Laravel 5.7 par la pratique – L'administration

La galerie est maintenant bien avancée. Les utilisateurs peuvent gérer des albums personnels, changer leur profil, modifier toutes les caractéristiques de leurs images. Dans ce chapitre nous allons mettre en place des outils d'administrations : suppression des images orphelines, galerie en mode maintenance et gestion des utilisateurs.

Les images orphelines

Lorsqu'on supprime une image 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 suppressions 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.

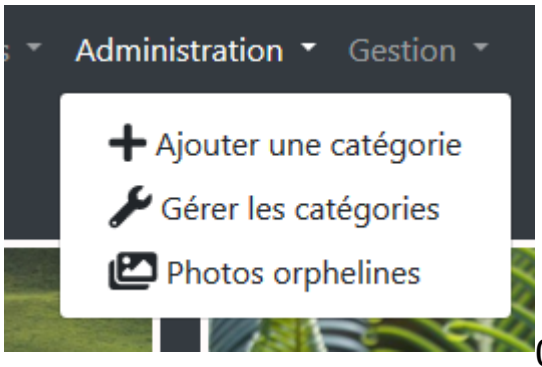
On va créer deux nouvelles routes :

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

    ...

    Route::name ('orphans.')->prefix('orphans')->group(function ()
{
                    Route::name  ('index')->get  ('/',
'AdminController@orphans');
                Route::name  ('destroy')->delete  ('/',
'AdminController@destroy');
    });
});
```

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

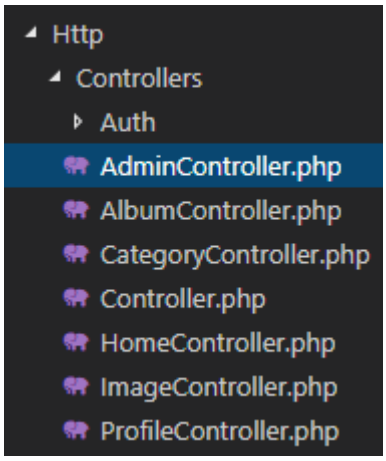
:

```
@admin
<li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle{{ currentRoute(
                            route('category.create'),
                            route('category.index'),
                                    route('category.edit',
request()->category?: 0),
                            route('orphans.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('orphans.index')
}}">
             <i class="fas fa-images fa-lg"></i> @lang('Photos
orphelines')
        </a>
    </div>
</li>
@endadmin
```



On crée un nouveau contrôleur :

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



Affichage des orphelines

Pour l'affichage des orphelines on crée une méthode **orphans** dans **AdminController** et on déclare le repository **ImageRepository** :

```
<?php

namespace App\Http\Controllers;

use App\Repositories\ImageRepository;
use Illuminate\Http\Request;

class AdminController extends Controller
{
    protected $repository;

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

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

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

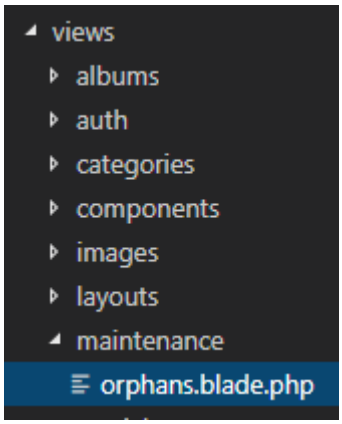
Et dans **ImageRepository** :

```
public function getOrphans()
```

```
{
    return collect (Storage::files ('images'))->transform(function
($item) {
        return basename($item);
    })->diff (Image::select ('name')->pluck ('name'));
}
```

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

On crée la vue :



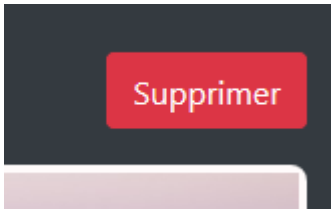
Avec ce code :

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

@section('content')

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

        <div class="card-columns">
            @foreach($orphans as $orphan)
                <div class="card">
                    <img class="img-fluid" src="{{ url('thumbs/' .
$orphan) }}" alt="image">
                </div>
            @endforeach
        </div>
```

```
    </main>

@endsection

@section('script')

    @include('partials.script-delete', ['text' => __('Vraiment supprimer toutes les photos orphelines ?'), 'return' => 'reload'])

@endsection
```

Suppression des orphelines

On a un bouton pour la suppression :



On code **AdminController** :

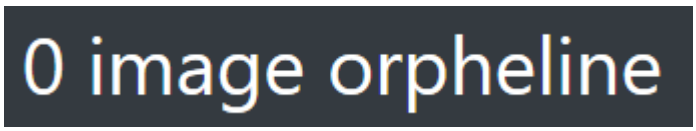
```
    ...

public function __construct(ImageRepository $repository)
{
```

```
    $this->repository = $repository;
    $this->middleware('ajax')->only('destroy');
}

public function destroy()
{
    $this->repository->destroyOrphans ();
    return response ()->json ();
}


    ...
```

Et **ImageRepository** :

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

    foreach ($orphans as $orphan) {
        Storage::delete ([
            'images/' . $orphan,
            'thumbs/' . $orphan,
        ]);
    }
}
```

Si on supprime on se retrouve avec ça :



Remarquez la gestion du pluriel au niveau de la vue :

```
{{  trans_choice(__('image  orpheline|images  orphelines'), $orphans->count) }}
```

Le mode maintenance

Maintenant voyons comment mettre en place le code pour pouvoir mettre notre galerie en maintenance et qu'ainsi elle ne soit plus accessible que par l'administrateur en se fondant sur son adresse IP.

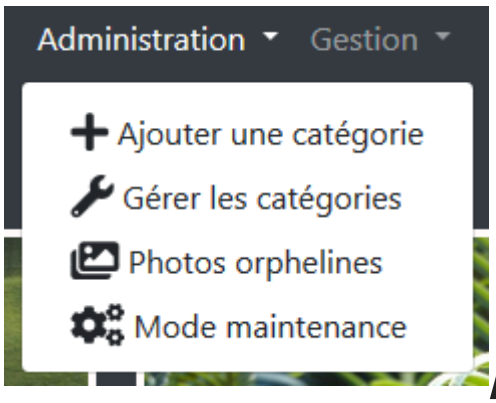
On va créer deux nouvelles routes :

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

    ...

                                                    Route::name
('maintenance.')->prefix('maintenance')->group(function () {
        Route::name ('index')->get ('/', 'AdminController@edit');
                        Route::name   ('update')->put   ('/',
'AdminController@update');
    });

});
```

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?: 0),
                            route('orphans.index'),
                            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('orphans.index') }}">
            <i class="fas fa-images fa-lg"></i> @lang('Photos orphelines')
        </a>
                    <a class="dropdown-item" href="{{ route('maintenance.index') }}">
            <i class="fas fa-cogs fa-lg"></i> @lang('Mode maintenance')
        </a>
    </div>
</li>
@endadmin
```



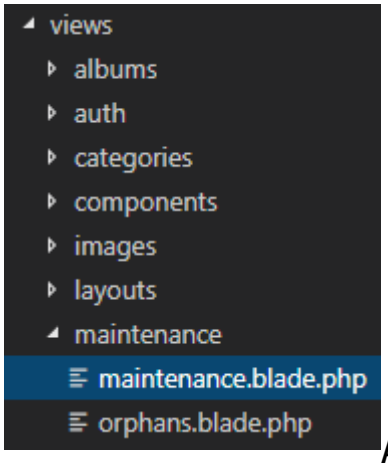
Affichage du formulaire

On ajoute cette méthode dans **AdminController** :

```
use Symfony\Component\HttpFoundation\IpUtils;
use Illuminate\Contracts\Foundation\Application;

    ...

public function edit(Request $request, Application $app)
{
    $maintenance = $app->isDownForMaintenance();
    $ipChecked = true;
    $ip = $request->ip();

    if($maintenance) {
        $data = json_decode(file_get_contents($app->storagePath().'/framework/down'), true);
        $ipChecked = isset($data['allowed']) && IpUtils::checkIp($ip, (array) $data['allowed']);
    }
```

```
        return view ('maintenance.maintenance', compact
('maintenance', 'ip', 'ipChecked'));
}
```

On crée la vue :

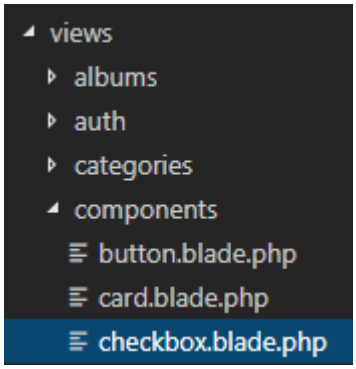


Avec ce code :

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

@section('card')

    @component('components.card')

        @slot('title')
            @lang('Mode maintenance')
        @endslot

        <form method="POST" action="{{ route('maintenance.update')
}}">
            @csrf
            @method('PUT')

            @component('components.checkbox', [
                    'name' => 'maintenance',
                    'label' => __('Mode maintenance'),
                    'checked' => $maintenance ? 'checked' : ''
                ])
            @endcomponent

            @component('components.checkbox', [
                    'name' => 'ip',
                    'label' => __('Autoriser mon IP ') . '(' . $ip
. ')',
                    'checked' => $ipChecked ? 'checked' : ''
```

```
            ])
        @endcomponent

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

    </form>

@endcomponent

@endsection
```

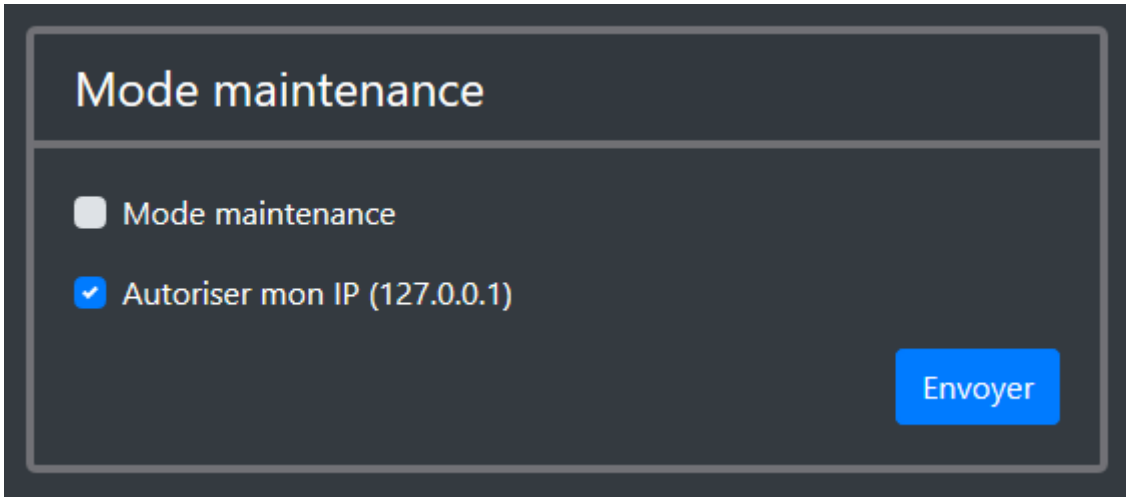
On ajoute un composant pour les cases à cocher :



Avec ce code :

```
<div class="form-group">
    <div class="custom-control custom-checkbox">
        <input type="checkbox" class="custom-control-input" id="{{
$name }}" name="{{ $name }}" @if($checked) checked @endif>
        <label class="custom-control-label" for="{{ $name }}"> {{
$label }}</label>
    </div>
</div>
```

Et le formulaire devrait apparaître :

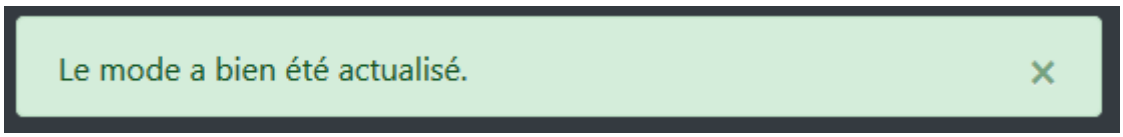


Traitement du formulaire

On ajoute la méthode **update** dans **AdminController** :

```
use Illuminate\Support\Facades\Artisan;

    ...

public function update(Request $request)
{
    if($request->maintenance) {
         Artisan::call ('down', $request->ip ? ['--allow' =>
$request->ip()] : []);
    } else {
        Artisan::call ('up');
    }

    return redirect()->route('maintenance.index')->with ('ok', __
('Le mode a bien été actualisé.'));
}
```

Une alerte nous informe du succès de l'opération :



Une alerte

Maintenant ce qui serait bien serait de disposer de quelque chose qui nous rappelle qu'on est en mode maintenance parce qu'on risque

d’oublier !

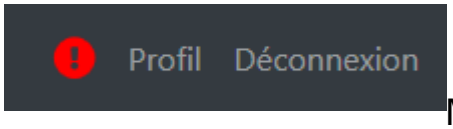
On va créer une nouvelle commande Blade dans **AppServiceProvider** :

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

    Blade::if ('maintenance', function () {
         return auth ()->check () && auth ()->user ()->admin &&
app()->isDownForMaintenance();
    });

    ...

}
```

Et on ajoute une petite icône dans **layouts.app** :

```
<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
        @maintenance
            <li class="nav-item">
                            <a  class="nav-link"  href="{{
route('maintenance.index')   }}"   data-toggle="tooltip"
title="@lang('Mode maintenance')">
                    <span class="fas fa-exclamation-circle  fa-lg"
style="color: red;">

                    </span>
                </a>
            </li>
        @endmaintenance
        <li class="nav-item{{ currentRoute(
                    route('profile.edit', auth()->id()),
                    route('profile.show', auth()->id())
                )}}">
```

```
            <a class="nav-link" href="{{ route('profile.edit',
auth()->id()) }}">@lang('Profil')</a>
        </li>
        <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>
```



Maintenant on risque moins d’oublier !

On en profite pour qu’un clic sur l’icône renvoie dans la page de maintenance.

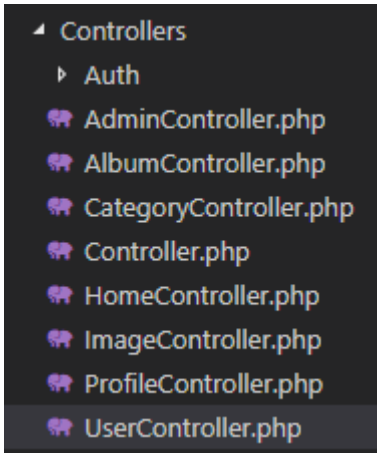
La gestion des utilisateurs

Pour gérer les utilisateurs on va encore créer des routes :

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

    ...

    Route::resource ('user', 'UserController', [
        'only' => ['index', 'edit', 'update', 'destroy']
    ]);

    ...

});
```

On crée le contrôleur associé :

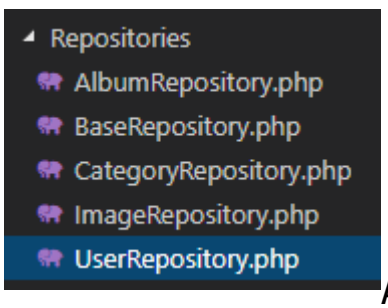
```
php artisan make:controller UserController --resource
```



Avec ce code de base :

```
<?php

namespace App\Http\Controllers;

use App\Repositories\UserRepository;
use App\Models\User;

class UserController extends Controller
{
    protected $repository;

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

On crée aussi un repository :



Avec ce code de base :

```
<?php

namespace App\Repositories;
```

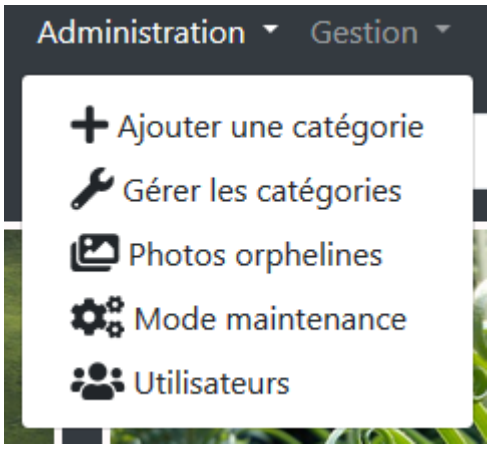
```
use App\Models\User;

class UserRepository extends BaseRepository
{
    public function __construct(User $user)
    {
        $this->model = $user;
    }
}
```

Le menu

On ajoute encore un item au menu (**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?: 0),
                            route('orphans.index'),
                            route('maintenance.index'),
                            route('user.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('orphans.index')
}}">
```

```
            <i class="fas fa-images fa-lg"></i> @lang('Photos
orphelines')
        </a>
                    <a  class="dropdown-item"  href="{{
route('maintenance.index') }}">
             <i class="fas fa-cogs fa-lg"></i> @lang('Mode
maintenance')
        </a>
        <a class="dropdown-item" href="{{ route('user.index') }}">
                    <i  class="fas  fa-users  fa-lg"></i>
@lang('Utilisateurs')
        </a>
    </div>
</li>
@endadmin
```



Afficher les utilisateurs

On va commencer par afficher une page avec la liste des utilisateurs et de leurs renseignements ainsi que des boutons pour modifier leurs données ou même les supprimer. On commence par coder **UserController** :

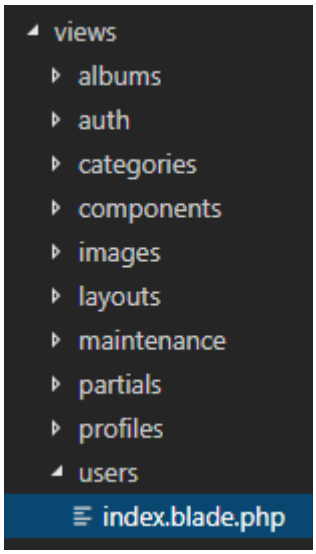
```
public function index()
{
    $users = $this->repository->getAllWithPhotosCount();
    return view ('users.index', compact('users'));
}
```

Et dans **UserRepository** :

```
public function getAllWithPhotosCount()
{
```

```
    return User::withCount('images')->oldest('name')->get();
}
```

On crée la vue :



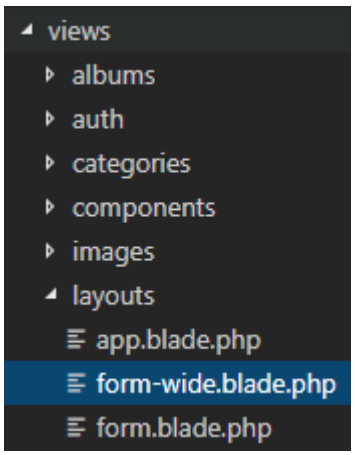
Avec ce code :

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

@section('css')

    <style>
        .fa-check { color: green; }
    </style>

@endsection


@section('card')

    @component('components.card')

        @slot('title')
             @lang('Gestion des utilisateurs (administrateurs en
rouge)')
        @endslot

        <div class="table-responsive">
            <table class="table table-dark text-white">
                <thead>
                    <tr>
                        <th scope="col">@lang('Nom')</th>
                        <th scope="col">@lang('Email')</th>
```

```
                        <th scope="col">@lang('Inscription')</th>
                        <th scope="col">@lang('Vérifié')</th>
                        <th scope="col">@lang('Adulte')</th>
                        <th scope="col">@lang('Photos')</th>
                        <th></th>
                        <th></th>
                    </tr>
                </thead>
                <tbody>
                @foreach($users as $user)
                      <tr @if($user->admin) style="color: red"
@endif>
                        <td>{{ $user->name }}</td>
                        <td>{{ $user->email }}</td>
                                                                <td>{{
$user->created_at->formatLocalized('%x') }}</td>
                          <td>@if($user->email_verified_at){{
$user->email_verified_at->formatLocalized('%x') }}@endif</td>
                         <td>@if($user->adult)<i class="fas fa-
check fa-lg"></i>@endif</td>
                        <td>{{ $user->images_count }}</td>
                        <td>
                                    <a type="button" href="{{
route('user.edit', $user->id) }}"
                                class="btn btn-warning btn-sm pull-
right mr-2 invisible" data-toggle="tooltip"
                                    title="@lang("Modifier
l'utilisateur") {{ $user->name }}"><i
                                       class="fas fa-edit fa-
lg"></i></a>
                        </td>
                        <td>
                            @unless($user->admin)
                                    <a type="button" href="{{
route('user.destroy', $user->id) }}"
                                class="btn btn-danger btn-sm pull-
right invisible" data-toggle="tooltip"
                                   title="@lang("Supprimer
l'utilisateur") {{ $user->name }}"><i
                                       class="fas fa-trash fa-
lg"></i></a>
                            @endunless
                        </td>
```

```
                    </tr>
                @endforeach
                </tbody>
            </table>
        </div>

    @endcomponent

@endsection

@section('script')

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

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

@endsection
```

Pour l’occasion on crée un nouveau layout pour ce formulaire plus large que les autres parce qu’on a beaucoup d’informations à y faire tenir :



Avec ce code :

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

@section('content')
    <div class="container py-5">
```

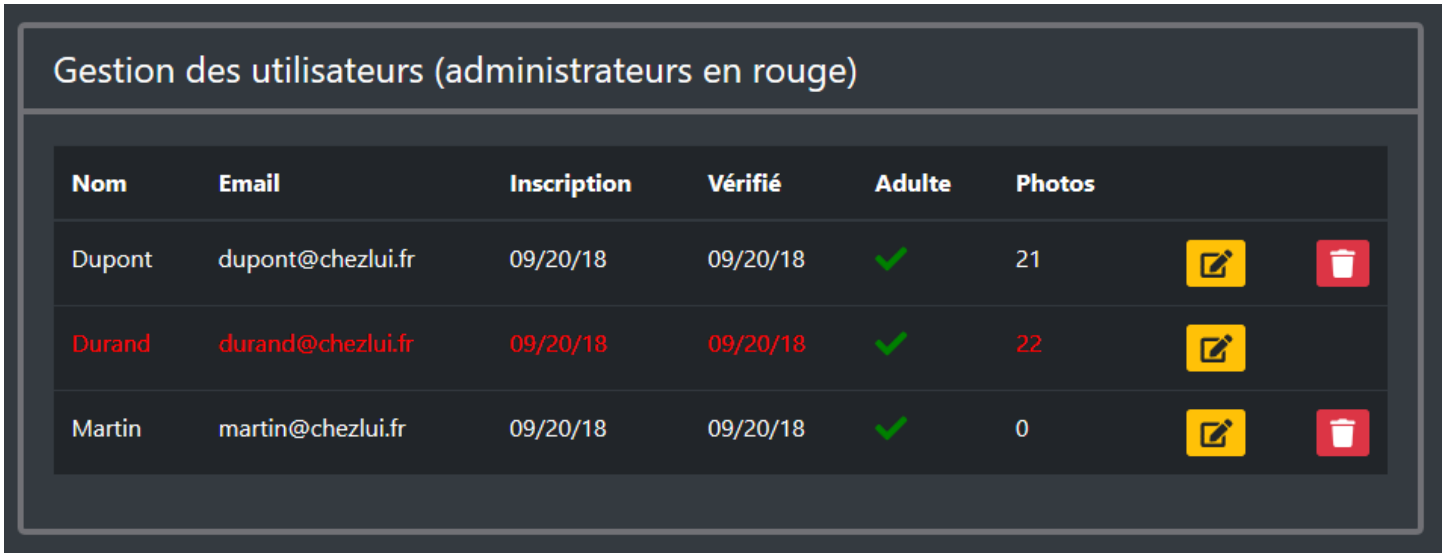
```
        <div class="row">
            <div class="col">
                @yield('card')
            </div>
        </div>
    </div>
@endsection
```

Pour afficher correctement les dates on va compléter ainsi le modèle **User** :

```
protected $dates = [
    'created_at',
    'updated_at',
    'email_verified_at',
];
```

Ainsi **email_verified_at** sera lui aussi automatiquement converti en instance de **Carbon**.

Et enfin on a la page :

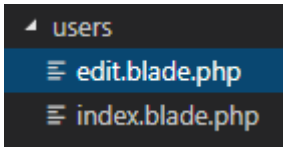


Modifier un utilisateur

On code la méthode **update** de **UserController** :

```
public function edit(User $user)
{
    return view ('users.edit', compact ('user'));
}
```

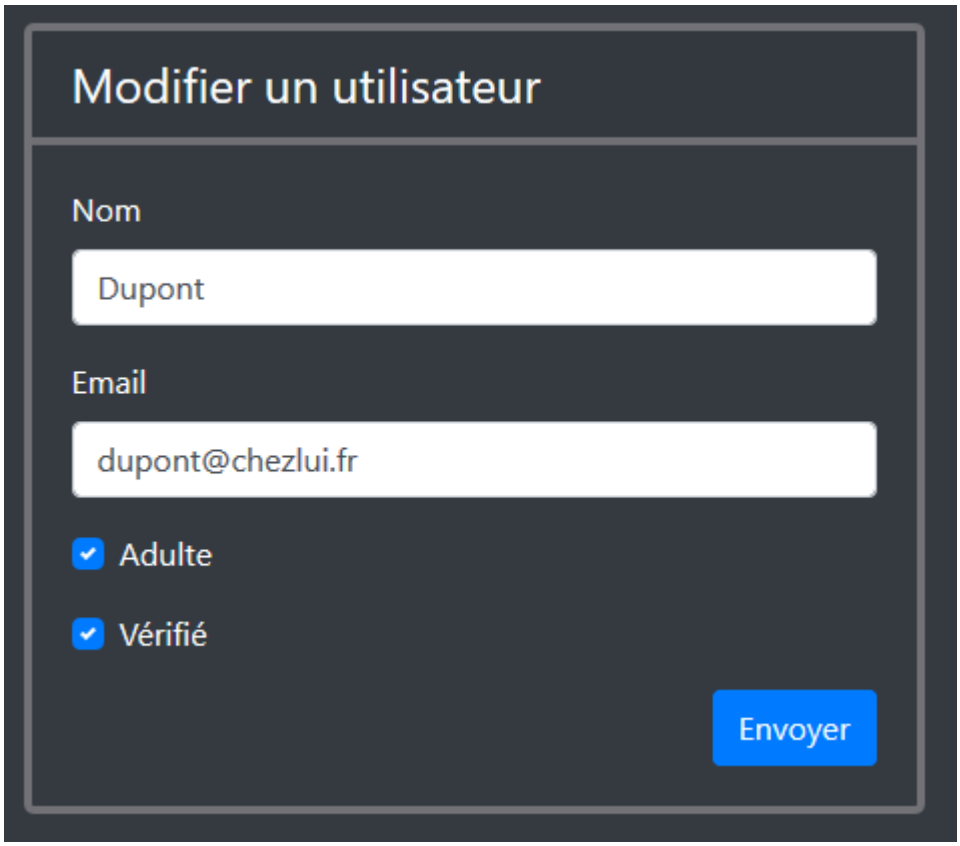
On crée la vue pour le formulaire de modification :



Avec ce code :

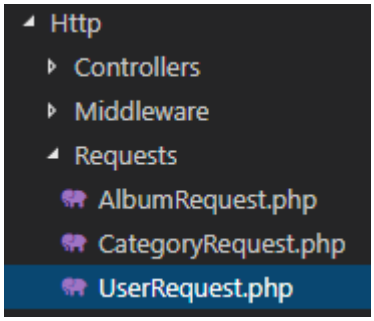
```
@extends('layouts.form')

@section('card')

    @component('components.card')

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

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

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

            @include('partials.form-group', [
                'title' => __('Email'),
                'type' => 'email',
                'name' => 'email',
                'value' => $user->email,
                'required' => true,
                ])

            <div class="form-group">
                <div class="custom-control custom-checkbox">
                    <input type="checkbox" class="custom-control-
input"  id="adult"  name="adult"  {{  $user->settings->adult  ?
```

```
'checked' : '' }}>
                        <label class="custom-control-label"
for="adult"> @lang('Adulte')</label>
                    </div>
                </div>

                <div class="form-group">
                    <div class="custom-control custom-checkbox">
                        <input type="checkbox" class="custom-control-
input" id="verified" name="verified" {{ $user->hasVerifiedEmail()
? 'checked' : '' }}>
                        <label class="custom-control-label"
for="verified"> @lang('Vérifié')</label>
                    </div>
                </div>

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

            </form>

        @endcomponent

@endsection
```



Pour la validation on crée une requête de formulaire :

hp artisan make:request UserRequest



On complète le code :

```
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

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

    public function rules()
    {
        return $rules = [
            'name' => 'required|string|max:255|unique:users,name,'
. $this->user->id,
                                                'email'    =>
'required|string|email|max:255|unique:users,email,'      .
$this->user->id,
        ];
    }
}
```

On code la méthode **update** dans **UserController** :

```
use App\Http\Requests\UserRequest;

    ...
```

```
public function update(UserRequest $request, User $user)
{
    $this->repository->update ($user, $request);

     return redirect ()->route('user.index')->with ('ok', __
("L'utilisateur a bien été modifié"));
}
```

Le traitement se fait dans **UserRepository** :

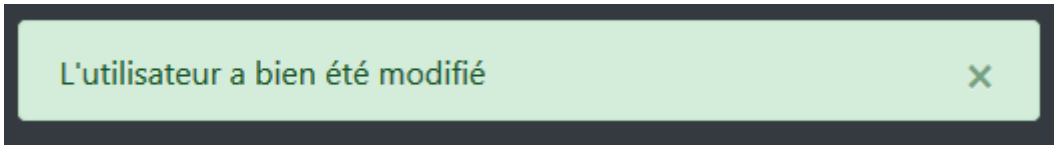
```
use Illuminate\Http\Request;
use Carbon\Carbon;

    ...

public function update(User $user, Request $request)
{
    if($user->hasVerifiedEmail() && !$request->verified) {
        $request->merge(['email_verified_at' => null]);
    }

    if(!$user->hasVerifiedEmail() && $request->verified) {
        $request->merge(['email_verified_at' => new Carbon]);
    }

    $user->adult = $request->adult;
    $user->update ($request->all());
}
```

Pour le changement du statut adulte on crée un mutateur dans le modèle **User** :

```
public function setAdultAttribute($value)
{
    $this->attributes['settings'] = json_encode ([
        'adult' => $value,
        'pagination' => $this->settings->pagination
    ]);
}
```

Ca clarifie bien la syntaxe !

Une petite alerte pour rassurer :



Supprimer un utilisateur

Pour terminer on va ajouter la possibilité de supprimer un utilisateur.

On code la méthode **destroy** dans **UserController** :

```
    ...

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

    $this->middleware('ajax')->only('destroy');
}

public function destroy(User $user)
{
    $user->delete ();

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

Evidemment on affiche une alerte avant modification :

Et si on dit oui alors il disparaît de la liste et de la base ainsi que toutes ses photos !

En résumé

Dans ce chapitre on a :

- ajouté la gestion des images orphelines
- ajouté le mode maintenance
- mis en place la gestion des utilisateurs

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