

Laravel 5.7 par la pratique – Les catégories 1/2

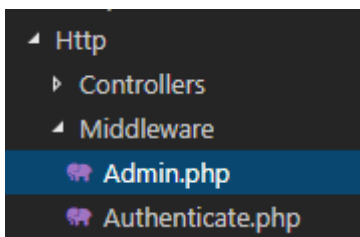
On va poursuivre la création de notre galerie photos en nous intéressant dans ce chapitre aux catégories. En effet, pour organiser un peu les photos de la galerie on les classe en catégories. Seul un administrateur peut créer, modifier ou supprimer une catégorie.

On va voir dans ce chapitre comment on crée une catégorie. Pour ça on va devoir déjà compléter le menu de la barre de navigation, créer un formulaire, les routes, le contrôleur et même un middleware et un événement...

Un middleware

Pour vérifier qu'on a affaire à un administrateur le plus simple est de créer un middleware :

```
php artisan make:middleware Admin
```



On va changer ainsi la fonction **handle** :

```
public function handle($request, Closure $next)
{
    $user = $request->user();

    if ($user && $user->admin) {
        return $next($request);
    }

    return redirect()->route('home');
}
```

Si c'est un administrateur on continue, sinon on renvoie sur la page d'accueil.

Vous êtes sans doute intrigué par `$user->admin` parce qu'il n'y a pas d'attribut `admin` dans le modèle `User`. On va donc s'arranger pour en avoir un en ajoutant cette fonction dans `App\Models\User` :

```
/**
 * User is admin.
 *
 * @return integer
 */
public function getAdminAttribute()
{
    return $this->role === 'admin';
}
```

On crée ainsi un accesseur, bien pratique !

On déclare maintenant le middleware dans `app/Http/Kernel.php` :

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

On va en profiter pour ajouter une directive à Blade dans `AppServiceProvider` :

```
use Illuminate\Support\Facades\Blade;

...

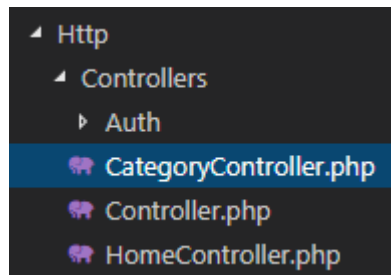
public function boot()
{
    Blade::if('admin', function () {
        return auth()->check() && auth()->user()->admin;
    });
}
```

On pourra ainsi écrire `@admin` dans les vues !

Le contrôleur

Pour gérer les catégories on va créer un contrôleur :

```
php artisan make:controller CategoryController --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('admin')->group(function () {  
  
    Route::resource ('category', 'CategoryController', [  
        'except' => 'show'  
    ]);  
  
});
```

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

On vérifie :

```
php artisan route:list
```

POST	category	category.store	App\Http\Controllers\CategoryController@store	web,admin
GET HEAD	category	category.index	App\Http\Controllers\CategoryController@index	web,admin
GET HEAD	category/create	category.create	App\Http\Controllers\CategoryController@create	web,admin
PUT PATCH	category/{category}	category.update	App\Http\Controllers\CategoryController@update	web,admin
DELETE	category/{category}	category.destroy	App\Http\Controllers\CategoryController@destroy	web,admin
GET HEAD	category/{category}/edit	category.edit	App\Http\Controllers\CategoryController@edit	web,admin

On a bien nos 6 routes pour notre contrôleur.

Le menu

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

```
<nav class="navbar fixed-top navbar-expand-lg navbar-dark bg-dark">
  <a class="navbar-brand" href="{{ route('home') }}">{{
config('app.name', 'Album') }}</a>
  <button class="navbar-toggler" type="button" data-
toggle="collapse" data-target="#navbarSupportedContent" aria-
controls="navbarSupportedContent" aria-expanded="false" aria-
label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
    <div class="collapse navbar-collapse"
id="navbarSupportedContent">
      <ul class="navbar-nav mr-auto">
        @admin
          <li class="nav-item dropdown">
            <a class="nav-link dropdown-toggle{{
currentRoute(route('category.create')) }}" 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>
              </div>
            </li>
          @endadmin
        </ul>
        <ul class="navbar-nav ml-auto">
          @guest
            <li class="nav-item{{ currentRoute(route('login'))
}}"><a class="nav-link" href="{{ route('login')
}}">@lang('Connexion')</a></li>
```

```

        <li class="nav-item{{ currentRoute(route('register'))
}}"><a class="nav-link" href="{{ route('register')
}}">@lang('Inscription')</a></li>
        @else
            <li class="nav-item">
                <a id="logout" class="nav-link" href="{{
route('logout') }}">@lang('Déconnexion')</a>
                <form id="logout-form" action="{{
route('logout') }}" method="POST" class="hide">
                    {{ csrf_field() }}
                </form>
            </li>
        @endguest
    </ul>
</div>
</nav>

```

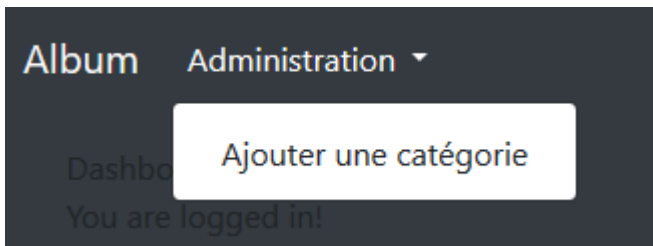
La partie ajoutée est celle-ci :

```

@admin
<li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle{{
currentRoute(route('category.create')) }}" 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>
    </div>
</li>
@endadmin

```

Si on a affaire à un administrateur (**@admin**) on crée un menu déroulant. Pour le moment on déroule un seul item mais on complétera plus tard.



Dans le code j'ai prévu une icône de Font Awesome mais elle n'apparaît pas parce qu'on n'a pas chargé la librairie ni les icônes. La librairie est passée en version 5 avec des icônes en SVG. On va utiliser npm pour charger la librairie :

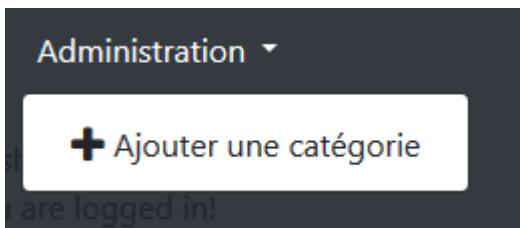
```
npm install @fortawesome/fontawesome-free
```

Et dans **resources/js/app.js** :

```
...  
try {  
  window.$ = window.jQuery = require('jquery');  
  require('bootstrap');  
  require('@fortawesome/fontawesome-free/js/all.js');  
} catch (e) {}
```

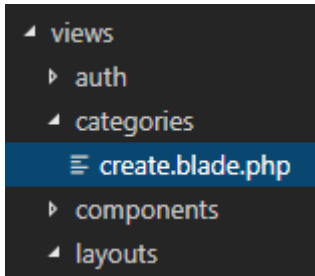
Plus qu'à recompiler tout ça avec **npm run dev**.

Et voilà une icône dans le menu déroulant :



La vue de création

On crée un dossier pour les catégories 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 une catégorie')
```

```
        @endslot
```

```
        <form method="POST" action="{{ route('category.store') }}">
```

```
            {{ csrf_field() }}
```

```
            @include('partials.form-group', [
```

```
                'title' => __('Nom'),
```

```
                'type' => 'text',
```

```
                'name' => 'name',
```

```
                'required' => true,
```

```
            ])
```

```
            @component('components.button')
```

```
                @lang('Envoyer')
```

```
            @endcomponent
```

```
        </form>
```

```
    @endcomponent
```

```
@endsection
```

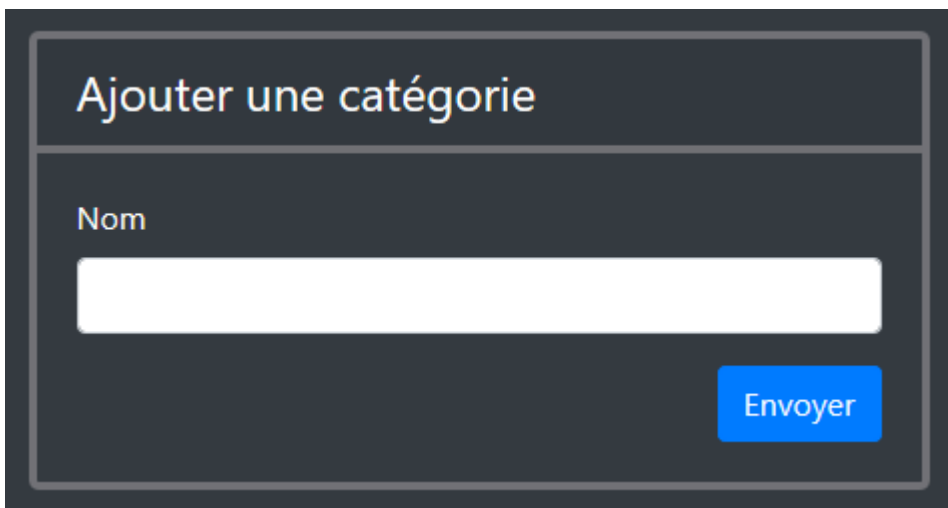
L'affichage du formulaire

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('categories.create');
}
```

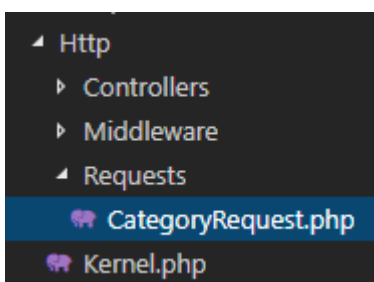
On vérifie avec le menu que ça marche (il faut se connecter comme administrateur) :

A screenshot of a web application interface. At the top, there is a dark header with the text "Ajouter une catégorie" in white. Below the header is a form area with a light gray border. Inside the form, there is a label "Nom" above a white text input field. To the right of the input field is a blue button with the text "Envoyer" in white.

La validation

Pour la validation on crée une requête de formulaire :

```
php artisan make:request CategoryRequest
```



Et on change ainsi le code :


```

<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

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

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

        return $rules = [
            'name' =>
            'required|string|max:255|unique:categories,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 :

```

use App\Models\Category;
use App\Http\Requests\CategoryRequest;
use App\Repositories\CategoryRepository;

...

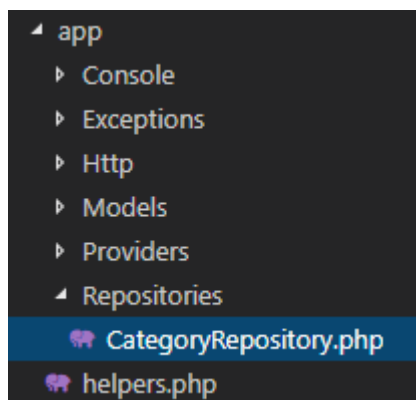
public function store(CategoryRequest $request, CategoryRepository
$repository)
{
    $repository->store($request->all ());
}

```

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

Un repository

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



Avec ce code :

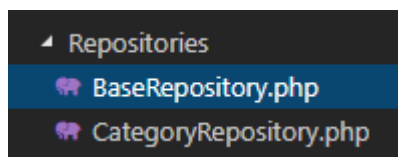
```
<?php

namespace App\Repositories;

use App\Models\Category;

class CategoryRepository extends BaseRepository
{
    public function __construct(Category $category)
    {
        $this->model = $category;
    }
}
```

On voit qu'on étend un repository de base parce qu'on va avoir des méthodes communes à tous nos repositories. On crée donc ce repository de base :



Avec ce code :

```
<?php
```

```

namespace App\Repositories;

abstract class BaseRepository
{
    protected $model;

    public function store(array $inputs)
    {
        return $this->model->create($inputs);
    }

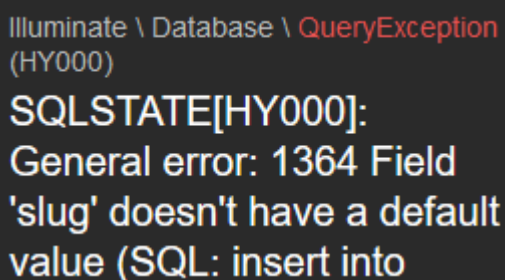
    public function getBySlug($slug)
    {
        return $this->model->whereSlug($slug)->firstOrFail();
    }

    public function getByUser($id)
    {
        return $this->model->whereUserId($id)->get();
    }

    public function getAll()
    {
        return $this->model->all();
    }
}

```

Mais évidemment ça ne va pas encore fonctionner :



```

Illuminate \ Database \ QueryException
(HY000)
SQLSTATE[HY000]:
General error: 1364 Field
'slug' doesn't have a default
value (SQL: insert into

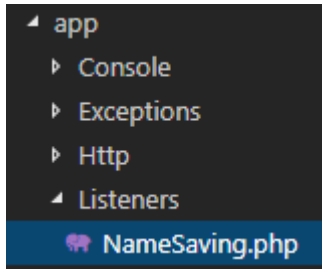
```

En effet on a pas donné de valeur au **slug**. Comme on va en avoir besoin dans le cas de la création et de la modification on va se servir d'un événement.

Un événement

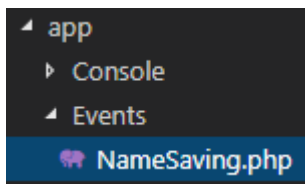
On crée le listener :

```
php artisan make:listener NameSaving
```



Et l'événement :

```
php artisan make:event NameSaving
```



Dans l'événement on se contente de transmettre le modèle :

```
<?php
```

```
namespace App\Events;
```

```
use Illuminate\ {  
    Queue\SerializesModels,  
    Database\Eloquent\Model  
};
```

```
class NameSaving
```

```
{  
    use SerializesModels;  
  
    public $model;  
  
    public function __construct(Model $model)  
    {  
        $this->model = $model;  
    }  
}
```

```
}
```

Et dans le listener on code le slug :

```
<?php
```

```
namespace App\Listeners;
```

```
use App\Events\NameSaving as EventNameSaving;
```

```
class NameSaving
```

```
{
```

```
    public function handle(EventNameSaving $event)
```

```
    {
```

```
        $event->model->slug = str_slug($event->model->name, '-');
```

```
    }
```

```
}
```

Merci à l'helper **str_slug** de Laravel !

On établit la liaison entre event et listener dans **EventServiceProvider** :

```
protected $listen = [
```

```
                                Registered::class =>
```

```
[SendEmailVerificationNotification::class,],
```

```
    'App\Events\NameSaving' => ['App\Listeners\NameSaving',],
```

```
];
```

Il ne reste plus qu'à propager l'événement à partir du modèle **Category** :

```
use App\Events\NameSaving;
```

```
...
```

```
protected $dispatchesEvents = [
```

```
    'saving' => NameSaving::class,
```

```
];
```

Maintenant la sauvegarde devrait fonctionner !

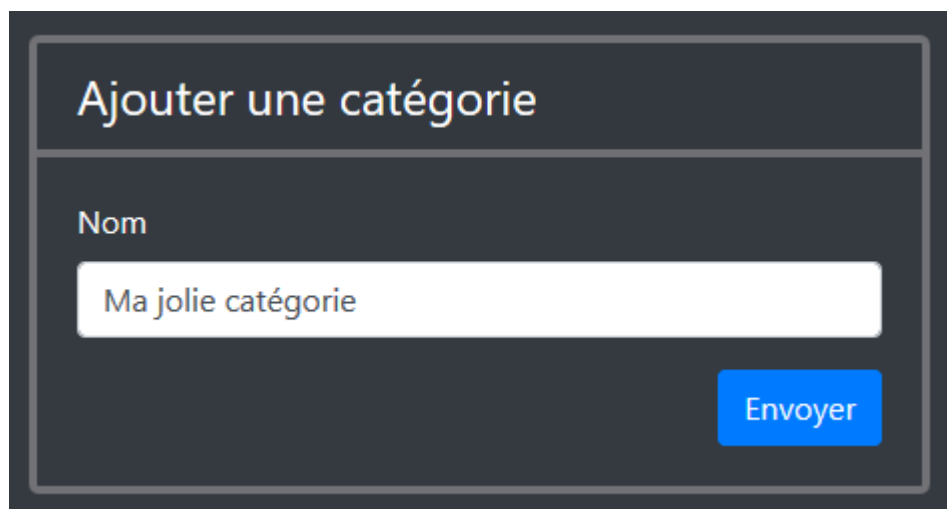
Le message

Dans le contrôleur on renvoie une information de réussite en flash session. Il faut prévoir dans le layout (**views/layouts/app**) de quoi l'afficher :

```
@if (session('ok'))
  <div class="container">
    <div class="alert alert-dismissible alert-success fade
show" role="alert">
      {{ session('ok') }}
      <button type="button" class="close" data-
dismiss="alert" aria-label="Close">
        <span aria-hidden="true">&times;</span>
      </button>
    </div>
  </div>
@endif
```

```
@yield('content')
```


Et maintenant ça doit fonctionner :



Ajouter une catégorie

Nom

Envoyer



La catégorie a bien été enregistrée

id	name	slug
1	Ma jolie catégorie	ma-jolie-categorie

Conclusion

Dans ce chapitre on a :

- créé un middleware pour l'administration
- créé une directive Blade
- créé routes, contrôleur, repository et requête de formulaire pour la création d'une catégorie
- complété la barre de navigation
- créé un événement pour la création du slug
- modifié le layout pour afficher un message

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