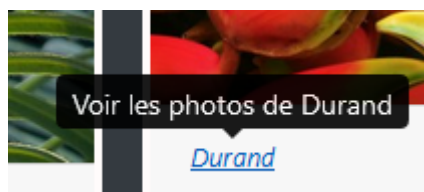


Créer une application avec Laravel 5.5 – La galerie 2/2

Dans le précédent chapitre on a affiché la galerie en prévoyant la possibilité d'une présentation par catégorie. On va dans ce chapitre compléter avec une présentation par utilisateur. D'autre part on va mettre en place le code pour la suppression des photos. On va également créer les pages pour les erreurs les plus classiques. On finira avec la suppression des images orphelines.

Les photos d'un utilisateur

Dans la galerie pour chaque photo on a le nom de celui qui l'a envoyée. C'est un lien avec un popup au survol :



Pour le moment on n'a pas référencé l'url correspondante (**views/home**) :

```
<a href="#" data-toggle="tooltip" title="{{ __('Voir les photos de ' . $image->user->name }}">{{ $image->user->name }}</a>
```

On commence par ajouter une route :

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

```
GET|HEAD | user/{user} | user | App\Http\Controllers\ImageController@user | web
```

On crée la fonction dans **ImageController** :

```
use App\Models\ { Category, User };
```

```
...
```

```
public function user(User $user)
{
```

```

    $images = $this->repository->getImagesForUser($user->id);

    return view('home', compact('user', 'images'));
}

```

La fonction dans **ImageRepository** :

```

public function getImagesForUser($id)
{
    return Image::latestWithUser()->whereHas('user', function
($query) use ($id) {
        $query->whereId($id);
    }->paginate(config('app.pagination')));
}

```

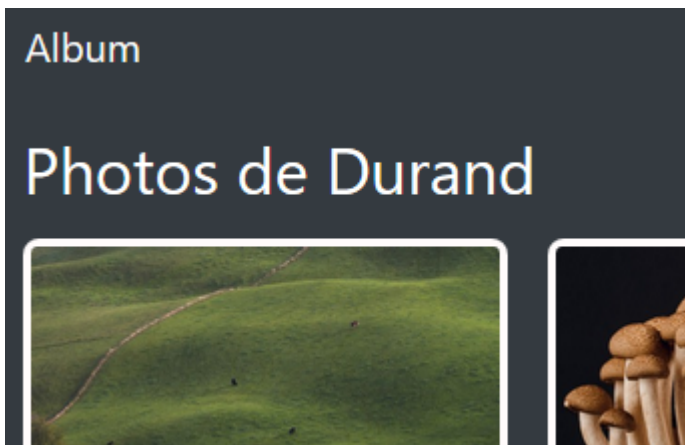
Et on complète la vue (**views/home**) :

```

<a href="{{ route('user', $image->user->id) }}" data-
toggle="tooltip" title="{{ __('Voir les photos de ') .
$image->user->name }}">{{ $image->user->name }}</a>

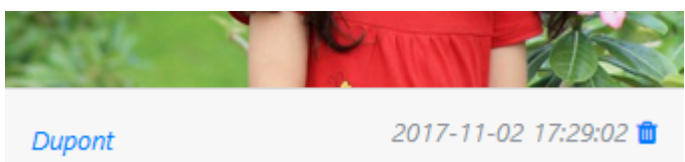
```

Et maintenant on peut cliquer sur le nom :



Supprimer une photo

Quand un utilisateur est connecté on lui permet de supprimer ses photos (et s'il est administrateur il peut toutes les supprimer) :



Dans la vue (**views/home**) c'est cette partie du code qui est concernée :

```
@adminOrOwner($image->user_id)
    <a class="form-delete" href="{{ route('image.destroy',
$image->id) }}" data-toggle="tooltip" title="@lang('Supprimer
cette photo')"><i class="fa fa-trash"></i></a>
    <form action="{{ route('image.destroy', $image->id) }}"
method="POST" class="hide">
        {{ csrf_field() }}
        {{ method_field('DELETE') }}
    </form>
@endadminOrOwner
```

...

```
$('.a.form-delete').click(function(e) {
    e.preventDefault();
    let href = $(this).attr('href')
    $("form[action='" + href + "']").submit()
})
```

On utilise la directive Blade qu'on a créée au précédent chapitre (**@adminOrOwner**) pour faire apparaître l'icône pour les utilisateurs concernés.

On a un formulaire et une soumission par Javascript.

On a déjà créé la route précédemment dans cette ressource :

```
Route::resource('image', 'ImageController', [
    'only' => ['create', 'store', 'destroy']
]);
```

On met ce code dans **ImageController** :

```
use App\Models\ { Category, User, Image };
```

...

```
public function destroy(Image $image)
{
    $image->delete();
}
```

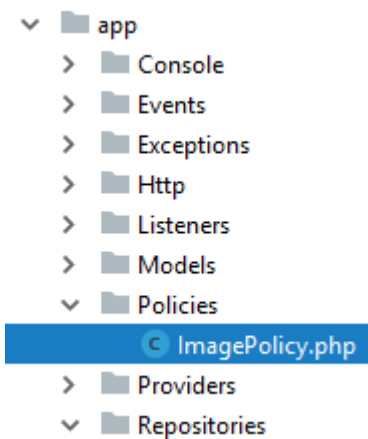
```
        return back();
    }
}
```

Et maintenant si on clique sur la petite poubelle l'image disparaît.

Mais on a quand même un petit souci de sécurité. ce n'est pas parce qu'on n'affiche pas une icône aux autres utilisateurs qu'ils ne sont pas capable de générer une requête pour supprimer une photo, même si ça demande quelques connaissances...

On va donc ajouter une autorisation pour verrouiller cette possibilité :

php artisan make:policy ImagePolicy



Avec ce code :

```
<?php
```

```
namespace App\Policies;
```

```
use App\Models\ { User, Image };
```

```
use Illuminate\Auth\Access\HandlesAuthorization;
```

```
class ImagePolicy
```

```
{
    use HandlesAuthorization;

    /**
     * Grant all abilities to administrator.
     *
     * @param  \App\Models\User  $user
     */
}
```

```

    * @return bool
    */
    public function before(User $user)
    {
        if ($user->role === 'admin') {
            return true;
        }
    }

    /**
     * Determine whether the user can delete the image.
     *
     * @param \App\Models\User $user
     * @param \App\Models\Image $image
     * @return mixed
     */
    public function delete(User $user, Image $image)
    {
        return $user->id === $image->user_id;
    }
}

```

Dans la fonction **before** on autorise les administrateurs et dans la fonction **delete** l'owner.

On l'enregistre dans **AuthServiceProvider** :

```

use App\Policies\ImagePolicy;
use App\Models\Image;

...

protected $policies = [
    Image::class => ImagePolicy::class,
];

```

Et on l'ajoute dans **ImageController** :

```

public function destroy(Image $image)
{
    $this->authorize('delete', $image);

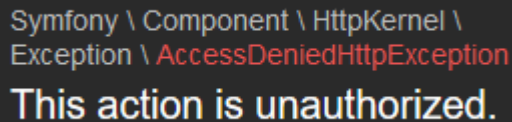
    $image->delete();
}

```

```
    return back();  
}
```

Maintenant on est sûrs qu'une petit malin ne pourra pas supprimer une photo qui ne lui appartient pas !

Pour vérifier que ça fonctionne supprimez la directive **@adminOrOwner** dans la vue home, connectez-vous avec **Dupont** tentez de supprimer une photo de **Durand** :



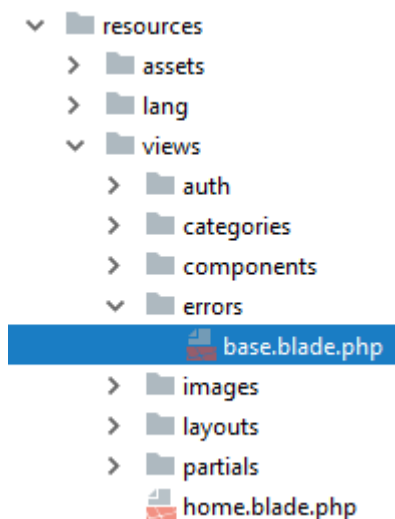
```
Symfony \ Component \ HttpKernel \  
Exception \ AccessDeniedHttpException  
This action is unauthorized.
```

Ce n'est pas élégant mais efficace !

Les pages d'erreur

On va en profiter pour améliorer l'affichage des erreurs comme celle vue ci-dessus.

On créer un dossier spécifique et un layout :



Avec ce code (inspiré de [cet exemple de Bootstrap 4](#)) :

```
@extends('layouts.app')  
  
@section('css')
```

```
<style>

  html,
  body {
    height: 100%;
  }
  body {
    color: white;
    text-align: center;
  }
  .site-wrapper {
    display: table;
    width: 100%;
    height: 100%;
    min-height: 100%;
  }
  .site-wrapper-inner {
    display: table-cell;
    vertical-align: middle;
    margin-right: auto;
    margin-left: auto;
    width: 100%;
    padding: 0 1.5rem;
  }

</style>
```

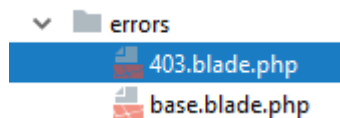
@endsection

@section('content')

```
<div class="site-wrapper">
  <main role="main" class="site-wrapper-inner">
    <h1>@yield('title')</h1>
    <p class="lead">@yield('text')</p>
  </main>
</div>
```

@endsection

On ajoute une vue **403** :



Avec ce code :

```
@extends('errors.base')
```

```
@section('title')
    @lang('Erreur 403')
@endsection
```

```
@section('text')
    @lang("Vos droits d'accès ne vous permettent pas d'accéder à
    cette ressource")
@endsection
```

On a maintenant quelque chose de plus joli :

Erreur 403

Vos droits d'accès ne vous permettent pas d'accéder à cette ressource

On va ajouter aussi **404** :

```
@extends('errors.base')
```

```
@section('title')
    @lang('Erreur 404')
@endsection
```

```
@section('text')
    @lang("Cette page n'existe pas")
@endsection
```

Erreur 404

Cette page n'existe pas

Et **503** :


```
@extends('errors.base')
```

```
@section('title')
  @lang('Erreur 503')
@endsection
```

```
@section('text')
  @lang("Service temporairement indisponible ou en maintenance")
@endsection
```

Erreur 503

Service temporairement indisponible ou en maintenance

Les images orphelines

Lorsqu'on supprime une image tel qu'on l'a fait ci-dessus ça a pour effet de supprimer la ligne dans la table images mais les deux versions de la photo (haute et basse résolution) restent sur le disque. Ce n'est pas vraiment gênant mais ça pourrait le devenir en cas de nombreuses suppression et puis ça serait quand même plus élégant de s'en occuper.

On pourrait ajouter cette action systématiquement quand on supprime une photo mais j'ai préféré créer une partie maintenance réservée à l'administrateur.

On va créer deux nouvelles routes :

```
Route::middleware('admin')->group(function () {
    ...

    Route::name('maintenance.index')->get('maintenance',
'AdminController@index');
    Route::name('maintenance.destroy')->delete('maintenance',
'AdminController@destroy');

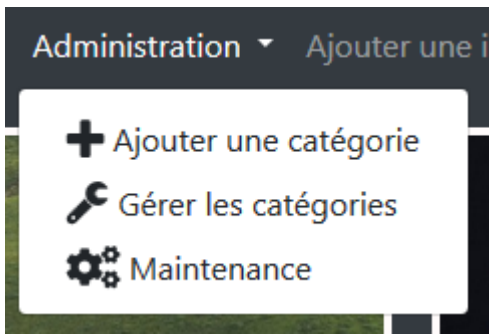
});
```

On ajoute un item au menu de l'administration (**views/layouts/app**)

:

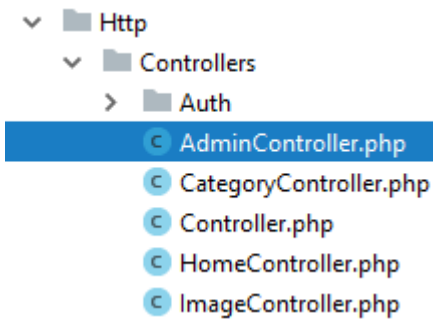
```
@admin
  <li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle{{ currentRoute(
      route('category.create'),
      route('category.index'),
      route('category.edit',
request()->category),
      route('maintenance.index')
    )}}" href="#" id="navbarDropdownGestCat"
role="button" 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>
      <a class="dropdown-item" href="{{
route('maintenance.index') }}">
        <i class="fas fa-cogs fa-lg"></i>
@lang('Maintenance')
      </a>
    </div>
  </li>
@endadmin
```

Et on a le nouvel item :



On crée un nouveau contrôleur :

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



On va utiliser **ImageRepository** et conserver les fonctions **index** et **destroy** :

```
<?php
```

```
namespace App\Http\Controllers;
```

```
use App\Repositories\ImageRepository;
```

```
class AdminController extends Controller
{
```

```
    protected $repository;
```

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

```
    public function index()
    {
        //
    }
```

```
    public function destroy($id)
```

```

    {
        //
    }
}

```

Affichage des orphelines

Pour l'affichage des orphelines on code la méthode **index** :

```

public function index()
{
    $orphans = $this->repository->getOrphans ();
    $countOrphans = count($orphans);

    return view('maintenance.index', compact ('orphans',
'countOrphans'));
}

```

Et dans **ImageRepository** :

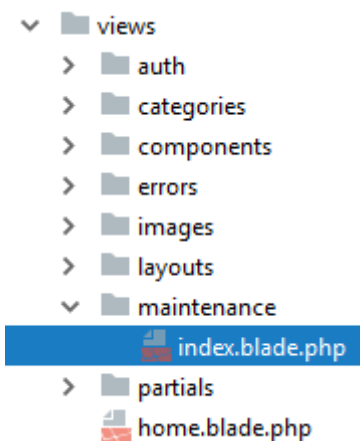
```

public function getOrphans()
{
    $files = collect(Storage::disk('images')->files());
    $images = Image::select('name')->get()->pluck('name');
    return $files->diff($images);
}

```

On en profite pour voir la puissance des collections de Laravel !

On crée la vue :



Avec ce code :

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

```
@section('content')
```

```
    <main class="container-fluid">
        <h1>
            {{ $countOrphans }} {{ trans_choice(__('image
orpheline|images orphelines'), $countOrphans) }}
            @if($countOrphans)
                <a class="btn btn-danger pull-right" href="{{
route('maintenance.destroy') }}"
role="button">@lang('Supprimer')</a>
            @endif
        </h1>

        <div class="card-columns">
            @foreach($orphans as $orphan)
                <div class="card">
                    
                </div>
            @endforeach
        </div>
    </main>
```

```
@endsection
```

```
@section('script')
```

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

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

            $('a.btn-danger').click(function(e) {
                let that = $(this)
                e.preventDefault()
                swal({
                    title: '@lang('Vraiment supprimer toutes les
photos orphelines ?')',
```

```

        type: 'warning',
        showCancelButton: true,
        confirmButtonColor: '#DD6B55',
        confirmButtonText: '@lang('Oui')',
        cancelButtonText: '@lang('Non')'
    }).then(function () {
        $.ajax({
            url: that.attr('href'),
            type: 'DELETE'
        })
        .done(function () {
            location.reload();
        })
        .fail(function () {
            swal({
                title: '@lang('Il semble y avoir
une erreur sur le serveur, veuillez réessayer plus tard...')',
                type: 'warning'
            })
        })
    })
})
</script>

```

@endsection

5 images orphelines

Supprimer

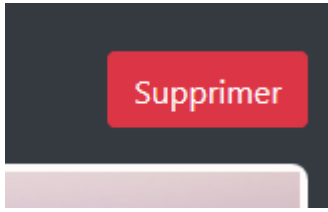






Suppression des orphelines

On a un bouton pour la suppression :



On code **AdminController** :

```
public function destroy()
{
    $this->repository->destroyOrphans ();

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

Et **ImageRepository** :

```
public function destroyOrphans()
{
    $orphans = $this->getOrphans ();

    foreach($orphans as $orphan) {
        Storage::disk('images')->delete($orphan);
        Storage::disk('thumbs')->delete($orphan);
    }
}
```

Dans la vue on a une alerte :



Si on supprime on se retrouve avec ça :

0 image orpheline

Remarquez la gestion du pluriel au niveau de la vue :

```
__('image orpheline|images orphelines')
```

Conclusion

Dans ce chapitre on a :

- affiché les photos par utilisateur
- codé la suppression des photos en prévoyant une autorisation
- ajouté les pages d'erreurs les plus usuelles
- ajouté la gestion des images orpheline dans l'administration

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