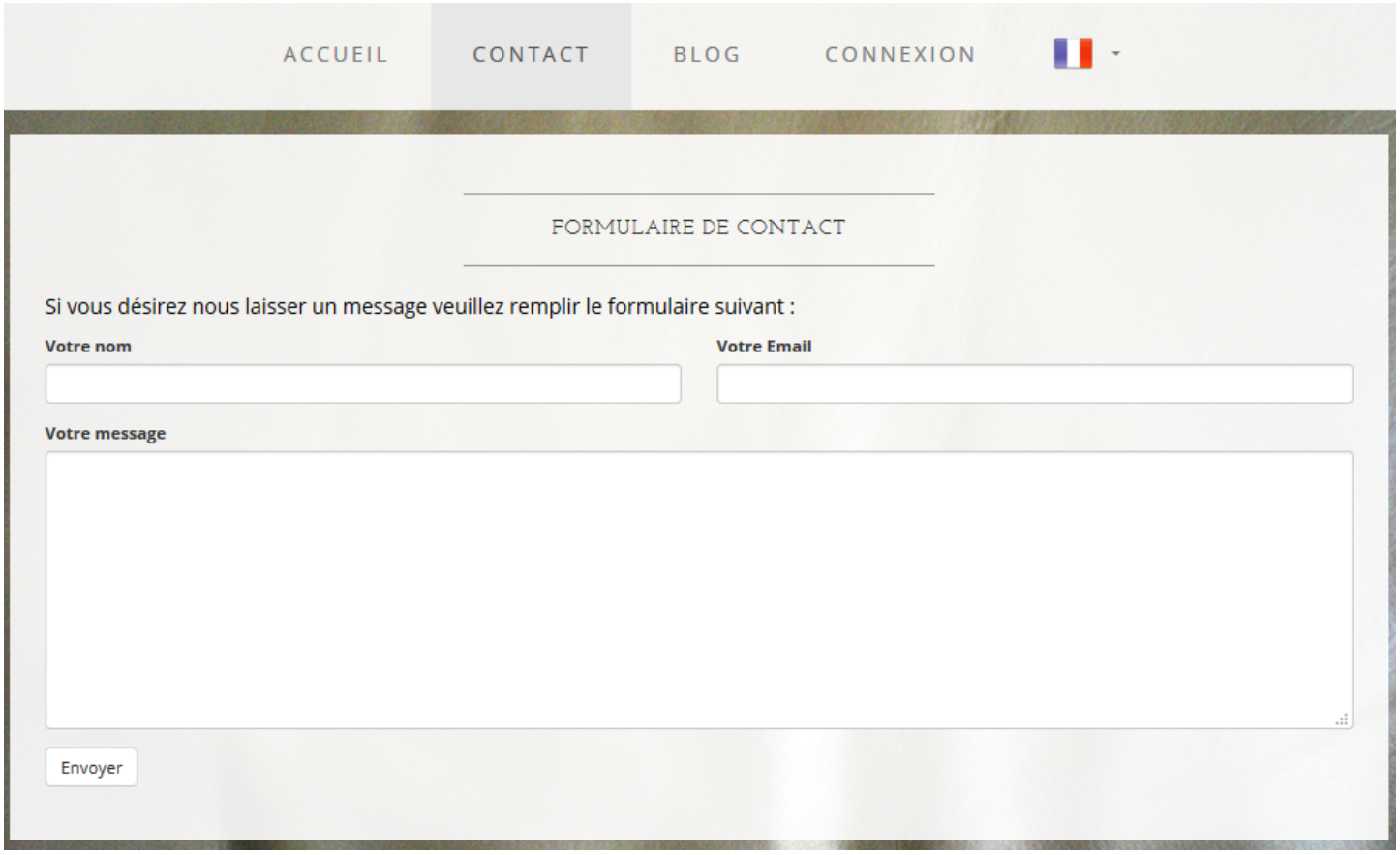
Cours Laravel 5.3 – plus loin – ajax

Ajax est une technologie Javascript fort répandue qui permet d'envoyer des requêtes au serveur et de recevoir des réponses sans rechargement de la page. Il est par ce moyen possible de modifier dynamiquement le DOM, donc une partie de la page.

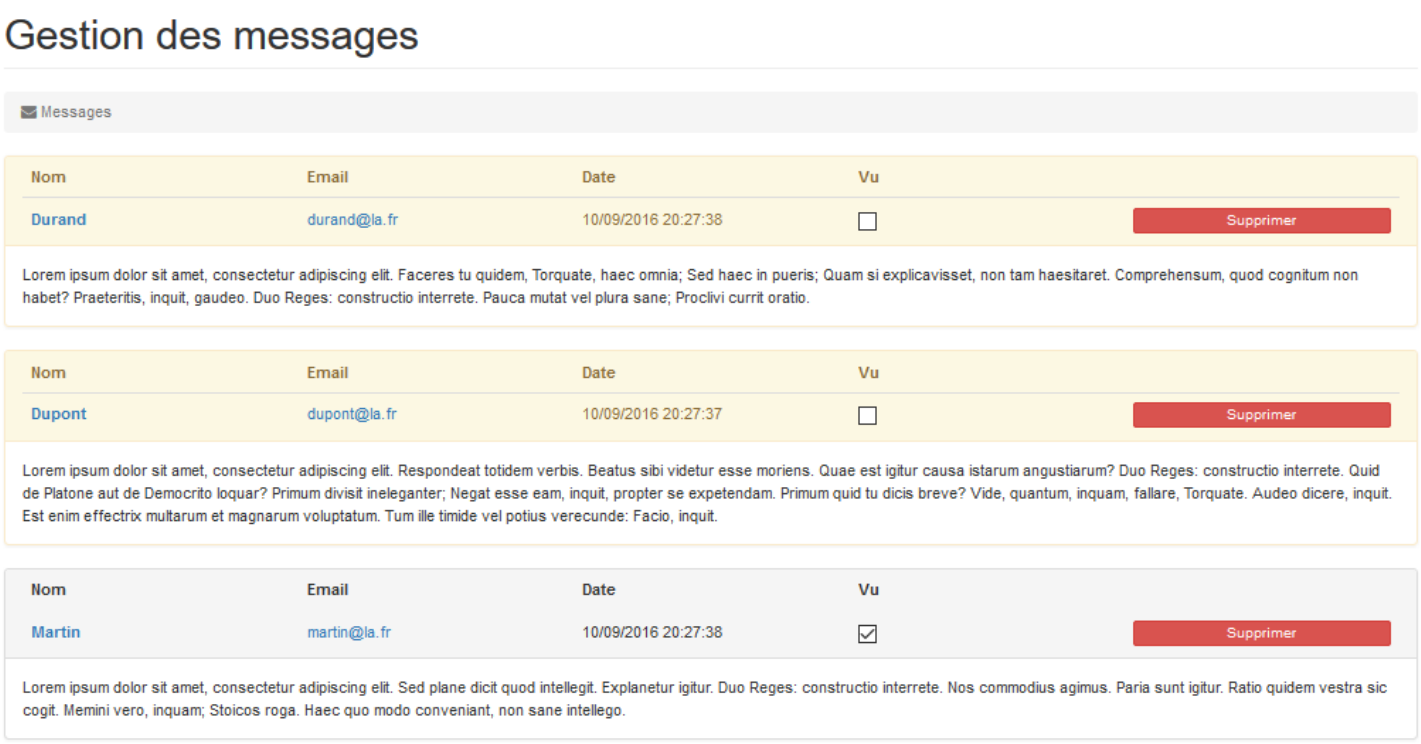
Dans ce chapitre nous allons voir comment mettre en œuvre Ajax avec Laravel en prenant un cas de l'application d'exemple.

Les messages dans l'application

Pour les utilisateurs autres que rédacteurs et administrateurs il y a la possibilité de laisser un message avec un formulaire :



Lorsqu'un administrateur se connecte et va dans le panneau d'administration il a une page pour ces messages :



Pour se connecter en tant qu'administrateur il suffit d'utiliser le login admin@la.fr avec le mot de passe admin (si vous avez effectué la population prévue dans l'application).

Les messages qui n'ont pas encore été vus apparaissent avec un fond jauni, par défaut on en a deux. D'autre part l'administrateur dispose pour chaque message d'une case à cocher pour changer ce statut.

Il est évident qu'il serait dommage de recharger la page pour ça alors qu'il suffit d'informer le serveur avec une requête Ajax et localement de changer la couleur du fond.

Pour améliorer l'ergonomie on a aussi prévu une petite animation à la place de la case à cocher le temps de l'échange de requêtes :

Il faut aussi gérer cette animation avec JavaScript.

La vue des messages

Dans la vue des messages (**views/back/messages/index.blade.php**) on a cette partie du code chargée de l'affichage du tableau des messages :

```
@foreach ($messages as $message)
     <div class="panel {!! $message->seen? 'panel-default' : 'panel-warning' !!}">
        <div class="panel-heading">
            <table class="table">
                <thead>
                    <tr>
                                        <th class="col-lg-1">{{ trans('back/messages.name') }}</th>
                                        <th class="col-lg-1">{{ trans('back/messages.email') }}</th>
                                        <th class="col-lg-1">{{ trans('back/messages.date') }}</th>
                                        <th class="col-lg-1">{{ trans('back/messages.seen') }}</th>
                        <th class="col-lg-1"></th>
                    </tr>
                </thead>
                <tbody>
                    <tr>
                           <td class="text-primary"><strong>{{ $message->name }}</strong></td>
                          <td>{!! HTML::mailto($message->email, $message->email) !!}</td>
                        <td>{{ $message->created_at }}</td>
                            <td>{!! Form::checkbox('seen', $message->id, $message->seen) !!}</td>
                        <td>
                            {!! Form::open(['method' => 'DELETE', 'route' => ['contact.destroy', $message->id]]) !!}
                                                                {!! Form::destroyBootstrap(trans('back/messages.destroy'), trans('back/messages.destroy-warning'), 'btn-xs') !!}
                             {!! Form::close() !!}
                        </td>
                    </tr>
```

```
                </tbody>
            </table>
        </div>
        <div class="panel-body">
            {{ $message->message }}
        </div>
    </div>
@endforeach
```

On a une boucle pour passer en revue tous les messages :

```
@foreach ($messages as $message)
    ...
@endforeach
```

On gère la couleur de fond avec deux classes de Bootstrap :

```
<div class="panel {!! $message->seen? 'panel-default' : 'panel-warning' !!}">
```

Les cases à cocher sont générées classiquement (laravelcollective/html) :

```
<td>{!! Form::checkbox('seen', $message->id, $message->seen) !!}</td>
```

Ce qui donne ce code pour la case cochée :

```
<td><input checked="checked" name="seen" type="checkbox" value="3"></td>
```

On distingue les cases avec l'attribut value dans lequel on mémorise l'identifiant du message.

Le JavaScript

On a aussi sur la page le JavaScript pour gérer Ajax :

```
$(function () {
    $(':checkbox').change(function () {
            $(this).parents('.panel').toggleClass('panel-warning').toggleClass('panel-default');
        $(this).hide().parent().append('<i class="fa fa-refresh fa-spin"></i>');
```

```
        $.ajax({
            url: 'contact/' + this.value,
            type: 'PUT',
            data: 'seen=' + this.checked
        })
        .done(function () {
            $('.fa-spin').remove();
            $('input[type="checkbox"]:hidden').show();
        })
        .fail(function () {
            $('.fa-spin').remove();
            var chk = $('input[type="checkbox"]:hidden');
                 chk.parents('.panel').toggleClass('panel-
warning').toggleClass('panel-default');
            chk.show().prop('checked', chk.is(':checked') ? null :
'checked');
            alert('{{ trans('back/messages.fail') }}');
        });
    });
});
```

On utilise jQuery qui est déjà chargé pour toutes les pages.

Même si ce n'est pas l'objet de ce cours on va un peu analyser ce code…

On détecte les changements sur toutes les cases à cocher (on n'a pas d'autres cases à cocher que celles des messages sur la page) :

```
$(':checkbox').change(function () {
```

On change la couleur de fond :

```
$(this).parents('.panel').toggleClass('panel-
warning').toggleClass('panel-default');
```

On cache la case à cocher et on fait apparaître l'icône animée :

```
$(this).hide().parent().append('<i  class="fa  fa-refresh  fa-
spin"></i>');
```

On envoie la requête Ajax :

```
$.ajax({
    url: 'contact/' + this.value,
```

```
    type: 'PUT',
    data: 'seen=' + this.checked
})
```

On définit ainsi :

- l'url : de la forme **contact/id**
- la méthode : **PUT**,
- l'information : **seen**: **true** ou **false**. (une autre syntaxe possible est **data: {seen:this.checked}**)

Si tout fonctionne bien on exécute ce code :

```
.done(function () {
    $('.fa-spin').remove();
    $('input[type="checkbox"]:hidden').show();
})
```

On supprime la petite animation et on affiche à nouveau la case à cocher.

S'il y a eu un problème on exécute ce code :

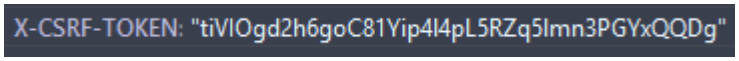
```
.fail(function () {
    $('.fa-spin').remove();
    var chk = $('input[type="checkbox"]:hidden');
                chk.parents('.panel').toggleClass('panel-
warning').toggleClass('panel-default');
     chk.show().prop('checked', chk.is(':checked') ? null :
'checked');
    alert('{{ trans('back/messages.fail') }}');
});
```

- on supprime aussi la petite animation,
- on change à nouveau la couleur de fond puisqu'on a échoué,
- on fait réapparaître la case à cocher en changeant son état,
- on affiche une alerte pour prévenir du problème.

La protection CSRF

Je vous ai dit plusieurs fois dans ce cours que Laravel met en place systématiquement la protection CSRF. Or ici à aucun moment on ne transmet le jeton (**token**) pour cela !

Si vous regardez dans les headers lors de l'envoi de la requête vous allez trouver ceci :



Le middleware de Laravel qui assure la protection (**VerifyCsrfToken**) ne se contente pas de chercher le jeton dans les paramètres de la requête, il va aussi voir dans les headers s'il y a une information **X-CSRF-TOKEN**. Ce qui est notre cas.

Mais comment s'est créée cette information ?

Si vous regardez le template (**resources/views/back/template.blade.php**) vous aller trouver ce code :

```
...
    <meta name="csrf-token" content="{{ csrf_token() }}">
...
    <script>

        $(function() {

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

Le jeton (**token**) est mémorisé dans les metas et avec la méthode **ajaxSetup** on demande à jQuery d'ajouter automatiquement l'information dans les headers. On n'a donc plus à s'en préoccuper ensuite…

Le traitement côté serveur

Le contrôleur

Voici la partie concernée du contrôleur de ressource :

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Requests\ContactRequest;
use App\Repositories\ContactRepository;

class ContactController extends Controller
{
    /**
     * The ContactRepository instance.
     *
     * @var \App\Repositories\ContactRepository
     */
    protected $contactRepository;

    /**
     * Create a new ContactController instance.
     *
         *  @param   \App\Repositories\ContactRepository
$contactRepository
     * @return void
     */
        public  function  __construct(ContactRepository
$contactRepository)
    {
        $this->contactRepository = $contactRepository;

        $this->middleware('admin')->except('create', 'store');
        $this->middleware('ajax')->only('update');
    }

    ...

    /**
```

```
     * Update the specified contact in storage.
     *
     * @param  \Illuminate\Http\Request $request
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, $id)
    {
        $this->contactRepository->update($request->input('seen'),
$id);

        return response()->json();
    }


    ...
}
```

La requête arrive dans la méthode **update**. On fait appel au repository pour modifier la base :

```
public function update($seen, $id)
{
    $contact = $this->getById($id);

    $contact->seen = $seen == 'true';

    $contact->save();
}
```

Et on renvoie une réponse JSON.

Le middleware

*On peut remarquer que la méthode concernée (**update**) est protégée par un middleware ajax.*

En effet on veut que notre méthode ne soit accessible que si on reçoit une requête Ajax. Ce middleware n'existe pas par défaut dans Laravel, il faut donc le créer (**app/Http/Middlewares/Ajax.php**) :

```
<?php
```

```
namespace App\Http\Middleware;

use Closure;

class IsAjax
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        if ($request->ajax()) {
            return $next($request);
        }

        abort(404);
    }
}
```

La méthode **ajax** de la requête nous permet facilement de savoir s'il s'agit d'une requête de ce type, si ce n'est pas le cas on revoie une erreur 404.

Il faut informer Laravel que notre middleware existe (**app/Http/Kernel.php**) :

```
protected $routeMiddleware = [
    ...
    'ajax' => \App\Http\Middleware\IsAjax::class
];
```

En résumé

- Ajax est facile à mettre en œuvre avec Laravel.
- On peut faire en sorte que la protection CSRF soit automatiquement mise en oeuvre pour toutes les requêtes.
- On peut prévoir un middleware particulier pour les requêtes Ajax.