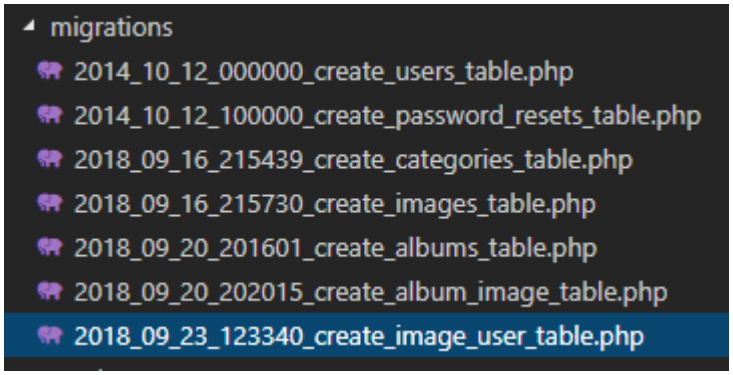
Laravel 5.7 par la pratique – Notation des photos

Dans ce chapitre on va permettre aux utilisateurs authentifier de noter les photos des autres utilisateurs. On va adopter une approche visuelle classique avec une série de 5 étoiles.

La base

Pour mémoriser les notes on va créer une nouvelle table. Un utilisateur peut noter plusieurs photos et une photo peut être notée par plusieurs utilisateurs, on a donc une relation de type **ManyToMany** avec une table pivot. Créons cette table :

php artisan make:migration create_image_user_table

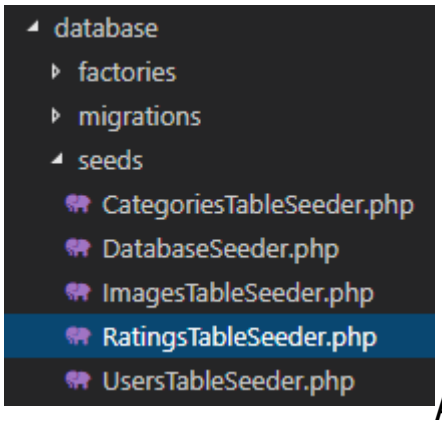


On complète le code :

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;

class CreateImageUserTable extends Migration
{
    public function up()
    {
        Schema::create('image_user', function (Blueprint $table) {
            $table->increments('id');
            $table->timestamps();
            $table->integer('rating');
```

```
            $table->unsignedInteger('user_id')->index();
            $table->unsignedInteger('image_id')->index();
$table->foreign('user_id')->references('id')->on('users')->onDelet
e('cascade');
$table->foreign('image_id')->references('id')->on('images')->onDel
ete('cascade');
        });
    }

    public function down()
    {
        Schema::drop('image_user');
    }
}
```

Histoire d’avoir déjà des notes on crée un seeder :



Avec ce code :

```
<?php

use Illuminate\Database\Seeder;

class RatingsTableSeeder extends Seeder
{
    public function run()
    {
        \DB::table('image_user')->insert([
            0 => [
                'image_id' => 39,
                'user_id' => 3,
                'rating' => 1,
            ],
            1 => [
                'image_id' => 40,
                'user_id' => 3,
```

```
        'rating' => 2,
    ],
    2 => [
        'image_id' => 37,
        'user_id' => 3,
        'rating' => 2,
    ],
    3 => [
        'image_id' => 43,
        'user_id' => 3,
        'rating' => 2,
    ],
    4 => [
        'image_id' => 39,
        'user_id' => 2,
        'rating' => 5,
    ],
    5 => [
        'image_id' => 37,
        'user_id' => 2,
        'rating' => 5,
    ],
    6 => [
        'image_id' => 41,
        'user_id' => 2,
        'rating' => 3,
    ],
    7 => [
        'image_id' => 36,
        'user_id' => 2,
        'rating' => 2,
    ],
    7 => [
        'image_id' => 31,
        'user_id' => 3,
        'rating' => 3,
    ],
    8 => [
        'image_id' => 32,
        'user_id' => 3,
        'rating' => 3,
    ]
]);
```

```
    }
}
```

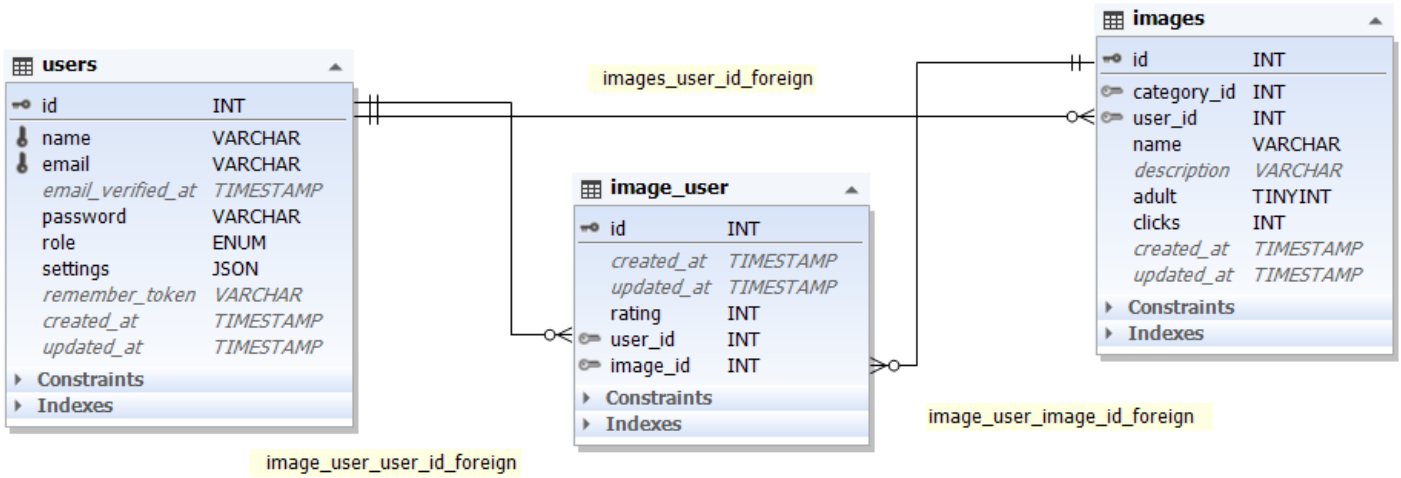
On complète le code de **DatabaseSeeder** :

```
public function run()
{
    ...

    $this->call(RatingsTableSeeder::class);
}
```

On n'a plus qu'à rafraîchir la base :

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

*Vous aurez peut-être besoin de lancer un **composer dumpautoload** pour charger la classe.*



Les relations

Dans le modèle **User** on ajoute la relation :

```
public function imagesRated()
{
    return $this->belongsToMany (Image::class);
}
```

Ainsi que dans le modèle **Image** :

```
public function users()
{
```

```
                        return      $this->belongsToMany
(User::class)->withPivot('rating');
}
```

Route et contrôleur

On va ajouter une route pour gérer la notation :

```
Route::name ('image.')->middleware ('ajax')->group (function () {
    Route::prefix('image')->group(function () {

        ...

    });
        Route::name  ('rating')->put  ('rating/{image}',
'ImageController@rate');
});
```

On va crée une méthode dans **ImageController** :

```
public function rate(Request $request, Image $image)
{
    $user = $request->user();

    // Is user image owner ?
    if($this->imageRepository->isOwner ($user, $image)) {
        return response()->json(['status' => 'no']);
    }

    // Rating
     $rate = $this->imageRepository->rateImage ($user, $image,
$request->value);
    $this->imageRepository->setImageRate ($image);

    return [
        'status' => 'ok',
        'id' => $image->id,
        'value' => $image->rate,
        'count' => $image->users->count(),
        'rate' => $rate
    ];
}
```

On vérifie que celui qui note n’est pas le propriétaire de la photo, parce qu’il n’a pas le droit de noter ses propres photos.

Comme d’habitude on délègue au repository (**ImageRepository**) la gestion des données :

```
public function rateImage($user, $image, $value)
{
          $rate   =   $image->users()->where('users.id', $user->id)->pluck('rating')->first();

    if($rate) {
        if($rate !== $value) {
             $image->users ()->updateExistingPivot ($user->id, ['rating' => $value]);
        }
    } else {
          $image->users ()->attach ($user->id, ['rating' => $value]);
    }

    return $rate;
}

public function isOwner($user, $image)
{
    return $image->user()->where('users.id', $user->id)->exists();
}
```

On vérifie si l’utilisateur en question a déjà noté la photo, dans ce cas il faut mettre à jour sa notation, sinon il faut la créer.

On a ainsi la gestion d’une note qui arrive mais il faut aussi envoyer la note de chaque image qu’on affiche pour savoir combien d’étoiles prévoir !

Ajouter la note aux photos

Donc chaque fois qu’on envoie les informations d’une photos à partir du serveur il faut maintenant ajouter la note. Ça va se passer dans **ImageRepository**. On commence par créer ces 3 fonctions :

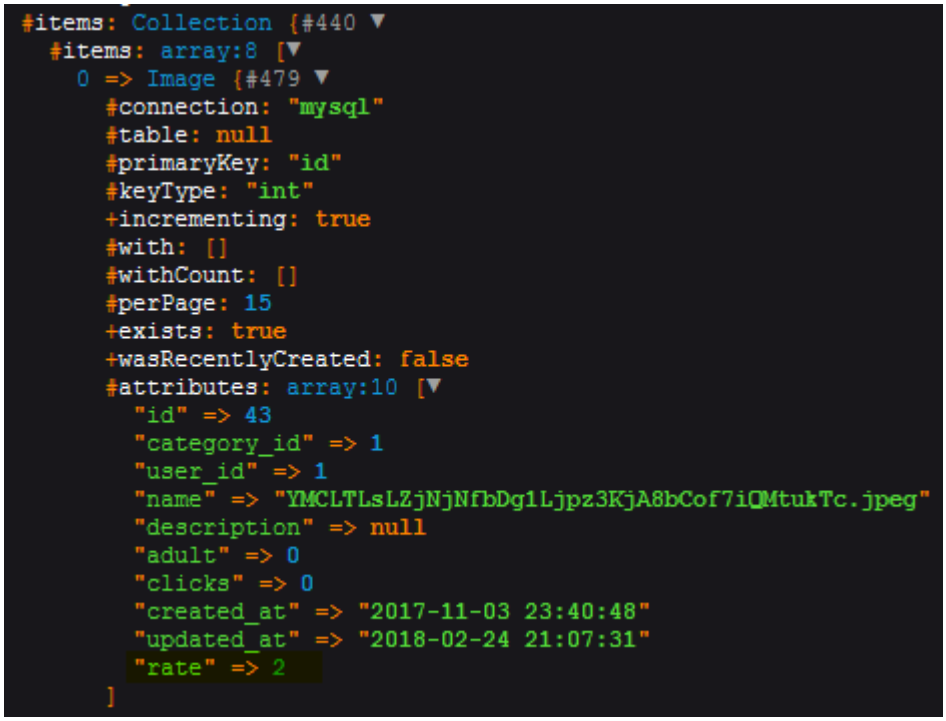
```
public function paginateAndRate($query)
{
    $images = $query->paginate (config ('app.pagination'));
    return $this->setRating ($images);
}

public function setRating($images)
{
    $images->transform(function ($image) {
        $this->setImageRate ($image);
        return $image;
    });
    return $images;
}

public function setImageRate($image)
{
    $number = $image->users->count();
       $image->rate = $number ? $image->users->pluck
('pivot.rating')->sum () / $number : 0;
}
```

Et partout où on se contentait de paginer on va ajouter la note aux images :

```
public function getAllImages()
{
    return $this->paginateAndRate (Image::latestWithUser());
}

public function getImagesForCategory($slug)
{
     $query = Image::latestWithUser ()->whereHas ('category',
function ($query) use ($slug) {
        $query->whereSlug ($slug);
    });
    return $this->paginateAndRate ($query);
}

public function getImagesForUser($id)
{
    $query = Image::latestWithUser ()->whereHas ('user', function
($query) use ($id) {
        $query->whereId ($id);
```

```
    });
    return $this->paginateAndRate ($query);
}

public function getImagesForAlbum($slug)
{
     $query = Image::latestWithUser ()->whereHas ('albums',
function ($query) use ($slug) {
        $query->whereSlug ($slug);
    });
    return $this->paginateAndRate ($query);
}
```

Vous n'allez encore rien remarquer à l'affichage de la galerie mais les données sont bien là :



La vue

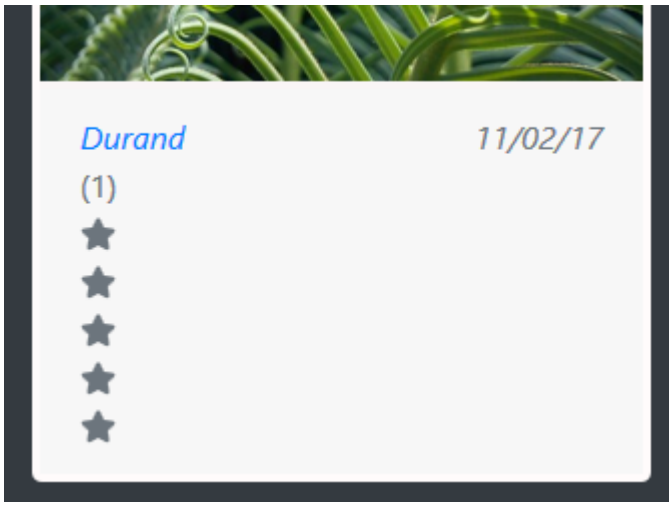
Il ne nous reste plus qu'à gérer le côté client…

On commence par placer les étoiles dans la vue **home** :

```
<div class="star-rating" id="{{ $image->id }}">
      <span class="count-number">({{ $image->users->count()
}})</span>
     <div id="{{ $image->id . '.5' }}" data-toggle="tooltip"
```

```
title="5" @if($image->rate > 4) class="star-yellow" @endif>
        <i class="fas fa-star"></i>
    </div>
     <div id="{{ $image->id . '.4' }}" data-toggle="tooltip"
title="4" @if($image->rate > 3) class="star-yellow" @endif>
        <i class="fas fa-star"></i>
    </div>
     <div id="{{ $image->id . '.3' }}" data-toggle="tooltip"
title="3" @if($image->rate > 2) class="star-yellow" @endif>
        <i class="fas fa-star"></i>
    </div>
     <div id="{{ $image->id . '.2' }}" data-toggle="tooltip"
title="2" @if($image->rate > 1) class="star-yellow" @endif>
        <i class="fas fa-star"></i>
    </div>
     <div id="{{ $image->id . '.1' }}" data-toggle="tooltip"
title="1" @if($image->rate > 0) class="star-yellow" @endif>
        <i class="fas fa-star"></i>
    </div>
    <span class="pull-right">
        @adminOrOwner($image->user_id)
```

Bon, ce n’est pas encore très fun :



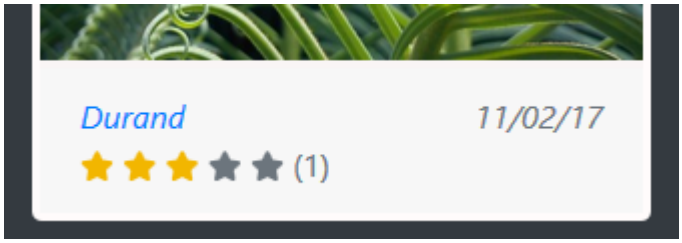
On va ajouter quelques règles CSS dans **resources/sass/app.scss** :

```
.star-rating div {
  display: inline-block;
  font-size: 15px;
  -webkit-transition: all .3s ease-in-out;
  transition: all .3s ease-in-out;
  cursor: pointer;
```

```
}
.star-yellow,
.star-rating div:hover,
.star-rating div:hover ~ div {
  color: #f2b600
}
.hover_img a { position:relative; }
.hover_img a span { position:absolute; display:none; z-index:99; }
.hover_img a:hover span { display:block; }
```

On relance la compilation avec **npm run dev**.

Ça s'est un peu arrangé :



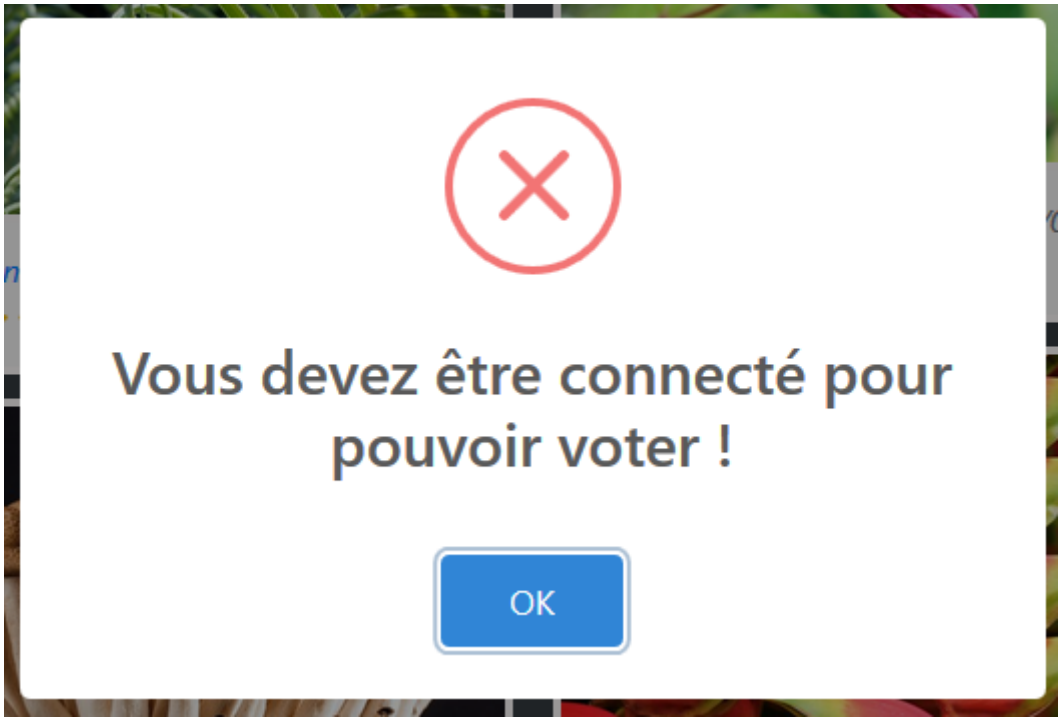
On ajoute le Javascript pour gérer tout ça dans **home** :

```
let memoStars = []

$('.star-rating div').click((e) => {
    @auth
        let element = $(e.currentTarget)
        let values = element.attr('id').split('.')
        element.addClass('fa-spin')
        $.ajax({
            url: "{{ url('rating') }}" + '/' + values[0],
            type: 'PUT',
            data: {value: values[1]}
        })
        .done((data) => {
            if (data.status === 'ok') {
                let image = $('#' + data.id)
                memoStars = []
                image.children('div')
                    .removeClass('star-yellow')
                    .each(function (index, element) {
                        if (data.value > 4 - index) {
                            $(element).addClass('star-yellow')
                            memoStars.push(true)
```

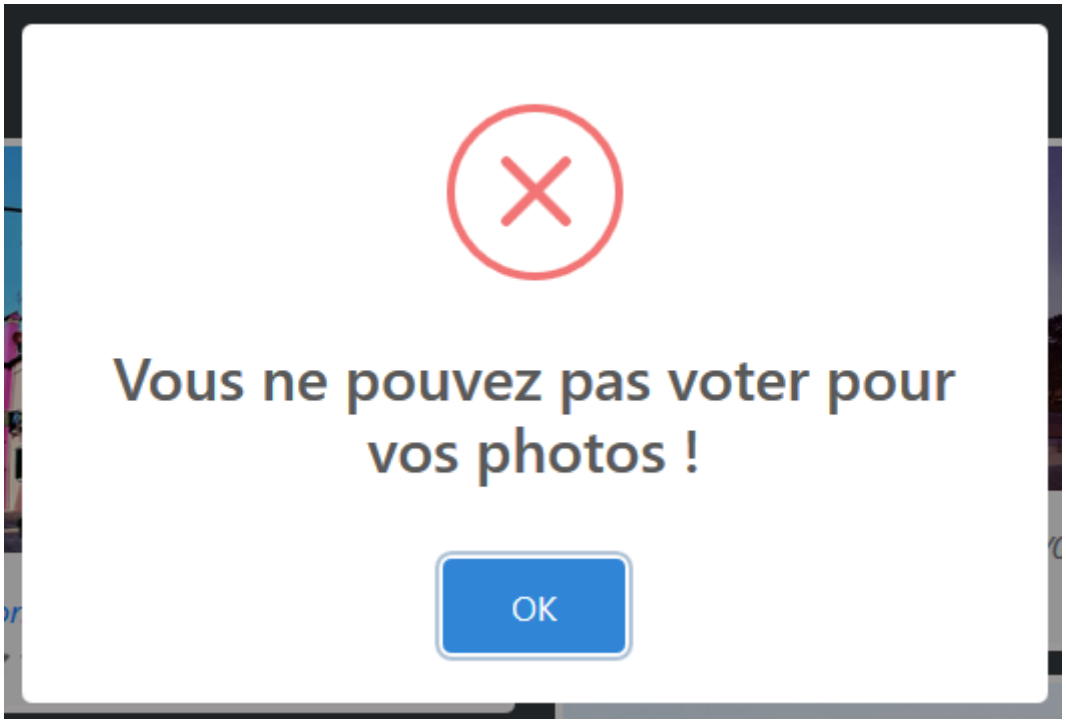
```
                        }
                        memoStars.push(false)
                    })
                    .end()
                    .find('span.count-number')
                    .text('(' + data.count + ')')
                if(data.rate) {
                    if(data.rate == values[1]) {
                        title = '@lang("Vous avez déjà donné cette
note !")'
                    } else {
                         title = '@lang("Votre vote a été modifié
!")'
                    }
                } else {
                    title = '@lang("Merci pour votre vote !")'
                }
                swal({
                    title: title,
                    type: 'warning'
                })
            } else {
                swal({
                     title: '@lang('Vous ne pouvez pas voter pour
vos photos !')',
                    type: 'error'
                })
            }
            element.removeClass('fa-spin')
        })
        .fail(() => {
            swallAlertServer()
            element.removeClass('fa-spin')
        })
    @else
        swal({
             title: '@lang('Vous devez être connecté pour pouvoir
voter !')',
            type: 'error'
        })
    @endauth
})
```

```
$('.star-rating').hover(
    (e) => {
        memoStars = []
        $(e.currentTarget).children('div')
            .each((index, element) => {
                memoStars.push($(element).hasClass('star-yellow'))
            })
            .removeClass('star-yellow')
    }, (e) => {
    $.each(memoStars, (index, value) => {
        if(value) {
              $(e.currentTarget).children('div:eq(' + index +
')').addClass('star-yellow')
        }
    })
})
```

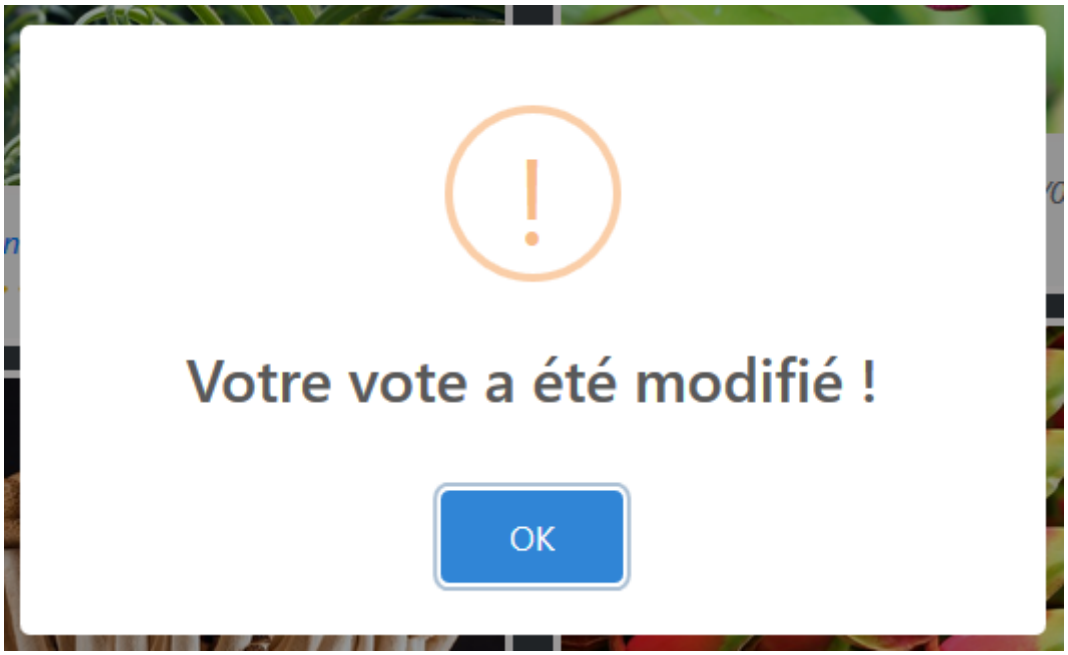
Maintenant au survol on a les étoiles qui réagissent correctement.
Su un utilisateur non connecté clique il a ce message :



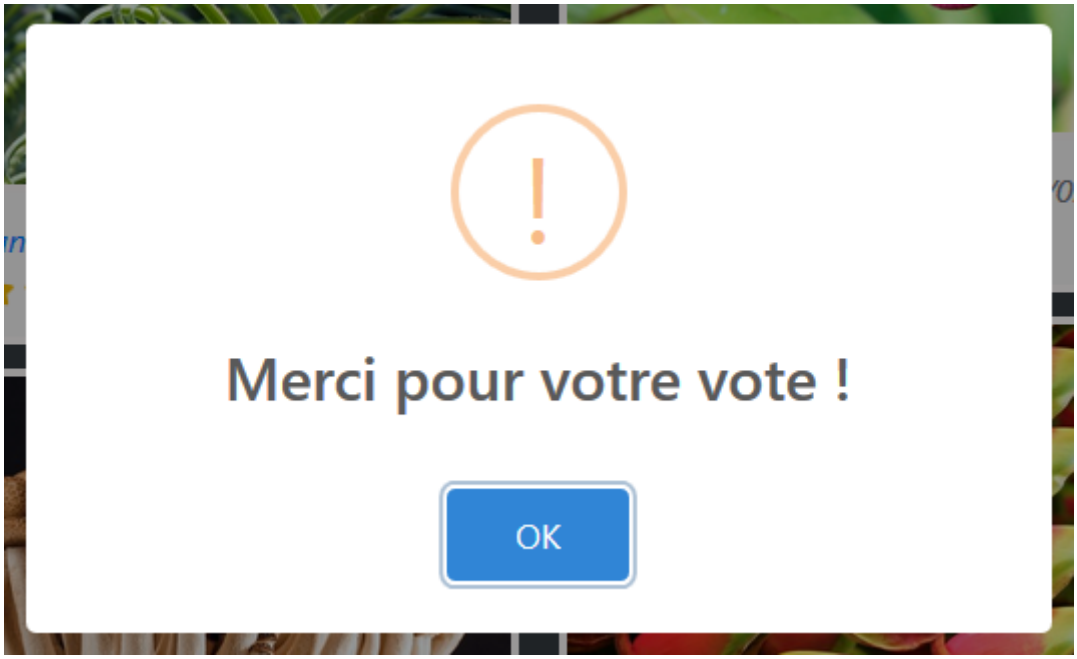
Si on essaie de voter pour ses photos :



Si on change son vote :



Et si on ajoute un vote :



Le chiffre entre parenthèses indique le nombre de votes et évidemment on fait la moyenne des votes pour afficher la bonne étoile.

Le nombre de vues

On va terminer ce chapitre en ajoutant le nombre de fois qu'une photo a été cliquée. On n'a pas beosin de toucher à la base parce qu'on a déjà prévu un champ **clicks** dans la table **images**.

On ajoute cette route :

```
Route::middleware('ajax')->name('image.click')->patch('image/{image}/click', 'ImageController@click');
```

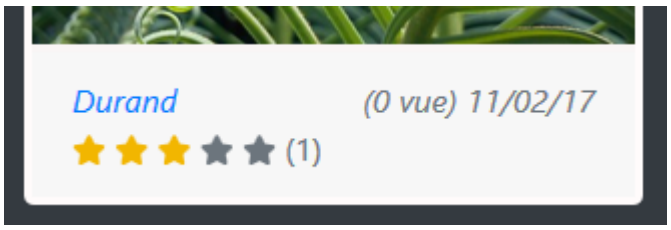
On ajoute cette fonction dans **ImageController** :

```
public function click(Request $request, Image $image)
{
      if ($request->session()->has('images') && in_array($image->id, session ('images'))) {
        return response ()->json (['increment' => false]);
    }
    $request->session()->push('images', $image->id);
    $image->increment('clicks');
    return ['increment' => true];
}
```

On mémorise le clic en session pour ne pas comptabiliser plusieurs fois le clic d'un même utilisateur.

Dans la vue **home** on ajoute la valeur à côté de la date :

```
<div class="pull-right">
    <em>
        (<span class="image-click">{{ $image->clicks }}</span> {{ trans_choice(__('vue|vues'),  $image->clicks)  }})  {{ $image->created_at->formatLocalized('%x') }}
    </em>
</div>
```



On gère le pluriel avec **trans_choice**.

On ajoute le Javascript pour gérer ça :

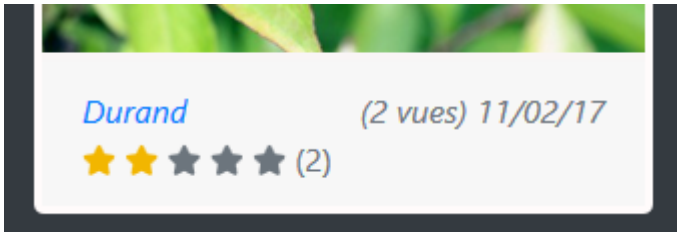
```
$('a.image-link').click((e) => {
    e.preventDefault()
    let that = $(e.currentTarget)
    $.ajax({
        method: 'patch',
        url: that.attr('data-link')
    }).done((data) => {
        if(data.increment) {
                let numberElement = that.siblings('div.card-footer').find('.image-click')
            numberElement.text(parseInt(numberElement.text()) + 1)
        }
    })
})
```

Pour récupérer la bonne url on ajoute une référence pour chaque image (**data-link**) :

```
@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" data-link="{{ route('image.click', $image->id)
}}">
```

Et maintenant ça devrait fonctionner :



En résumé

Dans ce chapitre on a :

- ajouté la notation des images
- ajouté le nombre de vues

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