TP n°9 - AJaX et DataTables dans Laravel

Site:	Université de Reims Champagne-Ardenne	Imprimé par:	CYRIL RABAT
Cours:	INFO0303 - Technologies Web 2	Date:	mercredi 15 novembre 2023, 14:17
Livre:	TP n°9 - AJaX et DataTables dans Laravel		

Table des matières

1. Pour commencer

Pour ce TP, nous réutilisons le code des actualités du TP précédent. Vous pouvez récupérer le projet qui correspond à l'état attendu à la fin du TP n°8 en utilisant Breeze :
https://gitlab-mi.univ-reims.fr/rabat01/tp8

Pour commencer, nous allons faire des modifications générales sur l'application du TP précédent : nous allons mettre à jour le *template* (pour ajouter du CSS et/ou du *JavaScript* dans certaines vues) et gérer correctement les catégories. Ensuite, nous allons exploiter *AJaX* et la bibliothèque *JavaScript* / *JQuery* nommée *DataTables*.

2. Modification du template

Actuellement, le template (`resources/views/template.blade.php`) ne contient que deux sections : `title` et `content`. Nous souhaitons pouvoir ajouter des liens vers des fichiers CSS et des scripts *JavaScript* dans des vues spécifiques, sans les ajouter dans toutes les pages du site.

1. Dans le *template*, ajoutez la section *Blade* `head` avant la balise `</head>` et la balise `script` à la fin du `body` contenant la section *Blade* `script`.

```
<!DOCTYPE html>
<html lang="fr">
  <head>
...
    @yield('head')
  </head>
  <body>
...
    <script> @yield('script') </script>
  </body>
</html>
```

Pour illustrer l'utilisation de ces sections, nous allons modifier la vue `items.list` et demander la confirmation à l'utilisateur avant la suppression de l'article. Pour cela, nous utilisons les *modals Bootstrap* qui sont des boîtes de dialogue.

2. Commencez par ajouter le code suivant au début de la section *Blade* `content` :

```
<div class="modal fade" id="supprimerModal" tabindex="-1">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="supprimerModalLabel">Suppression d'un article</h5>
        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
      </div>
      <div class="modal-body">
        Êtes-vous sûr de vouloir supprimer cet article ?
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-outline-primary" data-bs-dismiss="modal">Annuler</button>
        <button type="button" class="btn btn-outline-danger" onclick="confirmer('{{url('item/')}}')">Confirmer</button>
      </div>
    </div>
  </div>
</div>
```

Il s'agit d'une boîte de dialogue, cachée par défaut. Le bouton `Confirmer` appellera la fonction *JavaScript* `confirmer`.

3. Modifiez le bouton de suppression de l'article qui appelle maintenant la fonction `supprimer` et qui ouvre la boîte de dialogue (attributs `data-bs-toggle` et `data-bs-target`).

```
<button type="button" data-bs-toggle="modal" data-bs-target="#supprimerModal" onclick="supprimer({{$item->id}})" class="btn btn-sm btn-danger mb-1">
 <i class="bi bi-trash"></i>
</button>
```

Il reste maintenant à lier la bibliothèque *JavaScript* de *Bootstrap* et à ajouter le code *JavaScript*.

4. À la fin de la vue (après la section `content`), ajoutez la section *Blade* `head` :

```
@section('head')
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js" integrity="sha384-C6RzsynM9kWDrMNeT87bh95OGNyZPhcTNXj1NW7RuBCsyN/o0jlpcV8Qyq46cDfL" crossorigin="anonymous"></script>
<script defer src="{{asset('storage/js/delete.js')}}"></script>
@endsection
```

5. Récupérez le fichier delete.js et placez-le dans le répertoire `storage/app/public/js` (normalement, il a été lié lorsque vous avez saisi la commande `php artisan storage:link`).

Vous pouvez maintenant tester et vérifier que la suppression est fonctionnelle.

3. Gestion des catégories

L'association des catégories aux articles nécessite de modifier le contrôleur et différentes vues. Lors de l'ajout ou de la modification, nous proposons de sélectionner les catégories à l'aide d'une liste déroulante. Avec *JavaScript* (et *jQuery*), nous masquons les éléments de la liste des catégories déjà sélectionnés. Pour récupérer la liste des catégories sélectionnées dans le contrôleur, nous créons un tableau de nom `categories`.

Dans les vues `items.create` et `items.edit`, nous avons besoin d'ajouter un lien vers le code *JavaScript* et vers le CDN de *jQuery*. De même, nous aurons besoin d'exécuter du code *JavaScript* à la volée pour présélectionner les catégories.

1. Ajoutez le script categories.js dans le répertoire `storage/app/public/js`.

La fonction `selection` est appelée lorsque l'utilisateur clique sur une catégorie dans la liste déroulante. Elle cache la catégorie sélectionnée et crée un élément HTML `div` qui permet d'afficher la catégorie avec un bouton de suppression. Ce dernier appelle la fonction `deleteCategory` qui ré-affiche la catégorie dans la liste et supprime l'élément `div`. Enfin, la fonction `ajouter` permet d'ajouter manuellement une catégorie : elle sera appelée au moment de l'édition pour masquer les catégories dans la liste tout en créant les éléments `div`.

2. Comme pour les fournisseurs, ajoutez les catégories (triées selon l'intitulé) comme paramètre à la vue dans la méthode `create` du contrôleur des articles.

3. Dans la vue `items.create`, après le champ "fournisseurs", ajoutez le champ "catégories". Le `span` va contenir les catégories sélectionnées ajoutées via *JavaScript*.

```
<div class="mb-3 row">
  <label for="categories" class="col-sm-2 col-form-label">Catégorie(s)</label>
  <div class="col-sm-10">
    <select id="categories" class="form-select" onchange="javascript:selection()">
      <option value="-1">Sélectionnez une catégorie</option>
@foreach($categories as $category)
      <option value="{{$category->id}}">{{$category->title}}</option>
@endforeach
    </select>
    <span id="categoriesList"></span>
  </div>
</div>
```

4. Après la section `content`, ajoutez les liens vers les scripts :

```
@section('head')
<script
  src="https://code.jquery.com/jquery-3.7.1.slim.min.js"
  integrity="sha256-kmHvs0B+OpCW5GVHUNjv9rOmY0IvSIRcf7zGUDTDQM8="
  crossorigin="anonymous"></script>
<script src="{{asset('storage/js/categories.js')}}"></script>
@endsection
```

5. Ajoutez également le script *JavaScript* permettant d'ajouter les catégories présélectionnées :

```
@section('script')
  @if(old('categories') != null)
    $(document).ready(function() {
    @foreach(old('categories') as $category)
      ajouter({{$category}});
    @endforeach
    });
  @endif
@endsection
```

6. Dans la méthode `store` du contrôleur des articles, après la sauvegarde de l'article, ajoutez l'association des catégories grâce à la méthode `attach` :

```
$item->categories()->attach($request->categories);
```

7. Ajoutez un article avec des catégories et vérifiez qu'elles sont bien attachées.

8. Comme pour la méthode `store`, ajoutez les catégories de la base de données comme paramètre de la vue `items.edit`.

9. Dans la méthode `update` du contrôleur, après l'appel à la méthode `update`, ajoutez l'instruction suivante pour mettre à jour les catégories :

```
$item->categories()->sync($request->categories);
```

10. Dans la vue `items.edit`, ajoutez le champ "catégories" et les liens vers les scripts *JavaScript*. La section `script` est un peu différente :

```
@section('script')
  $(document).ready(function() {
  @foreach($item->categories as $category)
    ajouter({{$category->id}});
  @endforeach
  });
@endsection
```

11. Pour finir, il faut supprimer les catégories d'un article lors de sa suppression. Ajoutez le code suivant dans la méthode `destroy` du contrôleur (avant l'appel à la méthode `delete`) :

```
$item->categories()->detach();
```

Une autre solution consiste à spécifier le mode "*cascade*" sur la clé étrangère. En supprimant l'article, le SGBD supprime automatiquement tous les mots-clés associés.

4. Utilisation d'AJaX

Pour illustrer le fonctionnement d'*AJaX*, nous allons simplement afficher un article sur la page d'accueil, au-dessus de la liste. Il est récupéré dynamiquement, choisi aléatoirement et est modifié toutes les trois secondes.

1. Au début de la section *Blade* `content` dans la vue `items.list`, ajoutez le code HTML suivant :

```
<div class="row">
  <div class="offset-sm-3 col-sm-6 mb-3 rounded border border-secondary">
    <strong id="title"></strong> <span id="price"></span><br/>
    <div id="description" style="height: 3em; overflow: hidden;"></div>
    vendu par <em><span id="supplier"></span></em><br/>
    <b>Catégorie(s)</b> : <span id="categories"></span>
  </div>
</div>
```

Nous allons ajouter une fonction `recherche` qui fera l'appel *AJaX*. Elle sera appelée dès le chargement de la page.

2. Dans la section *Blade* `javaScript`, ajoutez le code *JavaScript* suivant :

```
function recherche() {
    let requete = $.ajax({
        "type" : "GET",
        "url" : "{{ url('/item/random') }}",
        "dataType": "json"
        });
    requete.fail(function (jqXHR, textStatus, errorThrown){
            console.log(jqXHR);
        });
    requete.done(function (response, textStatus, jqXHR) {
        $('#price').text("(" + response['price'] + "€)");
        $('#title').text(response['title']);
        let description = $('#description');
        description.text(response['description']);
        let wordArray = description.html().split(' ');
        while(description.prop("scrollHeight") > description.prop("offsetHeight")) {
          wordArray.pop();
          description.html(wordArray.join(' ') + '...');
        }
        $('#supplier').text(response['supplier']['name']);
        $('#categories').text("");
        response['categories'].forEach(element => $('#categories').append(element['title']+" ") );
        setTimeout(recherche, 3000);
    });
}
```

3. Puis celui-ci pour appeler la fonction `recherche` lors du chargement de la page :

```
$(document).ready(function (){
    recherche();
  });
```

Il nous reste maintenant à gérer le côté serveur pour répondre à la requête *AJaX*.

4. Ajoutez la fonction suivante dans le contrôleur des articles (notez les méthodes utilisées) :

```
/**
 * Get a random item.
 */
public function random() {
    return Item::inRandomOrder()->with(['supplier', 'categories'])->first();
}
```

5. Pour que l'URL soit accessible, il faut également ajouter la route dans le fichier `web.php` (avant l'instruction `Route:resource(...)`) :

```
Route::get('item/random', [ItemController::class, 'random']);
```

Vous pouvez tester le bon fonctionnement de la route et de la méthode en saisissant l'URL `/item/random`.

5. Utilisation de DataTables

La bibliothèque *DataTables* permet de mettre en forme des données d'un tableau HTML en ajoutant la pagination, la recherche, le tri, *etc.* Elle est constituée d'une partie CSS et d'une partie *JavaScript* qui est basée sur *JQuery*.

1. Dans la vue `items.list`, ajoutez les instructions suivantes dans la section `head` :

```
<link href="https://cdn.datatables.net/v/bs5/dt-1.13.6/datatables.min.css" rel="stylesheet">
<script src="https://cdn.datatables.net/v/bs5/dt-1.13.6/datatables.min.js"></script>
```

2. À la place de l'élément `ul`, nous utilisons maintenant l'élément `table`. Récupérer le code list_extrait.blade.php et remplacez le code dans la vue.

3. Pour "activer" *DataTables* sur le tableau, ajoutez le code suivant dans la section *Blade* `javascript` :

```
$(document).ready(function (){
        $('#listeItems').DataTable({
                "pageLength": 10,
                "stateSave": true,
                "columnDefs": [
                  { "targets": [2], "orderable": false, "searchable": false }
                ]
            });
    });
```

4. Modifiez le *seeder* des articles : au lieu d'en créer 10, indiquez 100 (voire plus si vous êtes aventureux). N'oubliez pas de relancer le peuplement.

5. Modifiez la méthode `index` du contrôleur pour récupérer tous les articles (au lieu de 10) :

```
$itemList = Item::orderBy('title', 'desc')->get();
```

5. Vérifiez que tout est fonctionnel : testez le tri en cliquant sur l'en-tête d'une colonne, faites une recherche, changez de page.

6. Pour mettre en français les contrôles de *DataTables*, récupérez le fichier french.json et placez-le dans le répertoire `storage/app/public.js`.

7. Dans le JSON utilisé pour l'initialisation de *DataTables*, ajoutez le champ suivant :

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

6. AJaX et DataTables avec YajraBox

Comme vous l'avez sans doute constaté, le temps de chargement de la page peut être long car tous les articles sont mis en mémoire, même s'ils ne sont pas tous affichés. Nous allons donc utiliser *DataTables* avec *AJaX*. Il est possible de le faire manuellement ou bien d'utiliser la bibliothèque YajraBox.

1. Dans le répertoire de votre application, tapez la commande suivante :

```
composer require yajra/laravel-datatables-oracle:"^10.3.1"
```

2. Dans la vue `items.list`, supprimez le contenu entre les balises `<tbody>` et `</tbody>` (il sera maintenant récupéré via *AJaX*).

Nous allons maintenant modifier la méthode `index` du contrôleur. Il est possible de savoir s'il s'agit d'une requête *AJaX* grâce au paramètre `$request` (de type `Request`) : la méthode `ajax` retourne `true`.

3. Remplacez le code présent par celui-ci :

```
if($request->ajax()) {
    $model = Item::query()->with('supplier')->with('categories');
    return Datatables::of($model)
        ->addColumn('actions', function($row) {
                return "";
            })
        ->make(true);
 }

 return view('items.list');
```

On récupère l'ensemble des articles puis on fait appel à la méthode `of` de la classe `Datatables`. La méthode `addColumn` permet d'ajouter une colonne qui correspond ici aux boutons (pour le moment, nous retournons une chaîne vide).

4. Ajoutez l'instruction `use` pour utiliser la classe `Datatables` :

```
use Yajra\DataTables\Facades\DataTables;
```

5. Dans la vue, remplacez le code *JavaScript* concernant *DataTables* par celui-ci :

```
$(document).ready(function (){
    $('#listeItems').DataTable({
      "processing" : true,
      "serverSide" : true,
      "columns": [
          { "data": "title", "name": "title" },
          { "data": "price", "name": "price" },
          { "data": "actions", "name": "actions", "orderable": false, "searchable": false }
        ],
      "ajax": {
        "url": "{{ url('/item') }}"
      },
      "language": { "url": "{{ asset('storage/js/french.json') }}" },
      "pageLength": 10,
      "stateSave": true
    });
  });
```

Plusieurs changements sont à noter : l'attribut `column` indique les colonnes présentes avec leur nom, l'attribut `ajax` indique l'URL à appeler et l'attribut `serverSide` indique que *DataTables* utilise *AJaX*.

6. Rechargez la page et vérifiez que tout est fonctionnel.

Par défaut, seul l'intitulé des articles est affiché, mais il manque la description, son fournisseur et ses catégories.

7. Ajoutez le code suivant sur la méthode `of` :

```
->addColumn('title', function($row) {
    $content = "<strong>{$row['title']}</strong> ";
    if(strlen($row['description']) > 50)
      $content .= mb_substr($row['description'], 0, 50)."...";
    else
      $content .= $row['description'];
    $content .= "<br/>vendu par <em>{$row->supplier->name}</em><br/><strong>Catégories</strong> : ";
    foreach($row->categories as $category)
      $content .= $category->title." ; ";
    return $content;
  })
```

La colonne contient du code HTML qui n'est pas interprété par défaut. Il faut donc ajouter l'appel à la méthode `rawColumns` et lui passer en paramètre un tableau contenant les colonnes à interpréter.

8. Ajoutez le code ci-dessous sur la méthode `of` et vérifiez que les données s'affichent maintenant correctement.

```
->rawColumns(['title'])
```

Si vous cliquez sur la colonne `Article` sur la page web, vous pouvez remarquer que le tri ne fonctionne plus. Il faut donc ajouter une nouvelle méthode : `orderColumn`.

9. Ajoutez le code ci-dessous sur la méthode `of` et vérifiez que le tri fonctionne.

```
->orderColumn('title', function($query, $order) {
    $query->orderBy('title', $order);
  })
```

10. Modifiez le code pour que le prix soit affiché avec le symbole € (indication : vous devez utiliser à la fois `addColumn` et `orderColumn`).

Pour finir, nous allons ajouter les boutons.

11. Modifiez l'ajout de la colonne actions par le code suivant (n'oubliez pas d'ajouter la colonne dans le tableau passé en paramètre de la méthode `rawColumns`) :

```
->addColumn('actions', function($row) {
    $buttons =
        "<a href=\"".route('item.show', ['item' => $row['id']]).
        "\" class=\"btn btn-sm btn-primary mb-1 me-1\">".
        "<i class=\"bi bi-eye\"></i></a>";
    return $buttons;
}
```

Nous allons ajouter les deux autres boutons, à condition que l'utilisateur soit logué.

12. Ajoutez l'instruction `use` pour utiliser la classe `Auth` :

```
use Illuminate\Support\Facades\Auth;
```

13. Puis modifiez le code qui permet de créer les boutons en ajoutant les instructions suivantes :

```
if(Auth::check())
    $buttons .=
        "<a href=\"".route('item.edit', ['item' => $row['id']]).
        "\" class=\"btn btn-sm btn-primary mb-1 me-1\">".
        "<i class=\"bi bi-pencil-square\"></i></a>".
        "<button onclick=\"supprimer({$row['id']})\" data-bs-toggle=\"modal\" ".
        "data-bs-target=\"#supprimerModal\" class=\"btn btn-sm btn-danger mb-1 me-1\">".
        "<i class=\"bi bi-trash\"></i></button>";
```

Il reste un dernier élément à prendre en compte : la recherche. Si vous saisissez un texte, vous pouvez voir que cela n'a pas d'effet. Il faut ajouter encore un appel à une méthode : `filter`. Pour simplifier, nous allons faire la recherche uniquement sur l'intitulé de l'article et sur sa description.

14. Ajoutez le code ci-dessous et vérifiez que la recherche est fonctionnelle. Notez l'utilisation de `like` et des symboles `%` qui permettent de chercher une chaîne dans un champ en SQL. L'attribut `search` contenu dans la requête est envoyé par *DataTables* (il s'agit de la chaîne saisie par l'utilisateur).

```
->filter(function ($query) {
        $query->where('title', 'like', "%" . request('search')['value'] . "%")
        ->orWhere('description', 'like', "%" . request('search')['value'] . "%");
    })
```

Normalement, tout est maintenant fonctionnel. Il faut savoir que *DataTables* et la bibliothèque *YajraBox* permettent de faire bien plus. Mais pour le projet, c'est amplement suffisant...