maintenant la fonction **update** dans le contrôleur :

```
public function update(AlbumRequest $request, Album $album)
{
    $this->authorize('manage', $album);
    $album->update ($request->all ());
     return redirect ()->route('album.index')->with ('ok', __
("L'album a bien été modifié"));
}
```

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 l'album 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) :



Afficher les albums

Il nous faut maintenant les afficher ces albums !

On va ajouter le menu dans la barre (**layouts/app**) :

```
@isset($albums)
    <li class="nav-item dropdown">
        <a class="nav-link dropdown-toggle
```

```
            @isset($album)
                {{ currentRoute(route('album', $album->slug))}}
            @endisset
              " href="#" id="navbarDropdownAlbum" role="button"
data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
            @lang('Albums')
        </a>
                        <div   class="dropdown-menu"   aria-
labelledby="navbarDropdownAlbum">
            @foreach($albums as $album)
                <a class="dropdown-item"
                  href="{{ route('album', $album->slug) }}">{{
$album->name }}</a>
            @endforeach
        </div>
    </li>
@endisset
@admin
```

On voit qu'on attend une variable $albums. Comme on en aura besoin dans toutes les pages on va utiliser un composeur de vue dans **AppServiceProvider** :

```
use App\Repositories\ { CategoryRepository, AlbumRepository };

    ...

if (request ()->server ("SCRIPT_NAME") !== 'artisan') {

                    view     ()->share     ('categories',
resolve(CategoryRepository::class)->getAll());

    view ()->composer('layouts.app', function ($view)
    {
        if(auth()->check()) {
                                        $albums   =   resolve
(AlbumRepository::class)->getByUser(auth()->id());
            if($albums->isNotEmpty()) {
                $view->with('albums', $albums);
            }
        }
    });
}
```

On ajoute la route pour les slugs :

```
Route::name       ('album')->get       ('album/{slug}',
'ImageController@album');
```

Dans **ImageController** on ajoute la fonction pour aller chercher le bon album et ses images :

```
....

use App\Repositories\ {
    ImageRepository, AlbumRepository, CategoryRepository
};

    ...

protected $albumRepository;

public function __construct(
    ImageRepository $imageRepository,
    AlbumRepository $albumRepository,
    CategoryRepository $categoryRepository)
{
    $this->imageRepository = $imageRepository;
    $this->albumRepository = $albumRepository;
    $this->categoryRepository = $categoryRepository;
}

    ...

public function album($slug)
{
    $album = $this->albumRepository->getBySlug ($slug);
    $images = $this->imageRepository->getImagesForAlbum ($slug);
    return view ('home', compact ('album', 'images'));
}
```

Et on ajoute le traitement dans **ImageRepository** :

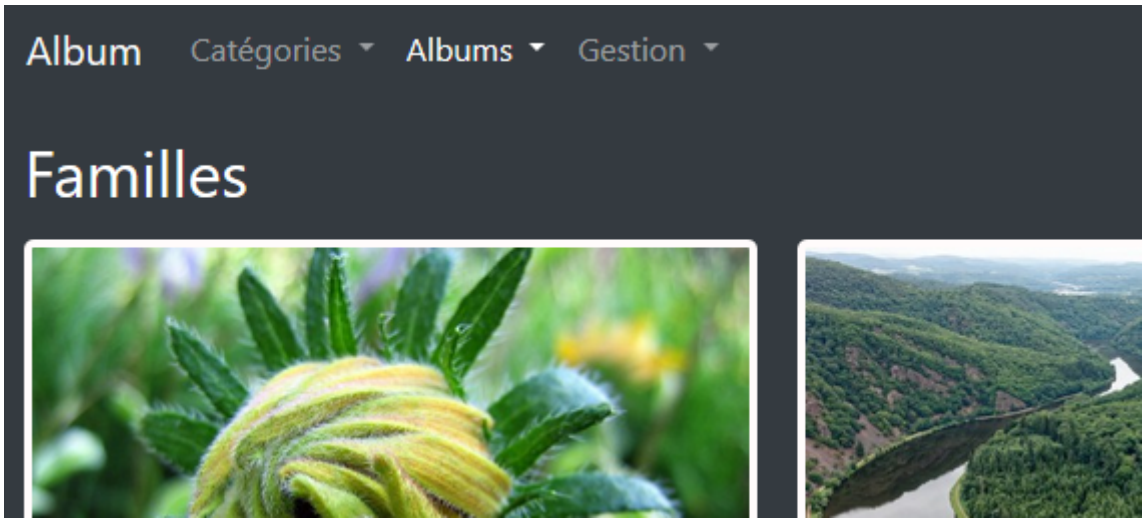
```
public function getImagesForAlbum($slug)
{
    return Image::latestWithUser ()->whereHas ('albums', function
($query) use ($slug) {
        $query->whereSlug ($slug);
```

```
    })->paginate(config('app.pagination'));
}
```

Maintenant le menu doit fonctionner mais on n'a pas encore mis des photos dans des albums ! Pour voir si ça fonctionne vous pouvez directement renseigner la table pivot **album_image**.

On va juste ajouter le nom de l'album en cours dans la vue **home** :

```
@isset($album)
    <h2 class="text-title mb-3">{{ $album->name }}</h2>
@endif
```



On remplit les albums

Maintenant qu'on sait créer des albums et les afficher il faut pouvoir les garnir avec des photos !

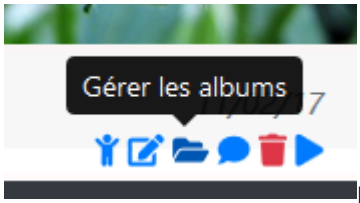
On va ajouter une icône dans le menu des images dans la vue **home** :

```
<a class="albums-manage"
   href="{{ route('image.albums', $image->id) }}"
   data-toggle="tooltip"
   title="@lang('Gérer les albums')">
   <i class="fa fa-folder-open"></i>
</a>
```

Et pour que ça fonctionne on ajoute la route :

```
Route::middleware ('auth', 'verified')->group (function () {
```

```
    ...

    Route::name ('image.')->middleware ('ajax')->group (function
() {
        Route::prefix('image')->group(function () {

            ...

                Route::name('albums')->get('{image}/albums',
'ImageController@albums');
        });
    });
});
```



Dans un premier temps on met en place le Javascript dans la vue **home** pour envoyer la demande des albums disponibles :

```
$('a.albums-manage').click((e) => {
    e.preventDefault()
    let that = $(e.currentTarget)
    that.tooltip('hide')
    that.children().removeClass('fa-folder-open').addClass('fa-cog
fa-spin')
    e.preventDefault()
    $.get(that.attr('href'))
        .done((data) => {
                    that.children().addClass('fa-folder-
open').removeClass('fa-cog fa-spin')
            $('#listeAlbums').html(data)
            $('#manageAlbums').attr('action', that.attr('href'))
            $('#editAlbums').modal('show')
        })
        .fail(() => {
                    that.children().addClass('fa-folder-
open').removeClass('fa-cog fa-spin')
            swallAlertServer()
        })
})
```

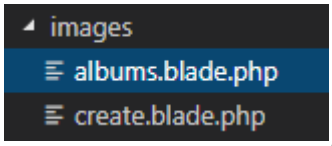
La demande arrive dans **ImageController** :

```
public function albums(Request $request,  Image $image)
{
    $this->authorize ('manage', $image);
      $albums  =  $this->albumRepository->getAlbumsWithImages
($request->user ());
    return view ('images.albums', compact('albums', 'image'));
}
```

On traite ça dans **AlbumRepository** :

```
public function getAlbumsWithImages($user)
{
    return $user->albums()->with('images')->get();
}
```

On voit qu'on renvoie une vue **images.albums**, créons cette vue :



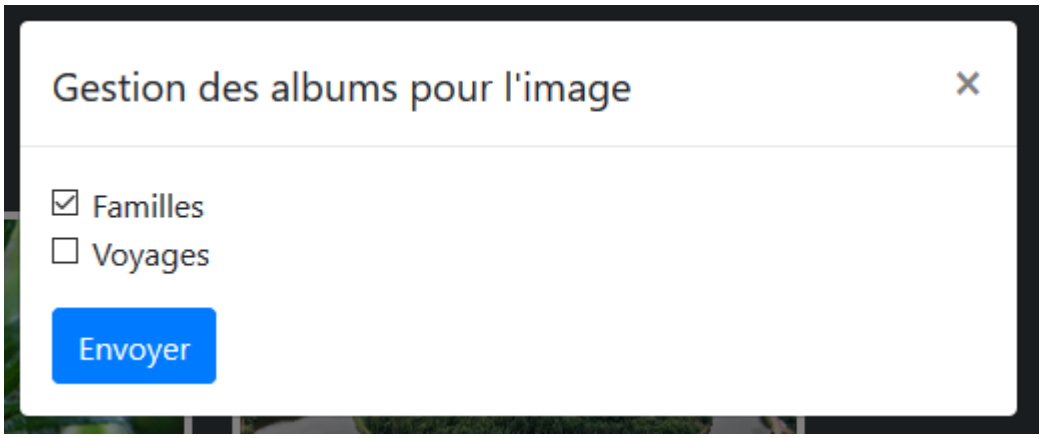
Avec ce code :

```
@foreach($albums as $album)
    <div class="form-check">
        <label class="form-check-label">
            <input
                class="form-check-input"
                name="albums[]"
                value="{{ $album->id }}"
                type="checkbox"
                 @if ($album->images->contains('id', $image->id))
checked @endif
            >
            {{ $album->name }}
        </label>
    </div>
@endforeach
```

Dans la vue **home** on prévoit une feuille modale pour afficher la liste des albums sous forme de formulaire avec cases à cocher :

```
<div  class="modal  fade"  id="editAlbums"  tabindex="-1"
role="dialog" aria-labelledby="albumLabel" aria-hidden="true">
```

```
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                                    <h5  class="modal-title"
id="albumLabel">@lang("Gestion des albums pour l'image")</h5>
                    <button type="button" class="close" data-
dismiss="modal" aria-label="Close">
                     <span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body">
                <form id="manageAlbums" action="" method="POST">
                                    <div  class="form-group"
id="listeAlbums"></div>
                        <button type="submit" class="btn btn-
primary">@lang('Envoyer')</button>
                </form>
            </div>
        </div>
    </div>
</div>
```



On gère la soumission en Ajax dans la vue **home** :

```
$('#manageAlbums').submit((e) => {
    e.preventDefault()
    let that = $(e.currentTarget)
    $.ajax({
        method: 'put',
        url: that.attr('action'),
        data: that.serialize()
    })
        .done((data) => {
            if(data === 'reload') {
```

```
                location.reload();
            } else {
                $('#editAlbums').modal('hide')
            }
        })
        .fail(() => {
            swallAlertServer()
        })
})
```

On crée la route :

```
Route::name ('image.')->middleware ('ajax')->group (function () {
    Route::prefix('image')->group(function () {
          Route::name ('albums.update')->put ('{image}/albums',
'ImageController@albumsUpdate');

        ...

    });
});
```

Ça arrive dans **ImageController** :

```
public function albumsUpdate(Request $request, Image $image)
{
    $this->authorize ('manage', $image);
    $image->albums()->sync($request->albums);

    $path = pathinfo (parse_url(url()->previous())['path']);

    if($path['dirname'] === '/album') {

             $album = $this->albumRepository->getBySlug
($path['basename']);

        if($this->imageRepository->isNotInAlbum ($image, $album))
{
            return response ()->json('reload');
        }
    }

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

On a pas la même réponse selon que l'image se trouvait à l'origine dans un album affiché, auquel cas il faut recharger la page, sinon on ne change rien. Pour savoir si une image est dans un certain album on ajoute cette fonction dans **ImageRepository** :

```
public function isNotInAlbum($image, $album)
{
          return   $image->albums()->where('albums.id', $album->id)->doesntExist();
}
```

On peut maintenant gérer complètement les albums !

Conclusion

Dans ce chapitre on a :

- modifié le menu de la barre de navigation pour ajouter un item pour la gestion des albums
- créé une vue pour lister les albums avec des boutons pour la modification et la suppression
- créé le code pour la suppression des albums
- créé la vue et le code pour la modification des albums
- ajouté l'affichage des albums avec un menu
- ajouté la possibilité d'affecter les images aux albums à l'aide d'une icône dans leur menu

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