Laravel 5.7 par la pratique – La galerie

Maintenant qu'on a créé l'essentiel de la gestion des catégories et des images on va enfin passer à la réalisation de la galerie elle-même pour visualiser les images sur la page d'accueil. On va aussi donner la possibilité de zoomer les images. On va prévoir une pagination, un affichage par catégorie…

Une light box

Quand on fait une recherche sur Internet pour une light box on a surtout l'embarras du choix. je me suis décidé pour Magnific Popup qui est simple, rapide et esthétique. On l'installe avec **npm** :

```
npm i magnific-popup -D
```

Il faut importer dans **resources/sass/app.scss** :

```
@import '~magnific-popup/dist/magnific-popup.css';
```

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

```
try {
    ...
    require('magnific-popup');
} catch (e) {}
```

Et relance **npm run dev**.

Pour l'utilisation on trouve une documentation assez détaillée sur le site.

Une nouvelle directive Blade

On a déjà créé une directive Blade pour filtrer des administrateurs. On va en ajouter une pour filtrer « owner ou admin ». On ajoute ce code dans **AppServiceProvider** :

```
public function boot()
{
    ...

    Blade::if ('adminOrOwner', function ($id) {
         return auth ()->check () && (auth ()->id () === $id ||
auth ()->user ()->admin);

   ...
}
```

On pourra ainsi écrire **@adminOrOwner** dans les vues ! Ça va nous servir pour autoriser la suppression des images.

Les réglages utilisateur

On va ajouter un champ **settings** dans la table **users** pour pouvoir personnaliser le statut adulte et la pagination. Pour prévoir l'avenir on choisit un champ JSON :

```
$table->json('settings');
```

On va aussi compléter le **seeder** pour renseigner ce champ :

```
class UsersTableSeeder extends Seeder
{

    public function run()
    {
        User::create([
            'name' => 'Durand',
            'email' => 'durand@chezlui.fr',
            'role' => 'admin',
            'password' => bcrypt('admin'),
            'settings' => '{"pagination": 8, "adult": true}',
            'email_verified_at' => Carbon::now(),
        ]);

        User::create([
            'name' => 'Dupont',
            'email' => 'dupont@chezlui.fr',
            'password' => bcrypt('user'),
            'settings' => '{"pagination": 8, "adult": true}',
```

```
            'email_verified_at' => Carbon::now(),
        ]);

        User::create([
            'name' => 'Martin',
            'email' => 'martin@chezlui.fr',
            'password' => bcrypt('user'),
            'settings' => '{"pagination": 8, "adult": true}',
            'email_verified_at' => Carbon::now(),
        ]);
    }
}
```

On relance la migration et la population :

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

Il faut actualiser le contrôleur **RegisterController** :

```
protected function create(array $data)
{
    return User::create([
        'name' => $data['name'],
        'email' => $data['email'],
        'password' => Hash::make($data['password']),
        'settings' => '{"pagination": 8, "adult": false}',
    ]);
}
```

Et le modèle **User** :

```
protected $fillable = [
    'name', 'email', 'password', 'settings'
];
```

La vue pour la galerie

Pour l'affichage des vignettes de photos on va utiliser encore le composant card de Bootstrap 4 avec l'intéressante possibilité de les afficher automatiquement en colonnes sans effort !

Voilà le nouveau code de la vue **views/home** :

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

```
@section('content')

    <main class="container-fluid">
        @isset($category)
            <h2 class="text-title mb-3">{{ $category->name }}</h2>
        @endif
        @isset($user)
             <h2 class="text-title mb-3">{{ __('Photos de ') .
$user->name }}</h2>
        @endif
        <div class="d-flex justify-content-center">
            {{ $images->links() }}
        </div>
        <div class="card-columns">
            @foreach($images as $image)
                <div class="card @if($image->adult) border-danger
@endif" id="image{{ $image->id }}">
                    <a href="{{ url('images/' . $image->name) }}"
class="image-link">
                        <img class="card-img-top"
                             src="{{ url('thumbs/' . $image->name)
}}"
                             alt="image">
                    </a>
                    @isset($image->description)
                        <div class="card-body">
                                    <p class="card-text">{{
$image->description }}</p>
                        </div>
                    @endisset
                    <div class="card-footer text-muted">
                        <em>
                            <a href="#" data-toggle="tooltip"
                               title="{{ __('Voir les photos de ')
. $image->user->name }}">{{ $image->user->name }}</a>
                        </em>
                        <div class="pull-right">
                            <em>
                                                                {{
$image->created_at->formatLocalized('%x') }}
                            </em>
                        </div>
                    </div>
```

```
                </div>
            @endforeach
        </div>
        <div class="d-flex justify-content-center">
            {{ $images->links() }}
        </div>
    </main>

@endsection

@section('script')

    <script>
        $(() => {

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

            $('.card-columns').magnificPopup({
                delegate: 'a.image-link',
                type: 'image',
                                        tClose:  '@lang("Fermer
(Esc)")'@if($images->count() > 1),
                gallery: {
                    enabled: true,
                    tPrev: '@lang("Précédent (Flèche gauche)")',
                    tNext: '@lang("Suivant (Flèche droite)")'
                },
                callbacks: {
                    buildControls: function () {
this.contentContainer.append(this.arrowLeft.add(this.arrowRight))
                    }
                }@endif
            })

        })
    </script>
@endsection
```

La gestion

Le contrôleur HomeController

On va compléter le code du contrôleur **HomeController** pour envoyer les images dans la vue :

```
<?php

namespace App\Http\Controllers;

use App\Repositories\ImageRepository;

class HomeController extends Controller
{
    public function index(ImageRepository $repository)
    {
        $images = $repository->getAllImages ();

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

On voit qu’on délègue la gestion des données au repository.

Le repository ImageRepository

On complète ainsi son code :

```
public function getAllImages()
{
       return  Image::latestWithUser()->paginate  (config
('app.pagination'));
}
```

La pagination par défaut est réglée dans la configuration (**config/app.php**) :

```
'pagination' => 8,
```

Un scope dans le modèle Image

Dans le repository on fait appel au modèle **Image** avec un **scope** :

```
public function scopeLatestWithUser($query)
```

```
{
    $user = auth()->user();

    if($user && $user->adult) {
        return $query->with ('user')->latest ();
    }

    return $query->with ('user')->whereAdult(false)->latest ();
}
```

Un accesseur dans le modèle User

Dans le scope ci-dessus on demande l'attribut **adult** du modèle **User**, cet attribut n'existe pas, on doit créer un accesseur (et tant qu'à faire un second pour faire propre) :

```
public function getAdultAttribute()
{
    return $this->settings->adult;
}

public function getSettingsAttribute($value)
{
    return json_decode ($value);
}
```

Le code est ainsi bien organisé !

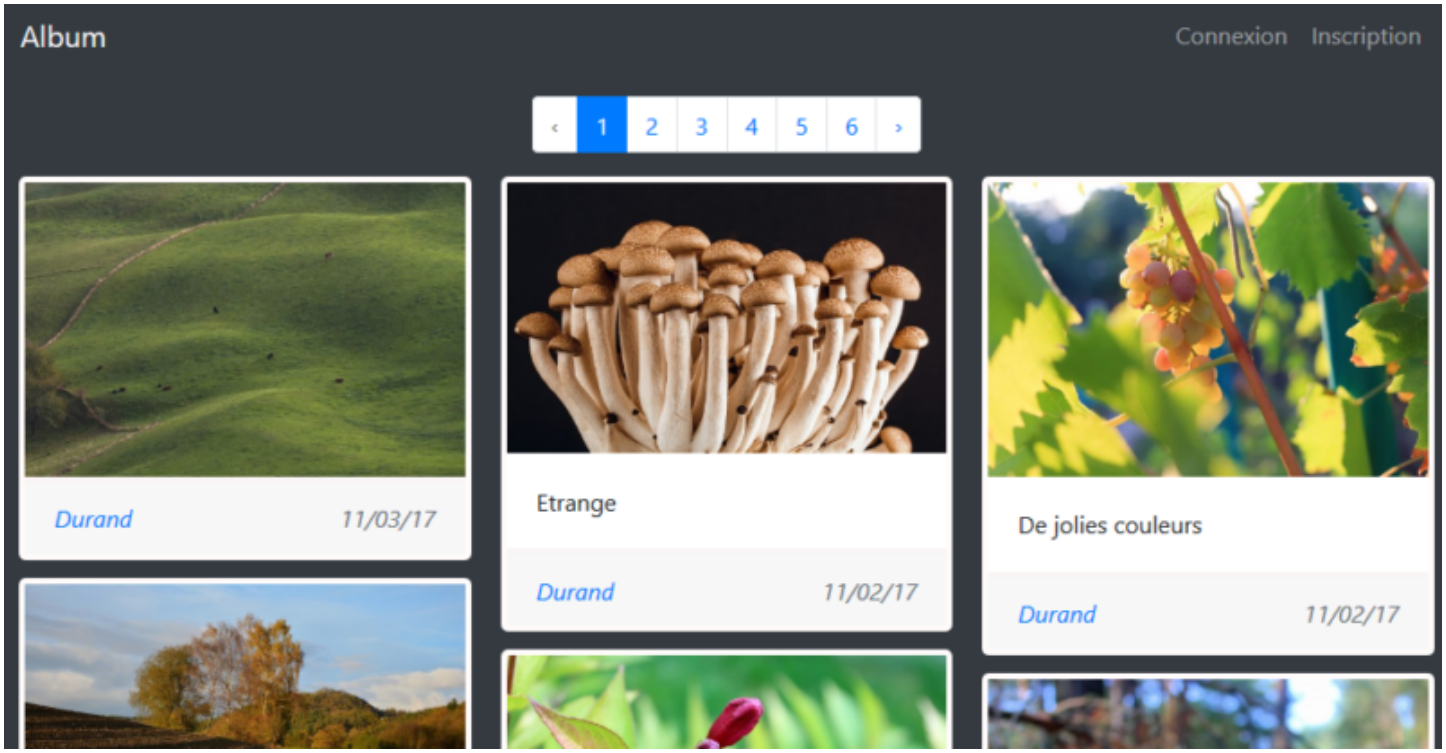
Les routes

On avait précédemment ajouté le middleware **verified** à la route home. On va maintenant l'enlever pour rendre la galerie accessible pour tout le monde et réserver cette vérification pour l'accès aux autres fonctionnalités. :

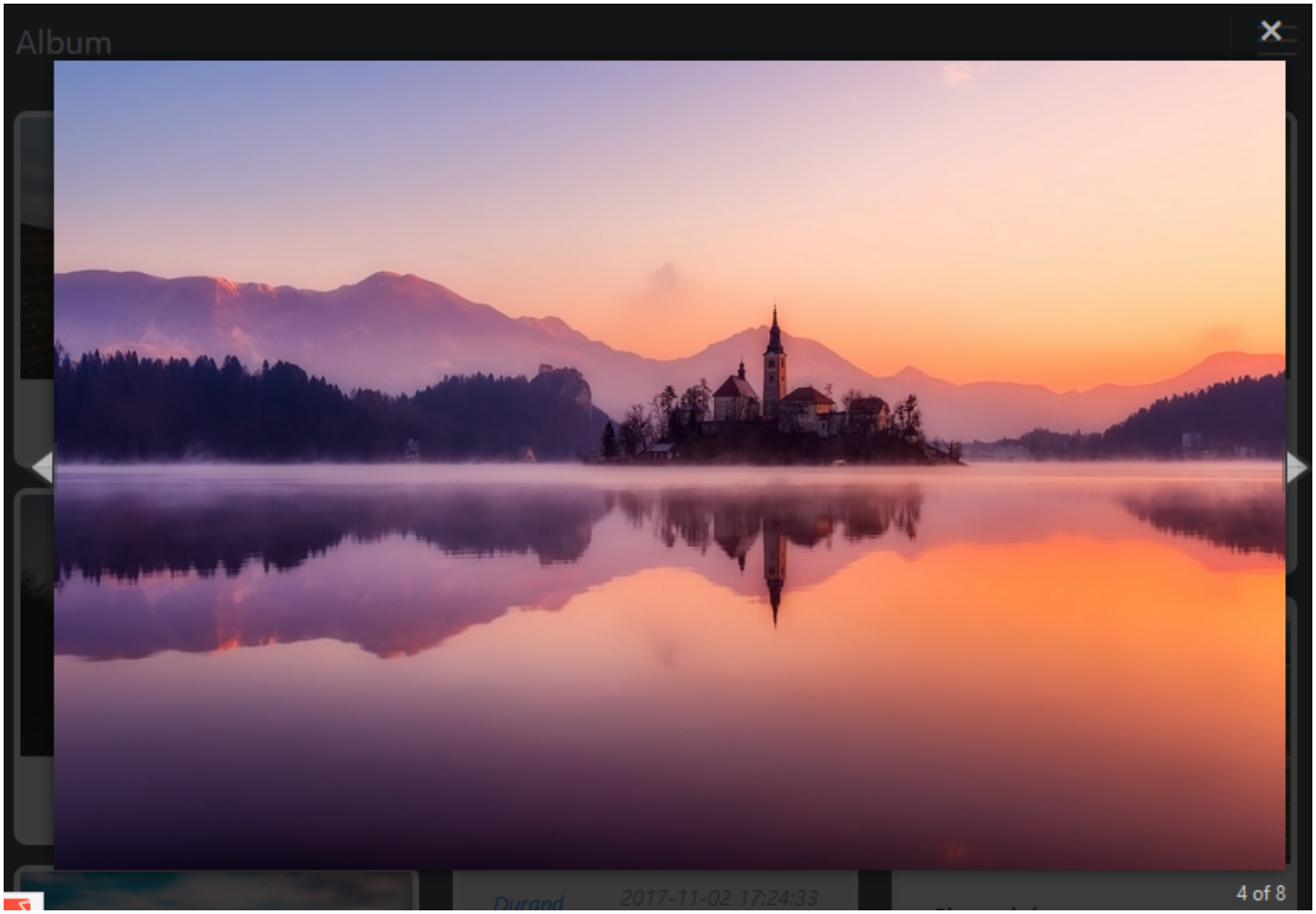
```
Route::get('/', 'HomeController@index')->name('home');
```

La galerie

On peut enfin voir la galerie :



Vérifiez que la lightbox fonctionne en cliquant sur une image :



Le nom de celui qui a envoyé la photo apparaît. Pour le moment le lien n’est pas actif, on codera ça plus tard.

Le colonnes

Il y a quand même un petit souci avec l'organisation des colonnes, on a que deux situations : 3 colonnes pour les écrans larges et une colonne quand ça se réduit vraiment. Il faudrait apporter plus de finesse à ce réglage. Dans **resources/sass/app.scss** ajoutez ce code :

```
.card-columns {
    @include media-breakpoint-only(sm) {
      column-count: 2;
    }
    @include media-breakpoint-only(lg) {
      column-count: 3;
    }
    @include media-breakpoint-only(xl) {
      column-count: 4;
    }
}
```

Relancez la compilation (**npm run dev**).

On a maintenant 4 situations avec 1, 2, 3 ou 4 colonnes, ce qui est bien plus esthétique !

Les transitions

Un autre aspect n'est pas très élégant : comme il faut du temps pour charger les images c'est assez saccadé quand on change de page ou qu'on arrive sur la galerie. Ce qui serait bien c'est de faire apparaître une icône mobile d'attente jusqu'à ce que la page soit totalement chargée.

On va donc recouvrir toute la page avec l'icône mobile au milieux. On ajoute dans la vue home ce code :

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

@section('content')

    <div class="site-wrapper">
        <div class="site-wrapper-inner text-white text-center">
```

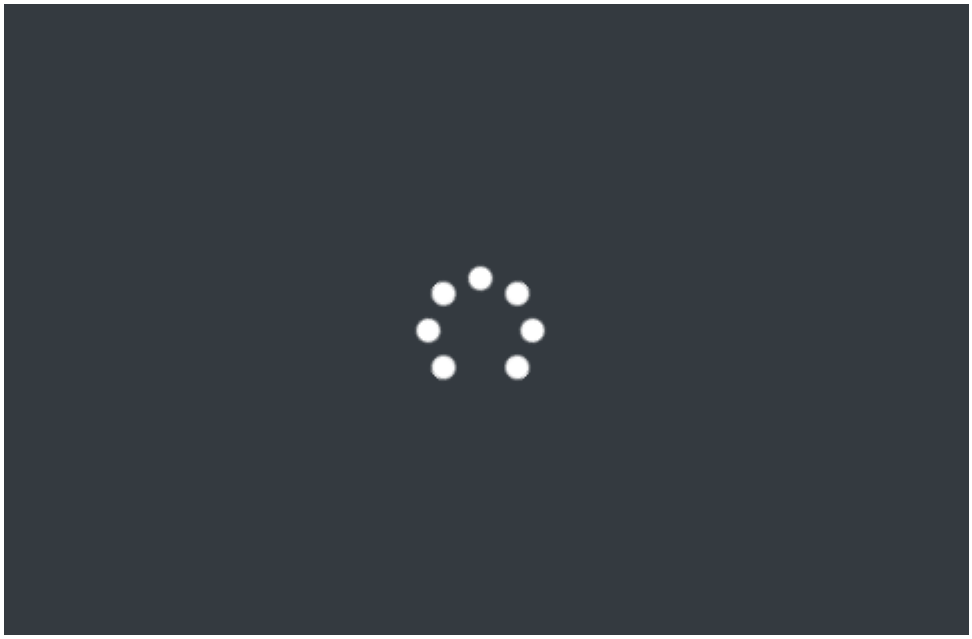
```
            <i class="fas fa-spinner fa-pulse fa-4x"></i>
        </div>
    </div>

    ...

    <script>
        $(() => {

            $('.site-wrapper').fadeOut(1000)

            ...

        })
    </script>
@endsection
```

Et un peu de style pour le recouvrement dans **resources/sass/app.scss** :

```
html, body {
    height: 100%;
}
.site-wrapper {
    display: grid;
    height: 100%;
    grid-template-columns: 1fr;
    grid-template-rows: 1fr;
    grid-template-areas: "center";
  }
  .site-wrapper-inner {
    grid-area: center;
    align-self: center;
}
```

J’ai adopté le **Grid Layout** qui est maintenant bien supporté par les navigateurs.

On relance la compilation et maintenant on a l’icône d’attente au milieu d’un écran noir le temps du chargement :



L’affichage par catégorie

Le contrôleur

Pour l’affichage par catégorie on va commencer par ajouter la route :

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

Ajouter cette fonction dans le contrôleur **ImageController** :

```
...

use App\Repositories\ {
    ImageRepository, CategoryRepository
};

class ImageController extends Controller
{
    ...

    protected $categoryRepository;

    public function __construct(
        ImageRepository $imageRepository,
        CategoryRepository $categoryRepository)
```

```
    {
    {
        $this->imageRepository = $imageRepository;
        $this->categoryRepository = $categoryRepository;
    }

    ...

    public function category($slug)
    {
        $category = $this->categoryRepository->getBySlug ($slug);
         $images = $this->imageRepository->getImagesForCategory
($slug);

        return view ('home', compact ('category', 'images'));
    }
}
```

Là on délègue encore le traitement aux repositories.

Les repositories

Dans le repository **ImageRepository** :

```
public function getImagesForCategory($slug)
{
    return Image::latestWithUser()->whereHas('category', function
($query) use ($slug) {
        $query->whereSlug($slug);
    })->paginate(config('app.pagination'));
}
```

En ce qui concerne le repository **CategoryRepository** on a déjà tout ce qu'il faut.

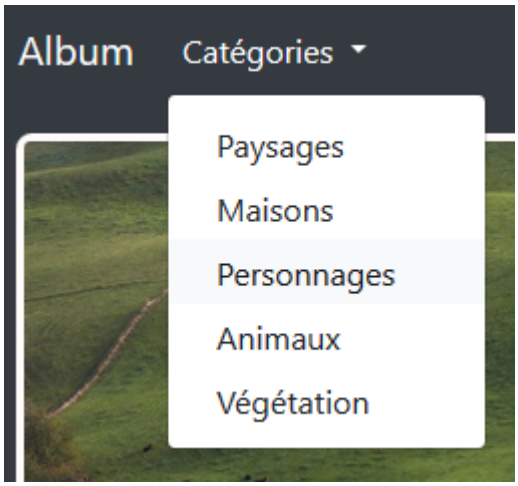
La vue

On ajoute un menu déroulant dans la barre de navigation :

```
<li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle
        @isset($category)
            {{ currentRoute(route('category', $category->slug)) }}
```

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

Et normalement :



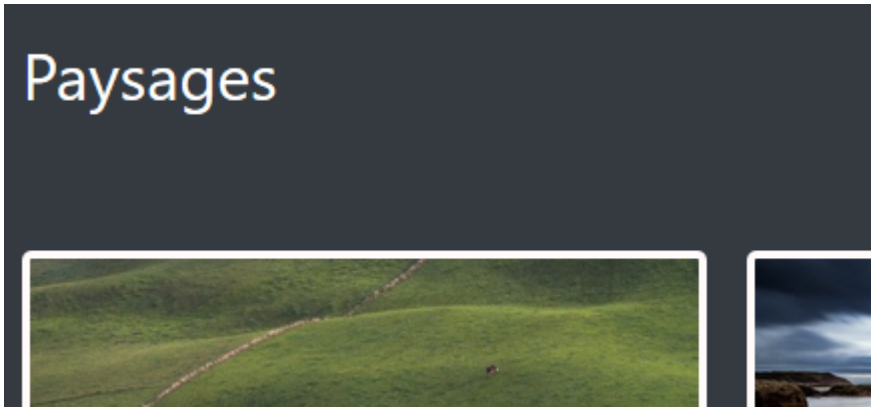
On affiche bien les images par catégorie mais le nom de la catégorie n’est pas visible…

Ajoutez ce code dans **resources/assets/css/app.css** :

```
.text-title, h1 {
    color: white;
}
```

Et relancez npm.

Le nom de la catégorie doit être maintenant visible :



On verra dans le prochain chapitre l’affichage par utilisateur, la suppression des images, et bien d’autres choses !

Conclusion

Dans ce chapitre on a :

- ajouté une lighbox pour la visualisation des images
- ajouté une directive Blade
- créé la route et la vue pour la galerie
- ajouté la gestion de la galerie dans le contrôleur et les repositories
- ajouté un écran d’attente de chargement de la page
- ajouté un menu pour les catégories
- écrit le code pour l’affichage par catégorie

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