Laravel 5.7 par la pratique – Les albums 1/2

On va poursuivre la création de notre galerie photos en nous intéressant dans ce chapitre aux albums. On a vu qu'on organise les photos de la galerie avec des catégories et seul un administrateur peut créer, modifier ou supprimer une catégorie.

Maintenant on va permettre à chaque utilisateur inscrit d'organiser ses photos dans des albums personnels.

On va voir dans ce chapitre comment on crée un album et forcément le code est très proche de celui utilisé pour les catégories. Mais on va devoir ajouter 2 tables à la base de données et installer quelque relations. Il faudra également compléter le menu de la barre de navigation, créer un formulaire, les routes, le contrôleur, un événement…

La base de données

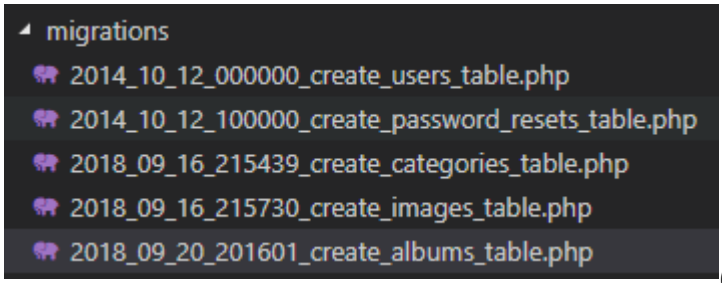
Il nous faut compléter notre base de données pour intégrer les albums. Une table pour les informations des albums : nom, slug, dates. D'autre part un album sera relié à un utilisateur, il faudra donc établir une relation avec **User** de type **One To Many**.

On doit aussi établir une relation entre les albums et les images : un album contient plusieurs images et une image peut être dans plusieurs albums, on aura donc une relation **Many To Many**.

Les migrations

On va ajouter une migration pour la table **albums** :

```
php artisan make:migration create_albums_table --create=albums
```

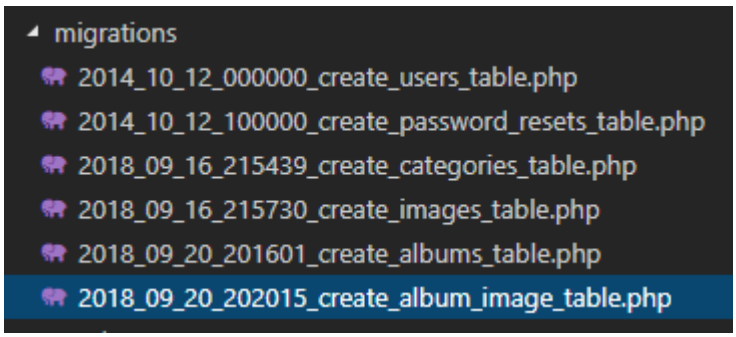


Changez ainsi le code :

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateAlbumsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('albums', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name')->unique();
            $table->string('slug')->unique();
            $table->integer('user_id')->unsigned();
$table->foreign('user_id')->references('id')->on('users')->onDelet
e('cascade');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('albums');
    }
}
```

On ajoute aussi une migration pour la table pivot entre les albums et les images :

```
php artisan make:migration create_album_image_table
```



Avec ce code :

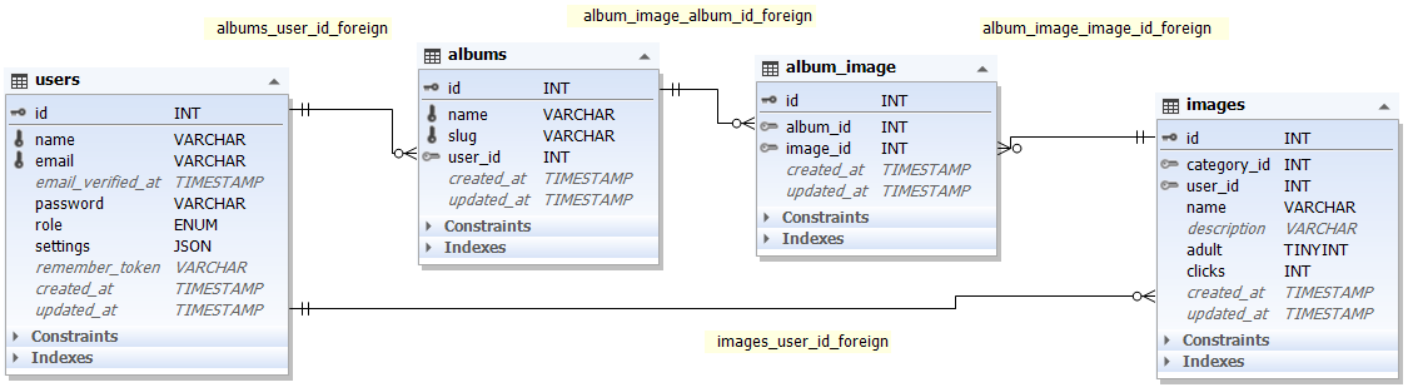
```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateAlbumImageTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('album_image', function (Blueprint $table)
{
            $table->increments('id');
            $table->unsignedInteger('album_id');
            $table->unsignedInteger('image_id');
$table->foreign('album_id')->references('id')->on('albums')->onDel
ete('cascade');
$table->foreign('image_id')->references('id')->on('images')->onDel
ete('cascade');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
```

```
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('album_image');
    }
}
```

On peut maintenant rafraîchir la base :

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

On se retrouve avec cette organisation :



Les modèles

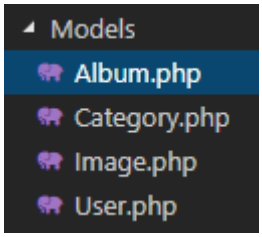
Dans le modèle **Image** on ajoute la relation :

```
public function albums()
{
    return $this->belongsToMany (Album::class);
}
```

Dans le modèle **User** on ajoute la relation :

```
public function albums()
{
    return $this->hasMany (Album::class);
}
```

On crée un modèle **Album** :



Avec ce code :

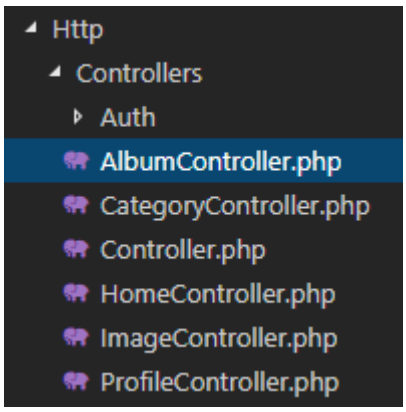
```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use App\Events\NameSaving;

class Album extends Model
{
    protected $fillable = [
        'name', 'slug',
    ];

    protected $dispatchesEvents = [
        'saving' => NameSaving::class,
    ];

    public function images()
    {
        return $this->belongsToMany (Image::class);
    }

    public function user()
    {
        return $this->belongsTo (User::class);
    }
}
```

On a mis en place les deux relations. D'autre part on a prévu le déclenchement de l'événement **NameSaving** pour la création du slug comme on l'avait fait pour les catégories.

Le contrôleur

Pour gérer les albums on va créer un contrôleur :

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



Le fait d'utiliser l'option **–resource** a généré les 7 méthodes de base. On va toutes les conserver sauf **show**.

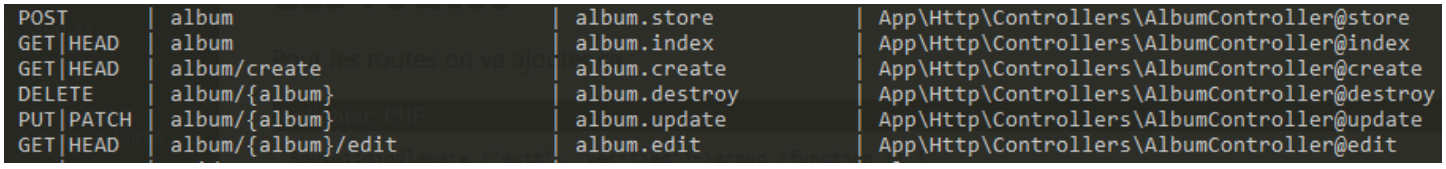
Les routes

Pour les routes on va ajouter ça :

```
Route::middleware ('auth', 'verified')->group (function () {
    Route::resource ('album', 'AlbumController', [
        'except' => 'show'
    ]);

    ...

});
```

On vérifie :

```
php artisan route:list
```



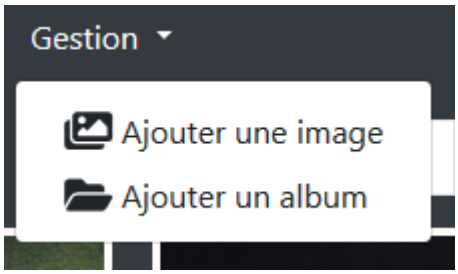
On a bien nos 6 routes pour notre contrôleur.

Le menu

Pour accéder au formulaire de création d'un album on va devoir compléter la barre de navigation dans **views/layouts/app** :

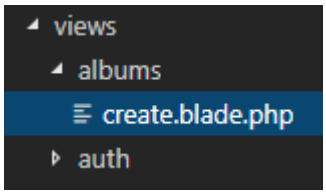
```
@auth
    <li class="nav-item dropdown">
        <a class="nav-link dropdown-toggle{{ currentRoute(
                            route('album.create'),
                            route('image.create')
                        )}}"
        href="#" id="navbarDropdownGestAlbum" role="button" data-
toggle="dropdown"
        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>
                        <a  class="dropdown-item"  href="{{
route('album.create') }}">
                    <i class="fas fa-folder-open fa-lg"></i>
@lang('Ajouter un album')
            </a>
        </div>
    </li>
@endauth
```

Si on a affaire à un utilisateur authentifié (**@auth**) on crée le menu déroulant. On l'avait créé pour les catégories, là on ajoute l'album :



La vue de création

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



Pour cette vue on utilise l'intendance qu'on a précédemment mise en place :

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

@section('card')

    @component('components.card')

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

        <form method="POST" action="{{ route('album.store') }}">
            @csrf

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

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

        </form>

    @endcomponent

@endsection
```

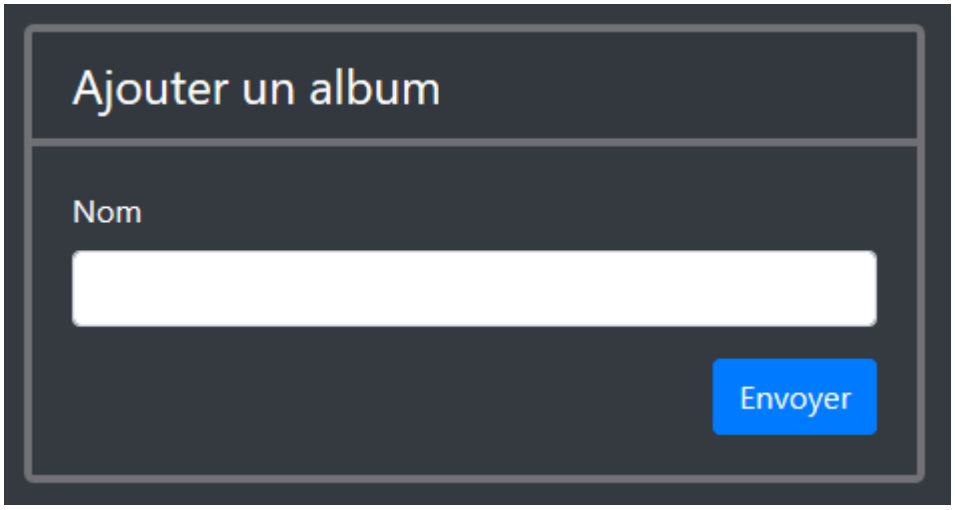
L’affichage du formulaire

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

Déjà il faut afficher le formulaire :

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

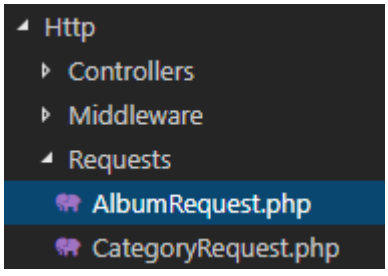
On vérifie avec le menu que ça marche (il faut se connecter comme administrateur) :



La validation

Pour la validation on crée une requête de formulaire :

```
php artisan make:request AlbumRequest
```



Et on change ainsi le code :

```
<?php

namespace App\Http\Requests;
use Illuminate\Foundation\Http\FormRequest;

class AlbumRequest extends FormRequest
{
    public function authorize()
    {
        return true;
    }

    public function rules()
    {
        $id = $this->album ? ',' . $this->album->id : '';

        return $rules = [
            'name' => 'required|string|max:255|unique:albums,name'
. $id,
        ];
    }
}
```

C’est déjà préparé pour gérer la validation de la modification. Dans ce cas on sait qu’il faut arranger un peu la règle d’unicité.

La soumission

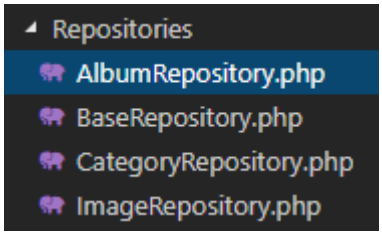
A la soumission on arrive dans la méthode **store** du contrôleur. On va la coder ainsi :

```
<?php

namespace App\Http\Controllers;

use App\Models\Album;
use App\Http\Requests\AlbumRequest;
use App\Repositories\AlbumRepository;
use Illuminate\Http\Request;

class AlbumController extends Controller
{
```

```
    protected $repository;

    public function __construct(AlbumRepository $repository)
    {
        $this->repository = $repository;
    }

    public function store(AlbumRequest $request)
    {
        $this->repository->create ($request->user(), $request->all
());

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

Un repository

On voit qu'on passe par un repository histoire de bien organiser le code. Créons ce repository :



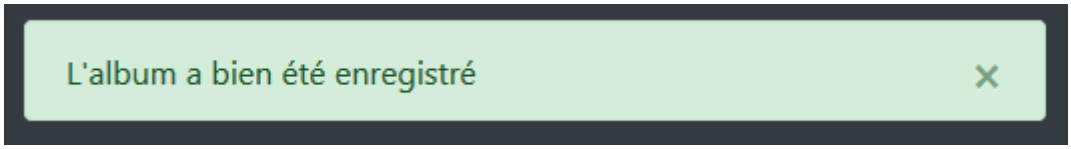
Avec ce code :

```
<?php

namespace App\Repositories;

use App\Models\Album;

class AlbumRepository extends BaseRepository
{
    public function __construct(Album $album)
    {
        $this->model = $album;
    }
```

```
    public function create($user, array $inputs)
    {
        $user->albums ()->create($inputs);
    }
}
```

On voit qu'on étend encore le repository de base parce qu'on va avoir des méthodes communes à tous nos repositories.

Maintenant on doit pouvoir ajouter un album :



id UNSIGNED INT(10)	name VARCHAR(255)	slug VARCHAR(255)	user_id UNSIGNED INT(10)	created_at TIMESTAMP	updated_at TIMESTAMP
1	Mon premier album	mon-premier-album	1	20/09/2018 20:37:23	20/09/2018 20:37:23

On verra dans le prochain chapitre la gestion des albums : modification et supression. On ajoutera aussi une icône sur les images pour gérer directement leur affectation aux albums.

Conclusion

Dans ce chapitre on a :

- créé les migrations pour les albums
- créé un modèle pour les albums
- créé les relations entre les modèles
- créé routes, contrôleur, repository et requête de formulaire pour la création d'un album
- complété la barre de navigation

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