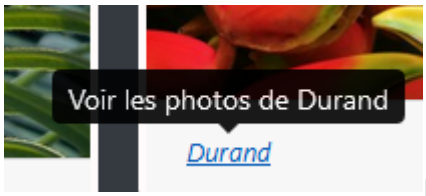
Laravel 5.7 par la pratique – gestion de ses photos

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 ajouter la possibilité de changer la description, la catégorie et le statut adulte.

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');
```

On crée la fonction dans **ImageController** :

```
use App\Models\User;

...

public function user(User $user)
{
	$images = $this->imageRepository->getImagesForUser ($user->id);
    return view ('home', compact ('user', 'images'));
}
```

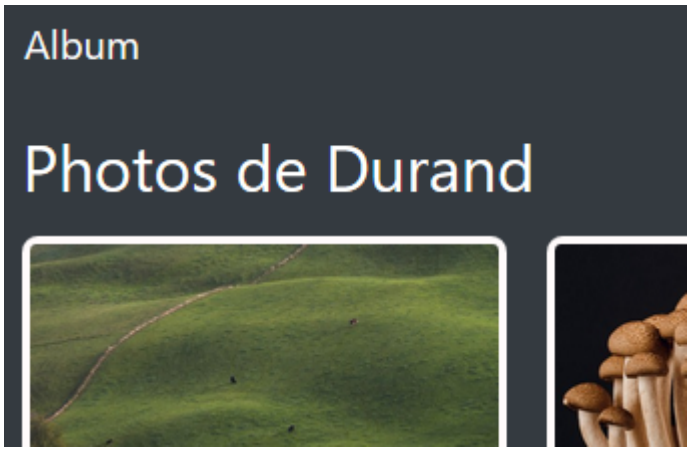
La fonction dans **ImageRepository** :

```
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**) :

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

Maintenant on peut cliquer sur le nom :



Un petit menu d’icônes

Pour la gestion des photos on va créer un petit menu d’icônes significatives. On va donc ajouter ce code dans la vue **home** :

```
...

<div class="star-rating" id="{{ $image->id }}">
    <span class="pull-right">
        @adminOrOwner($image->user_id)
            <a class="toggleIcons"
```

```
                href="#">
            <i class="fa fa-cog"></i>
            </a>
            <span class="menuIcons" style="display: none">
                <a class="form-delete text-danger"
                     href="{{ route('image.destroy', $image->id)
}}"
                    data-toggle="tooltip"
                    title="@lang('Supprimer cette photo')">
                    <i class="fa fa-trash"></i>
                </a>
                <a class="description-manage"
                    href="#"
                    data-toggle="tooltip"
                    title="@lang('Gérer la description')">
                    <i class="fa fa-comment"></i>
                </a>
                <a class="category-edit"
                    data-id="{{ $image->category_id }}"
                    href="{{ route('image.update', $image->id) }}"
                    data-toggle="tooltip"
                    title="@lang('Changer de catégorie')">
                    <i class="fa fa-edit"></i>
                </a>
                <a class="adult-edit"
                    href="#"
                    data-toggle="tooltip"
                    title="@lang('Changer de statut')">
                    <i class="fa @if($image->adult) fa-graduation-
cap @else fa-child @endif"></i>
                </a>
            </span>
             <form action="{{ route('image.destroy', $image->id)
}}" method="POST" class="hide">
                @csrf
                @method('DELETE')
            </form>
        @endadminOrOwner
    </span>
</div>

...
```

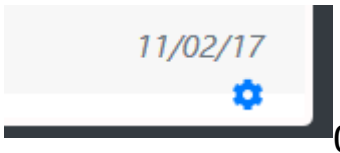
```
$('a.toggleIcons').click((e) => {
    e.preventDefault();
    let that = $(e.currentTarget)
     that.next().toggle('slow').end().children().toggleClass('fa-cog').toggleClass('fa-play')
})

...
```

L'englobement du code HTML dans la div avec la classe **star-rating** trouvera tout son sens quand on réalisera la partie notation des photos. On va aussi ajouter ce style dans **resources/sass/app.scss** :

```
a:hover {
  text-decoration: none;
}
.star-rating {
  direction: rtl;
  display: block;
}
```

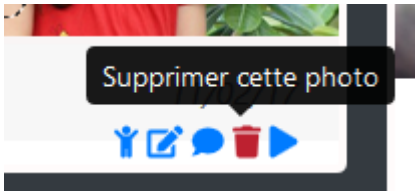
On recompile avec **npm run dev**.

Maintenant il apparaît cette icône dans le coin inférieur droit de l'image pour l'owner ou l'administrateur :



Quand on clique sur l'icône les autres icônes s'affichent en se déroulant élégamment (enfin moi je trouve…) :

Quand on survole une icône une petite aide (tooltip) apparaît :



Pour replier le menu il suffit de recliquer sur l’icône de droite.

Il ne nous reste plus qu’à coder tout ça !

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) à l’aide de l’icône qu’on a mis en place ci-dessus.

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

```
    <a class="form-delete text-danger"
        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
    @method('DELETE')
</form>
```

On va ajouter le code Javascript pour l’envoi du formulaire :

```
$('a.form-delete').click((e) => {
    e.preventDefault();
    let href = $(e.currentTarget).attr('href')
    swal({
        title: '@lang('Vraiment supprimer cette photo ?')',
        type: 'error',
        showCancelButton: true,
        confirmButtonColor: '#DD6B55',
        confirmButtonText: '@lang('Oui')',
        cancelButtonText: '@lang('Non')'
    }).then((result) => {
        if (result.value) {
```

```
            $("form[action='" + href + "']").submit()
        }
    })
})
```

Quand on clique sur la petite poubelle rouge on a une alerte :

Si on choisit « Non » rien ne se passe mais si on choisit « Oui » on envoie le formulaire.

La route existe déjà avec cette ressource :

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

On met ce code dans **ImageController** :

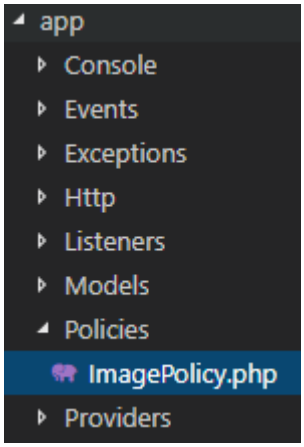
```
use App\Models\ {User, Image>};

...

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 capables 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;

    public function before(User $user)
    {
        if ($user->admin) {
            return true;
        }
    }

    public function manage(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** :

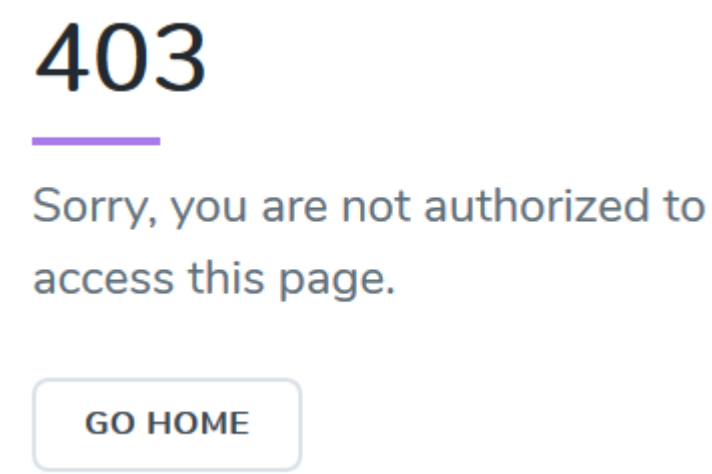
```
use App\Models\Image;
use App\Policies\ImagePolicy;

...

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

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

```
public function destroy(Image $image)
{
    $this->authorize ('manage', $image);
    $image->delete ();
    return back ();
}
```

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

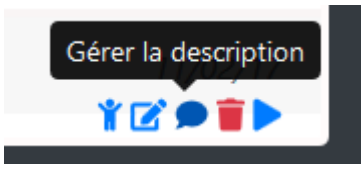
Pour vérifier que ça fonctionne supprimez la directive **@adminOrOwner** dans la vue **home**, connectez-vous avec **Dupont** tentez de supprimer une photo de **Durand** :



Au passage remarquez que Laravel dispose par défaut d'élégantes pages SVG pour les erreurs. On les harmonisera pour notre galerie plus tard.

Changer la description

On a vu qu'au téléchargement d'une image on peut écrire une brève description. IL serait bien de pouvoir modifier ensuite ce texte. On a prévu une petite icône pour ça :



On va donc écrire le code pour que ça fonctionne. Je vous propose d'utiliser une feuille modale.

Dans la vue home on a déjà le code de l'icône avec un **href** non renseigné :

```
<a class="description-manage"
    href="#"
    data-toggle="tooltip"
    title="@lang('Gérer la description')">
    <i class="fa fa-comment"></i>
</a>
```

On va renseigner ce **href** avec une route qui n'existe pas encore :

```
href="{{ route('image.description', $image->id) }}"
```

On ajoute dans **home** le code pour la feuille modale :

```
<div class="modal fade" id="changeDescription" tabindex="-1"
role="dialog" aria-labelledby="descriptionLabel"
     aria-hidden="true">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                                    <h5  class="modal-title"
id="descriptionLabel">@lang('Changement de la description')</h5>
                  <button type="button" class="close" data-
dismiss="modal" aria-label="Close">
                   <span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body">
```

```
                    <form id="descriptionForm" action=""
method="POST">
                    <div class="form-group">
                        <input type="text" class="form-control"
name="description" id="description">
                        <small class="invalid-feedback"></small>
                    </div>
                       <button type="submit" class="btn btn-
primary">@lang('Envoyer')</button>
                </form>
            </div>
        </div>
    </div>
</div>
```

Et on ajoute le code Javascript pour gérer ça :

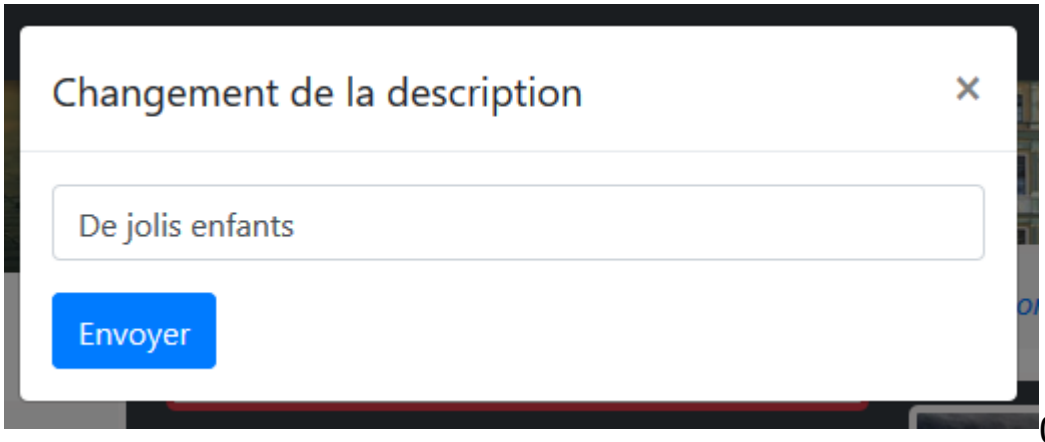
```
const swallAlertServer = () => {
    swal({
         title: '@lang('Il semble y avoir une erreur sur le
serveur, veuillez réessayer plus tard...')',
        type: 'warning'
    })
}

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

$('a.description-manage').click((e) => {
    e.preventDefault()
    let that = $(e.currentTarget)
    let text = that.parents('.card').find('.card-text').text()
    $('#description').val(text)
                    $('#descriptionForm').attr('action',
that.attr('href')).find('input').removeClass('is-
invalid').next().text()
    $('#changeDescription').modal('show')
})

$('#descriptionForm').submit((e) => {
    e.preventDefault()
    let that = $(e.currentTarget)
```

```
    $.ajax({
        method: 'put',
        url: that.attr('action'),
        data: that.serialize()
    })
        .done((data) => {
            let card = $('#image' + data.id)
            let body = card.find('.card-body')
            if(body.length) {
                body.children().text(data.description)
            } else {
                  card.children('a').after('<div class="card-
body"><p class="card-text">' + data.description + '</p></div>')
            }
            $('#changeDescription').modal('hide')
        })
        .fail((data) => {
            if(data.status === 422) {
                 $.each(data.responseJSON.errors, function (key,
value) {
                      $('#descriptionForm input[name=' + key +
']').addClass('is-invalid').next().text(value)
                })
            } else {
                swallAlertServer()
            }
        })
})
```

On va créer la route en prévoyant quelques groupes pour l’avenir :

```
Route::middleware ('auth', 'verified')->group (function () {


    ...
     Route::name ('image.')->middleware ('ajax')->group (function
() {
        Route::prefix('image')->group(function () {
                        Route::name  ('description')->put
('{image}/description', 'ImageController@descriptionUpdate');
        });
    });
});
```

Maintenant en cliquant sur l’icône on ouvre la feuille modale :



On va s'occuper maintenant du traitement côté serveur…

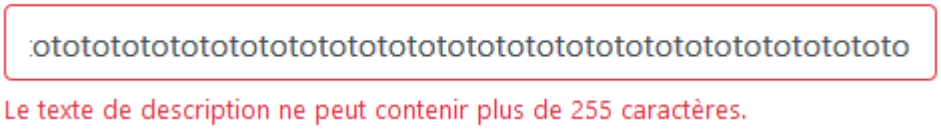
On ajoute la méthode dans **ImageController** :

```
public function descriptionUpdate(Request $request, Image $image)
{
    $this->authorize ('manage', $image);
    $request->validate ([
        'description' => 'nullable|string|max:255'
    ]);
    $image->description = $request->description;
    $image->save();
    return $image;
}
```

On en profite pour utilise l'autorisation de modification de l'image qu'on a déjà mis en place pour la suppression.

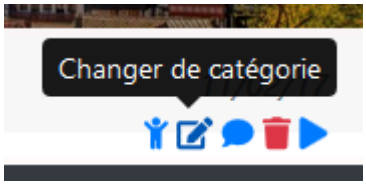
Et maintenant vous devriez pouvoir changer la description !

Vous pouvez aussi vérifier que la validation fonctionne :



Changer de catégorie

Pour changer de catégorie c'est comme pour la description et on va donc ajouter un code très proche. Ça se passe avec cette icône :



Dans la vue **home** on a déjà le bon code au niveau de l'icône :

```
<a class="category-edit"
    data-id="{{ $image->category_id }}"
    href="{{ route('image.update', $image->id) }}"
    data-toggle="tooltip"
    title="@lang('Changer de catégorie')">
    <i class="fa fa-edit"></i>
</a>
```

On ajoute le code pour la feuille modale :

```
<div  class="modal  fade"  id="changeCategory"  tabindex="-1"
role="dialog" aria-labelledby="categoryLabel" aria-hidden="true">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                                    <h5  class="modal-title"
id="categoryLabel">@lang('Changement de la catégorie')</h5>
                  <button type="button" class="close" data-
dismiss="modal" aria-label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body">
                <form id="editForm" action="" method="POST">
                    @method('PUT')
                    @csrf
                    <div class="form-group">
                              <select class="form-control"
name="category_id">
                            @foreach($categories as $cat)
                                <option value="{{ $cat->id }}">{{
$cat->name }}</option>
                            @endforeach
                        </select>
                    </div>
                       <button type="submit" class="btn btn-
primary">@lang('Envoyer')</button>
```

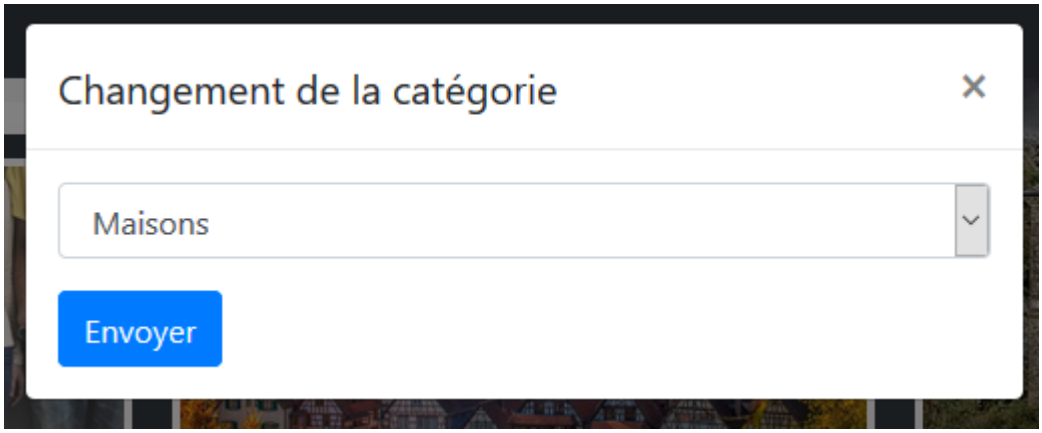
```
                </form>
            </div>
        </div>
    </div>
</div>
```

Pour la soumission cette fois on ne passera pas par le Javascript mais directement par le formulaire et on aura donc un rechargement de la page au retour du serveur. Ça permet de gérer le cas où on se trouve dans une catégorie et que l'image n'a plus à s'y trouver puisqu'on a changé sa catégorie !

Et on ajoute le code Javascript pour gérer l'ouverture de la feuille modale :

```
$('a.category-edit').click((e) => {
    e.preventDefault()
    let that = $(e.currentTarget)
    $('select').val(that.attr('data-id'))
    $('#editForm').attr('action', that.attr('href'))
    $('#changeCategory').modal('show')
})
```

Maintenant le clic sur l'icône ouvre la feuille modale avec une liste des catégories disponibles :



IL ne reste plus qu'à compléter le contrôleur **ImageController** :

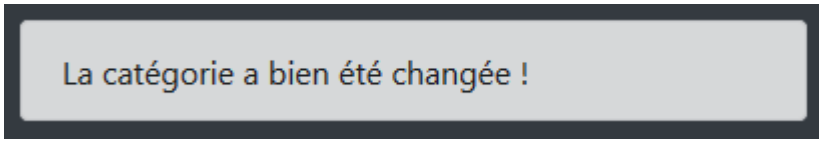
```
public function update(Request $request, Image $image)
{
    $this->authorize('manage', $image);

    $image->category_id = $request->category_id;
    $image->save();
```

```
    return back()->with('updated', __('La catégorie a bien été changée !'));
}
```

Et ça devrait fonctionner !

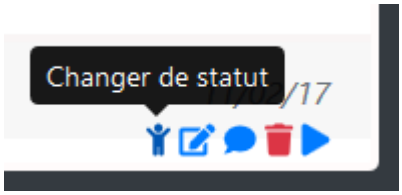
On va juste ajouter dans la vue **home** le code pour afficher l'information de changement de catégorie :

```
<main class="container-fluid">
    @if(session('updated'))
        <div class="alert alert-dark" role="alert">
            {{ session('updated') }}
        </div>
    @endif
```



Changer le statut

Voyons maintenant la dernière icône qui concerne le changement de statut (adulte ou pas adulte) :



Pour le moment on a le code de l'icône mais pas le **href** renseigné :

```
<a class="adult-edit"
    href="#"
    data-toggle="tooltip"
    title="@lang('Changer de statut')">
    <i class="fa @if($image->adult) fa-graduation-cap @else fa-child @endif"></i>
</a>
```

On va déclarer une route qu'on a pas encore créée :

```
href="{{ route('image.adult', $image->id) }}"
```

On ajoute cette route dans le fichier **routes/web** dans le groupe qu'on a déjà créé pour la description :

```
Route::prefix('image')->group(function () {
     Route::name ('description')->put ('{image}/description',
'ImageController@descriptionUpdate');
         Route::name  ('adult')->put  ('{image}/adult',
'ImageController@adultUpdate');
});
```

Dans la vue Home on ajoute aussi le Javascript qui va changer l'icône, envoyer la requête en Ajax et gérer la réponse du serveur :

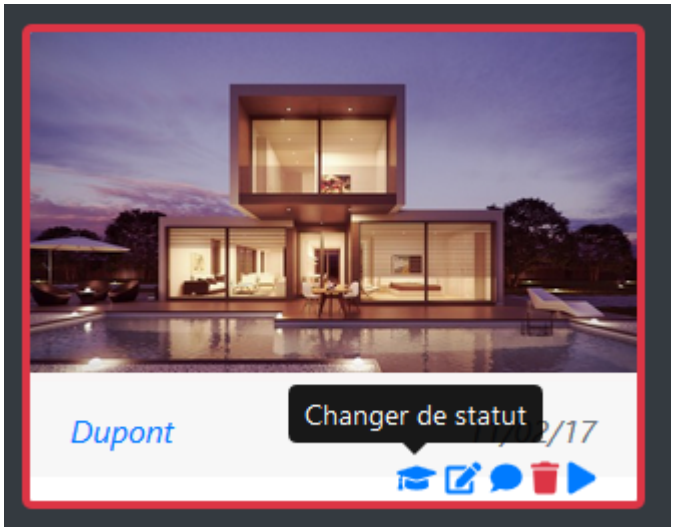
```
$('a.adult-edit').click((e) => {
    e.preventDefault()
    let that = $(e.currentTarget)
    let icon = that.children()
    let adult = icon.hasClass('fa-graduation-cap')
    if(adult) {
        icon.removeClass('fa-graduation-cap')
    } else {
        icon.removeClass('fa-child')
    }
    icon.addClass('fa-cog fa-spin')
    adult = !adult
    $.ajax({
        method: 'put',
        url: that.attr('href'),
        data: { adult: adult }
    })
        .done(() => {
            that.tooltip('hide')
            let icon = that.children()
            icon.removeClass('fa-cog fa-spin')
            let card = that.parents('.card')
            if(adult) {
                icon.addClass('fa-graduation-cap')
                card.addClass('border-danger')
            } else {
                icon.addClass('fa-child')
                card.removeClass('border-danger')
```

```
            }
        })
        .fail(() => {
            swallAlertServer()
        })
})
```

Enfin dans **ImageController** on crée la fonction qui va mettre à jour le champ **adult** :

```
public function adultUpdate(Request $request, Image $image)
{
    $this->authorize ('manage', $image);

    $image->adult = $request->adult == 'true';
    $image->save();

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

Maintenant on peut changer les statut avec un simple clic. Quand une image à le statut adulte elle a un liseré rouge :



Conclusion

Dans ce chapitre on a :

- créé la possibilité d'afficher les photos d'un utilisateur en cliquant sur son nom sur une de ses photos
- créé un menu d'icône animés pour les actions sur les photos
- donné la possibilité de supprimer une photo en limitant l'accès à l'owner et l'administrateur
- donné la possibilité de modifier la description d'une photo en limitant l'accès à l'owner et l'administrateur
- donné la possibilité de modifier la catégorie d'une photo en limitant l'accès à l'owner et l'administrateur
- donné la possibilité de modifier le statut adulte d'une photo en limitant l'accès à l'owner et l'administrateur

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