AJaX et Laravel

Le but de cette fiche est de présenter un exemple d'utilisation d'AJaX avec *Laravel*, ainsi que l'utilisation du plugin *JQuery DataTables*. Les exemples se basent sur le projet *Laravel* « exemple-bd » mis à disposition sur *GitLab*.

1. Exemple d'utilisation AJaX

Dans un premier temps, nous clonons le projet (remplacez :

```
git clone https://gitlab-mmi.univ-reims.fr/rabat01/laravel-exemple-bd exemple
```

Pour qu'il soit fonctionnel, il faut :

- Créer une base de données nommée `actualites` (par exemple)
- Modifier le fichier `.env` en spécifiant le nom de la base de données (par défaut `actualites`) et les identifiants de connexion (par défaut `root/root`)

Il faut ensuite lancer les migrations :

```
php artisan migrate
```

Et le remplissage de la base :

```
php artisan db:seed
```

Nous proposons de modifier la page d'accueil du projet pour que des actualités aléatoires s'affichent périodiquement.

Dans un premier temps, nous devons créer une méthode dans notre contrôleur pour récupérer le JSON d'une actualité aléatoire. Nous ajoutons la méthode suivante (dans la classe `ActualitesController.php` située dans le répertoire `app/http/controllers`) :

```
    public function actualite() {
        return Actualite::all()->random();
    }
```

Pour qu'elle soit accessible, nous ajoutons la route suivante dans le script `web.php` (dans le répertoire `routes/`) avant la déclaration du contrôleur :

```
Route::get('actualites/actualite', [ActualitesController::class, 'actualite']);
```

Nous pouvons vérifier son fonctionnement en saisissant l'URL http://localhost/nomprojet/public/actualites/actualite (en remplaçant « *nomprojet* » par le nom de votre projet). Une actualité apparaît au format JSON.

Il nous reste à modifier la page d'accueil. Dans la vue `index.blade.php` (dans le répertoire `resources/views`) nous ajoutons tout d'abord les éléments HTML qui accueilleront le contenu de l'actualité :

```
…
@section('contenu')
      <div id="actualite" class="mb-3">
        <h4 id="titre"></h4>
        <div id="texte"></div>
        <i id="date"></i>
      </div>
      …
```

Puis nous ajoutons l'inclusion de *JQuery* et la partie script *Javascript* suivante (après les liens HTML) :

```
      <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
      <script>
function recherche() {
    requete = $.ajax({
        "type" : "GET",
        "url" : "{{ url('/actualites/actualite') }}",
```

```
        "dataType": "json"
        });
    requete.fail(function (jqXHR, textStatus, errorThrown){
            console.log(jqXHR);
        });
    requete.done(function (response, textStatus, jqXHR) {
            $('#titre').text(response['titre']);
            $('#texte').text(response['message']);
            $('#date').text("(" + response['date'] + ")");
            setTimeout(recherche, 3000);
        });
}
$(document).ready(function (){
        recherche();
    });
      </script>
```

Retournez sur la page principale : http://localhost/nomprojet/public/. Vous devriez voir apparaître une actualité toutes les 3 secondes.

Vous pouvez récupérer ce projet en tapant la commande suivante :

```
git clone --branch ajax https://gitlab-mmi.univ-reims.fr/rabat01/laravel-exemple-bd
exemple
```

2. Utilisation de *DataTables* sans AJaX

DataTables est un plugin pour la bibliothèque *Javascript JQuery*. Il permet de créer des tableaux dynamiques. Il est possible pour l'utilisateur de trier les données, de réaliser des recherches dans les données du tableau, de naviguer sur les différentes pages, *etc.* Les données du tableau peuvent être récupérées via AJaX, ce que nous aborderons dans la section suivante. Vous pouvez consulter la documentation de ce plugin sur le site officiel : https://datatables.net/.

Nous proposons d'illustrer un fonctionnement basique pour afficher la liste des actualités. Nous allons créer une nouvelle vue que nous appelons `listdatatables.blade.php` (dans le répertoire `ressources/views/actualites`). Nous créons un tableau HTML « classique » :

```
…
@section('contenu')
<table class="table table-striped" id='liste'>
  <thead>
    <tr>
      <th scope="col">Date</th>
      <th scope="col">Titre</th>
    </tr>
  </thead>
  <tbody>
<?php
foreach($actualites as $actualite) {
?>
    <tr>
      <td> {{strftime('%d/%m/%Y', strtotime($actualite->date))}} </td>
      <td> {{$actualite->titre}} </td>
    </tr>
<?php
}
?>
  </tbody>
  <tfoot>
    <tr>
      <th scope="col">Date</th>
      <th scope="col">Titre</th>
    </tr>
```

```
  </tfoot>

</table>
…
```

Pour le moment, un tableau classique est affiché (même s'il possède un style défini dans *Bootstrap*). Nous incluons à la suite la bibliothèque *JQuery* et le plugin *DataTables* en passant par des CDN :

```
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<link rel="stylesheet" type="text/css" href="https://cdn.datatables.net/v/bs4/dt-
1.10.22/datatables.min.css"/>
<script type="text/javascript" src="https://cdn.datatables.net/v/bs4/dt-
1.10.22/datatables.min.js"></script>
```

Il reste maintenant à « transformer » le tableau HTML en un objet *DataTables*. Cela doit être réalisé une fois la page chargée (dans la méthode `ready`) et après les inclusions précédentes :

```
<script>
$(document).ready(function (){
        $('#liste').DataTable({ "pageLength": 10 });
    });
</script>
```

Il peut y avoir toute une série d'options spécifiées dans le constructeur. Ici, nous spécifions simplement le nombre de lignes affichées par défaut (ici 10).

Maintenant que la vue est prête, nous pouvons ajouter la route dans le script `web.php` (dans le répertoire `routes/`) :

```
Route::get('/listdatatables', function() {
        $actualites = Actualite::orderBy('date', 'desc')->get();
        return view('actualites.listdatatables', ['actualites' => $actualites]);
    });
```

Allez sur la page http://localhost/nomprojet/public/listdatatables. Vous pouvez tester les différentes fonctionnalités offertes par *DataTables* :

- Faites une recherche en tapant un mot dans la barre de recherche
- Changer l'ordre de tri du tableau en cliquant sur un titre d'une des deux colonnes
- Modifier le nombre d'éléments affichés (dans la liste déroulante)
- Testez la pagination (en-dessous de la table, vous pouvez cliquer sur les différentes pages)

Il existe une multitude d'autres options qu'il n'est pas possible d'expliquer dans cette fiche. Consultez la documentation officielle, ainsi que les différents exemples pour avoir une idée de ce que permet le plugin. Une option qui est souvent utilisée est la personnalisation des messages (qui sont en anglais par défaut). Pour cela, il faut créer un fichier de langue et l'indiquer lors de la création de l'objet *DataTables*.

Créez un répertoire JSON dans le répertoire `public/` de l'application. Créez un fichier `french.json` qui contient l'objet décrit en annexe. Ajoutez l'élément suivant (après *PageLengthi* et sans oublier d'ajouter une virgule) dans la configuration de l'objet *DataTables* :

```
"language": { "url": "{{ asset('json/french.json') }}" }
```

Pour rappel, {{ }} permet de spécifier des commandes *blade*. Le nom du fichier contenant la vue doit donc terminer par `.blade.php`. La fonction `asset` permet de spécifier un chemin avec le répertoire `public/`. Ce répertoire peut contenir plusieurs sous-répertoires : css, js, images, *etc.*

Vous pouvez récupérer ce projet en tapant la commande suivante :

```
git clone --branch datatables https://gitlab-mmi.univ-reims.fr/rabat01/laravel-exemple-
bd exemple
```

3. Utilisation de *DataTables* avec AJaX

L'exemple précédent implique que l'ensemble des données soit présent dans la page, même si le plugin *DataTables* n'affiche pas toutes les lignes. La génération de la page peut être assez longue si le nombre de lignes est grand. Une solution consiste à passer par des requêtes AJaX pour récupérer les données au fur-et-à-mesure.

Dans un premier temps, modifiez le *seeder* des actualités et indiquez 2000 à la place de 20 (fichier `ActualiteTableSeeder.php` situé dans le répertoire `datatabe/seeders/`). Lancez la génération à l'aide de la commande suivante :

```
php artisan db:seed
```

Si vous rechargez la page, vous vous rendrez compte que le temps de chargement est devenu très long, même si la recherche et le passage d'une page à l'autre restent rapides (ces fonctionnalités sont effectuées par le navigateur en *Javascript*, sans passer par des requêtes vers *Laravel* et la base de données).

Pour exploiter AJaX via *DataTables*, il est nécessaire d'ajouter une méthode dans le contrôleur qui sera appelée par les requêtes AJaX. Chaque requête en POST effectuée par *DataTables* contient les éléments suivants :

- `$_POST['start']` : le premier élément affiché
- `$_POST['length']` : le nombre d'éléments affichés
- `$_POST['search']['value']` : le contenu de la barre de recherche
- `$_POST['order']` : un tableau contenant des tableaux avec les éléments `column` (le numéro de colonne) et `dir` (l'ordre de tri qui peut être `asc` ou `desc`).

Ces éléments doivent être traduits pour générer la requête SQL correspondante. La réponse, au format JSON, doit contenir les éléments suivants :

- `data` : les données qui correspondent à un tableau de lignes, chaque ligne contenant un nombre d'éléments égal au nombre de colonnes du tableau
- `recordsFiltered` : le nombre d'éléments obtenus en fonction de la recherche saisie par l'utilisateur
- `recordsTotal` : le nombre d'éléments total (sans la recherche)
- `draw` : si cet élément est envoyé en POST, il doit être retourné tel quel (il est utilisé pour la vérification)

Dans le contrôleur des actualités, ajoutez la méthode suivante (pour le moment, cela ne prend pas en compte le tri sélectionné par l'utilisateur) :

```
public function datatables(Request $request)
{
    $recherche = "";
    if(isset($request['search']['value']))
        $recherche = $request['search']['value'];
    $nbActualites = Actualite::count();  // Le nombre d'actualités dans la table
    $nbRecherches = Actualite::query()
        ->where('titre', 'LIKE', "%{$recherche}%")
        ->orWhere('message', 'LIKE', "%{$recherche}%")
        ->count();                          // Récupère le nombre d'actualités
                                            // correspondant à la recherche
    $actualites = Actualite::query()
        ->where('titre', 'LIKE', "%{$recherche}%")
        ->orWhere('message', 'LIKE', "%{$recherche}%")
        ->offset($request['start'])
        ->limit($request['length'])
        ->orderBy('date', 'desc')
        ->get();                            // Récupère les actualités en fonction des
                                            // paramètres

    $json = [
            "data" => [],
            "recordsFiltered" => $nbRecherches,
            "recordsTotal" => $nbActualites,
            "draw" => $request['draw']
```

```
        ];                                  // Le tableau associatif qui sera retourné
    foreach($actualites as $actualite)
        $json['data'][] = [
                strftime('%d/%m/%Y', strtotime($actualite->date)),
                $actualite->titre
            ];

    return $json;
}
```

Il faut ensuite spécifier les routes dans le script `web.php` (la première instruction remplace celle ajoutée précédemment) :

```
Route::get('/listdatatables', function() {
        return view('actualites.listdatatables');
    });
Route::post('/datatables', [ActualitesController::class, 'datatables']);
```

Le script `listdatatables.blade.php` doit être modifié. La partie PHP qui génère les lignes du tableau doit être supprimée puisque les données seront récupérées via les requêtes AJaX. La configuration de l'objet *DataTables* devient la suivante :

```
$(document).ready(function (){
        $('#liste').DataTable({
                "pageLength": 10,
                "processing": true,
                "serverSide": true,
                "ajax": {
                    "url": "{{ asset('/datatables') }}",
                    "type": 'POST',
                    "headers": {
                        'X-CSRF-TOKEN' : '{{ csrf_token() }}'
                    }
                  },
                "language": { "url": "{{ asset('json/french.json') }}" }
            });
    });
```

Les nouveaux éléments de configuration sont les suivants :

- `processing` : permet d'afficher un message « *Traitement en cours…* » lors du chargement des données
- `serverSide` : indique que *DataTables* fonctionne en mode *Serveur-side* (les calculs sont effectués du côté serveur)
- `ajax` : la configuration de la requête AJaX. On indique l'URL et le type de requête (POST). On doit également spécifier le jeton de vérification de *Laravel* car dans le cas contraire, la requête serait refusée (voir les TP).

Après avoir effectué ces modifications, vous pouvez tester le bon fonctionnement de *DataTables*.

Il reste maintenant à gérer le tri. Pour cela, nous devons convertir le tableau envoyé par *DataTables* en une ligne à passer à la requête. Dans la méthode `datatables` du contrôleur, nous ajoutons les instructions suivantes avant la requête :

```
$orderBy = "";
if(count($request['order']) > 0) {
    $nbT = 0;
    $colonnes = [ 0 => 'date', 1 => 'titre' ];
    foreach($request['order'] as $elm) {
        if(isset($colonnes[$elm['column']])) {
            $orderBy .= $colonnes[$elm['column']]." ";
            if($elm['dir'] == "desc")
                $orderBy .= "DESC";
            else
                $orderBy .= "ASC";
```

```
            if($nbT < count($request['order']) - 1) $orderBy .= ", ";
                $nbT++;
        }
    }
}
```

La requête devient alors :

```
$actualites = Actualite::query()
    ->where('titre', 'LIKE', "%{$recherche}%")
    ->orWhere('message', 'LIKE', "%{$recherche}%")
    ->offset($request['start'])
    ->limit($request['length'])
    ->orderByRaw($orderBy)
    ->get();
```

Vous pouvez maintenant tester le bon fonctionnement en cliquant sur le titre de la colonne « *Titre* » une première fois, puis une deuxième fois (il y a un tri qui est effectué puis un tri inverse). Vous pouvez également cliquer sur le titre de la colonne « *Date* » en maintenant la touche *Majuscule* enfoncée : le tri sera effectué d'abord sur la colonne « *Titre* » et en cas d'égalité, c'est le tri de la colonne « *Date* » qui sera pris en compte.

Vous pouvez récupérer ce projet en tapant la commande suivante :

```
git clone --branch datatables-ajax https://gitlab-mmi.univ-reims.fr/rabat01/laravel-exemple-bd exemple
```

Annexes – exercice 1

Le script `index.blade.php` complet pour l'exercice 1 :

```
 @extends('template')

@section('titre')
Accueil
@endsection

@section('contenu')
      <div id="actualite" class="mb-3">
        <h4 id="titre"></h4>
        <div id="texte"></div>
        <i id="date"></i>
      </div>

      <a href="{{url("/actualites")}}">Lister les articles</a>
      <a href="{{url("/actualites/create")}}">Créer un article</a>

      <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
      <script>
function recherche() {
    requete = $.ajax({
        "type" : "GET",
        "url" : "{{ url('/actualites/actualite') }}",
        "dataType": "json"
        });
    requete.fail(function (jqXHR, textStatus, errorThrown){
            alert("Problème de requête AJaX.");
            console.log(jqXHR);
        });
    requete.done(function (response, textStatus, jqXHR) {
            $('#titre').text(response['titre']);
            $('#texte').text(response['message']);
            $('#date').text("(" + response['date'] + ")");
            setTimeout(recherche, 3000);
        });
}
$(document).ready(function (){
        recherche();
    });

      </script>
@endsection
```

Le script complet `web.php` (sans les commentaires) :

```
<?php
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\ActualitesController;

Route::get('/', function () {
    return view('index');
});
Route::get('actualites/actualite', [ActualitesController::class, 'actualite']);
Route::resource('actualites', ActualitesController::class);
```

Annexes – exercice 2

Le fichier `french.json` (à placer dans le répertoire public/JSON) :

```
{
    "processing":    "Traitement en cours...",
    "search":        "Rechercher :",
    "lengthMenu":    "Afficher _MENU_ &eacute;l&eacute;ment(s)",
    "info":           "Affichage de l'&eacute;lement _START_ &agrave; _END_ sur _TOTAL_
&eacute;l&eacute;ment(s)",
    "infoEmpty":      "Affichage de l'&eacute;lement 0 &agrave; 0 sur 0
&eacute;l&eacute;ment(s)",
    "infoFiltered":   "(filtr&eacute; de _MAX_ &eacute;l&eacute;ment(s) au total)",
    "infoPostFix":    "",
    "loadingRecords": "Chargement en cours...",
    "zeroRecords":    "Aucun &eacute;l&eacute;ment &agrave; afficher",
    "emptyTable":     "Aucune donnée disponible dans le tableau",
    "paginate": {
            "first":      "Premier",
            "previous":   "Pr&eacute;c&eacute;dent",
            "next":       "Suivant",
            "last":       "Dernier"
        },
    "aria": {
        "sortAscending": ": activer pour trier la colonne par ordre croissant",
        "sortDescending": ": activer pour trier la colonne par ordre décroissant"
    }
}
```

Le fichier `listdatatables.blade.php` (à placer dans le répertoire `resources/views/actualites/`) :

```
@extends('template')
@section('titre') Liste des actualités @endsection
@section('contenu')
<table class="table table-striped" id='liste'>
  <thead>
    <tr>
      <th scope="col">Date</th>
      <th scope="col">Titre</th>
    </tr>
  </thead>
  <tbody>
<?php
foreach($actualites as $actualite) {
?>
    <tr>
      <td> {{strftime('%d/%m/%Y', strtotime($actualite->date))}} </td>
      <td> {{$actualite->titre}} </td>
    </tr>
<?php
}
?>
  </tbody>
  <tfoot>
    <tr>
      <th scope="col">Date</th>
      <th scope="col">Titre</th>
    </tr>
  </tfoot>
</table>

<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
```

```
<link rel="stylesheet" type="text/css" href="https://cdn.datatables.net/v/bs4/dt-
1.10.22/datatables.min.css"/>
<script type="text/javascript" src="https://cdn.datatables.net/v/bs4/dt-
1.10.22/datatables.min.js"></script>
<script>
$(document).ready(function (){
        $('#liste').DataTable({
               "pageLength": 10,
               "language": { "url": "{{ asset('json/french.json') }}" }
            });
    });
</script>
@endsection
```

Le contenu du fichier `web.php` (répertoire `routes/`) :

```
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\ActualitesController;
use App\Models\Actualite;

Route::get('/', function () {
        return view('index');
    });
Route::get('/listdatatables', function() {
        $actualites = Actualite::orderBy('date', 'desc')->get();
        return view('actualites.listdatatables', ['actualites' => $actualites]);
    });
Route::get('actualites/actualite', [ActualitesController::class, 'actualite']);
Route::resource('actualites', ActualitesController::class);
```

Annexes – exercice 3

Le fichier `listdatatables.blade.php` (à placer dans le répertoire `resources/views/actualites/`) :

```
@extends('template')
@section('titre') Liste des actualités @endsection
@section('contenu')
<table class="table table-striped" id='liste'>
  <thead>
    <tr>
      <th scope="col">Date</th>
      <th scope="col">Titre</th>
    </tr>
  </thead>
  <tbody>
  </tbody>
  <tfoot>
    <tr>
      <th scope="col">Date</th>
      <th scope="col">Titre</th>
    </tr>
  </tfoot>
</table>

<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<link rel="stylesheet" type="text/css" href="https://cdn.datatables.net/v/bs4/dt-
1.10.22/datatables.min.css"/>
<script type="text/javascript" src="https://cdn.datatables.net/v/bs4/dt-
1.10.22/datatables.min.js"></script>
<script>
$(document).ready(function (){
        $('#liste').DataTable({
                "pageLength": 10,
                "processing": true,
                "serverSide": true,
                "ajax": {
                    "url": "{{ asset('/datatables') }}",
                    "type": 'POST',
                    "headers": {
                        'X-CSRF-TOKEN' : '{{ csrf_token() }}'
                    }
                  },
                "language": { "url": "{{ asset('json/french.json') }}" }
            });
    });
</script>
@endsection
```

Le contenu du fichier `web.php` (répertoire `routes/`) :

```
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\ActualitesController;

Route::get('/', function () {
        return view('index');
    });
Route::get('/listdatatables', function() {
        return view('actualites.listdatatables');
    });
Route::post('/datatables', [ActualitesController::class, 'datatables']);
Route::get('actualites/actualite', [ActualitesController::class, 'actualite']);
```

```
Route::resource('actualites', ActualitesController::class);
```

La méthode `datatables` située dans le contrôleur `ActualitesController.php` (situé dans le répertoire `app/http/controllers/`):

```
public function datatables(Request $request)
{
    $recherche = "";
    if(isset($request['search']['value']))
        $recherche = $request['search']['value'];
    $nbActualites = Actualite::count();
    $nbRecherches = Actualite::query()
        ->where('titre', 'LIKE', "%{$recherche}%")
        ->orWhere('message', 'LIKE', "%{$recherche}%")
        ->count();
    $orderBy = "";
    if(count($request['order']) > 0) {
        $nbT = 0;
        $colonnes = [ 0 => 'date', 1 => 'titre' ];
        foreach($request['order'] as $elm) {
            if(isset($colonnes[$elm['column']])) {
                $orderBy .= $colonnes[$elm['column']]." ";
                if($elm['dir'] == "desc")
                    $orderBy .= "DESC";
                else
                    $orderBy .= "ASC";
                if($nbT < count($request['order']) - 1) $orderBy .= ", ";
                $nbT++;
            }
        }
    }
    $actualites = Actualite::query()
        ->where('titre', 'LIKE', "%{$recherche}%")
        ->orWhere('message', 'LIKE', "%{$recherche}%")
        ->offset($request['start'])
        ->limit($request['length'])
        ->orderByRaw($orderBy)
        ->get();

    $json = [
            "data" => [],
            "recordsFiltered" => $nbRecherches,
            "recordsTotal" => $nbActualites,
            "draw" => $request['draw']
        ];
    foreach($actualites as $actualite)
        $json['data'][] = [
                strftime('%d/%m/%Y', strtotime($actualite->date)),
                $actualite->titre
            ];

    return $json;
}
```