Une liste de choix dynamique

On m'a demandé récemment comment créer simplement une liste de choix dynamique dans un formulaire avec Laravel. Je me suis dit de prime abord que ce devait être quelque chose de très simple et j'ai orienté cette personne vers des exemples existants, par exemple celui-ci. Je n'avais pas regardé de très près le code mais en y revenant je me suis aperçu d'une part qu'il était incorrect et que d'autre part il était incomplet.

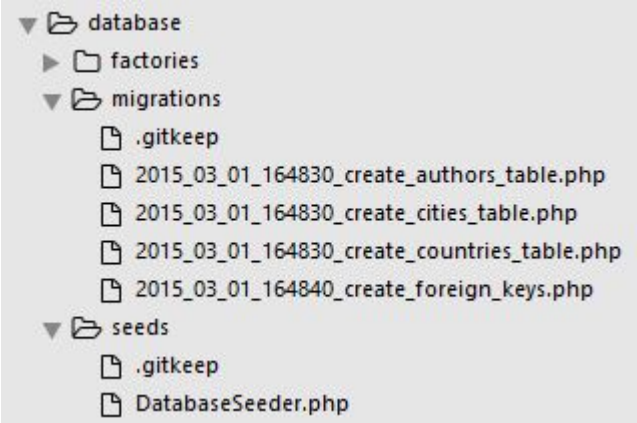
Je me suis donc dit que ça serait peut-être une bonne chose de faire un article sur le sujet en présentant les problématiques rencontrées et une façon de les résoudre.

Une base d'exemple

Je suis parti des 3 tables de mon exemple Les Relations avec Eloquent. Je n'ai conservé que les trois tables : countries, cities et authors :

Un pays a plusieurs villes et une ville a plusieurs auteurs.

J'ai sélectionné les migrations et seeds suffisants pour ces tables :



Migration pour la table authors

```
<?php

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

class CreateAuthorsTable extends Migration {

        public function up()
        {
                Schema::create('authors', function(Blueprint
$table) {
                        $table->increments('id');
                        $table->timestamps();
                        $table->string('name')->unique();
                        $table->integer('city_id')->unsigned();
                });
        }

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

Migration pour la table cities

```
<?php

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

```
class CreateCitiesTable extends Migration {

        public function up()
        {
                  Schema::create('cities', function(Blueprint
$table) {
                        $table->increments('id');
                        $table->timestamps();
                        $table->string('name')->unique();
                        $table->integer('country_id')->unsigned();
                });
        }

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

Migration pour la table countries

```
<?php

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

class CreateCountriesTable extends Migration {

        public function up()
        {
                  Schema::create('countries', function(Blueprint
$table) {
                        $table->increments('id');
                        $table->timestamps();
                        $table->string('name')->unique();
                });
        }

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

Migration pour les clés étrangères

```
<?php

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

class CreateForeignKeys extends Migration {

        public function up()
        {
                Schema::table('cities', function(Blueprint $table)
{
$table->foreign('country_id')->references('id')->on('countries')
->onDelete('cascade')
->onUpdate('cascade');
                });
                   Schema::table('authors', function(Blueprint
$table) {
$table->foreign('city_id')->references('id')->on('cities')
->onDelete('cascade')
->onUpdate('cascade');
                });
        }

        public function down()
        {
                Schema::table('cities', function(Blueprint $table)
{
$table->dropForeign('cities_country_id_foreign');
                });
                   Schema::table('authors', function(Blueprint
$table) {
$table->dropForeign('authors_city_id_foreign');
                });
        }
}
```

Population des tables

```
<?php
```

```
use Illuminate\Database\Seeder;
use Illuminate\Database\Eloquent\Model;

class DatabaseSeeder extends Seeder {

    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        for ($i = 1; $i < 11; $i++) {
            DB::table('countries')->insert(['name' => 'Country ' .
$i]);
        }
        for ($i = 1; $i < 21; $i++) {
             DB::table('cities')->insert(['name' => 'City ' . $i,
'country_id' => rand(1, 10)]);
        }
        for ($i = 1; $i < 21; $i++) {
             DB::table('authors')->insert(['name' => 'Author ' .
$i, 'city_id' => rand(1, 20)]);
        }
    }

}
```

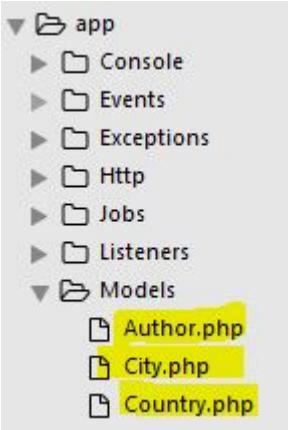
Lancez migrations et seed (**php artisan migrate –seed**), vous devriez avoir ces 4 tables :

Table
authors
cities
countries
migrations

Elles devraient être garnies de façon aléatoire pour les clés étrangères avec des incohérences (une ville peut se retrouver dans deux pays) mais ce n’est pas important pour l’exemple.

Les modèles

Les modèles sont rangés dans un dossier **app/Models** :



Il faut créer les modèles avec les relations correctes.

Le modèle Country

```
<?php namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Country extends Model {

	/**
	 * The fillable attributes.
	 *
	 * @var string
	 */
	public $fillable = ['name'];

	/**
	 * Has Many relation
	 *
	 * @return Illuminate\Database\Eloquent\Relations\hasMany
	 */
	public function cities()
	{
		return $this->hasMany('App\Models\City');
	}

	/**
```

```
         * Has Many Through relation
         *
                                                    *     @return
Illuminate\Database\Eloquent\Relations\hasManyThrough
         */
        public function authors()
        {
                return $this->hasManyThrough('App\Models\Author',
'App\Models\City');
        }

}
```

Le modèle City

```
<?php namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class City extends Model {

        /**
         * The fillable attributes.
         *
         * @var string
         */
        public $fillable = ['name', 'country_id'];

        /**
         * One to Many relation
         *
                                                    *     @return
Illuminate\Database\Eloquent\Relations\BelongsTo
         */
        public function country()
        {
                return $this->belongsTo('App\Models\Country');
        }

        /**
         * Has Many relation
         *
         * @return Illuminate\Database\Eloquent\Relations\hasMany
```

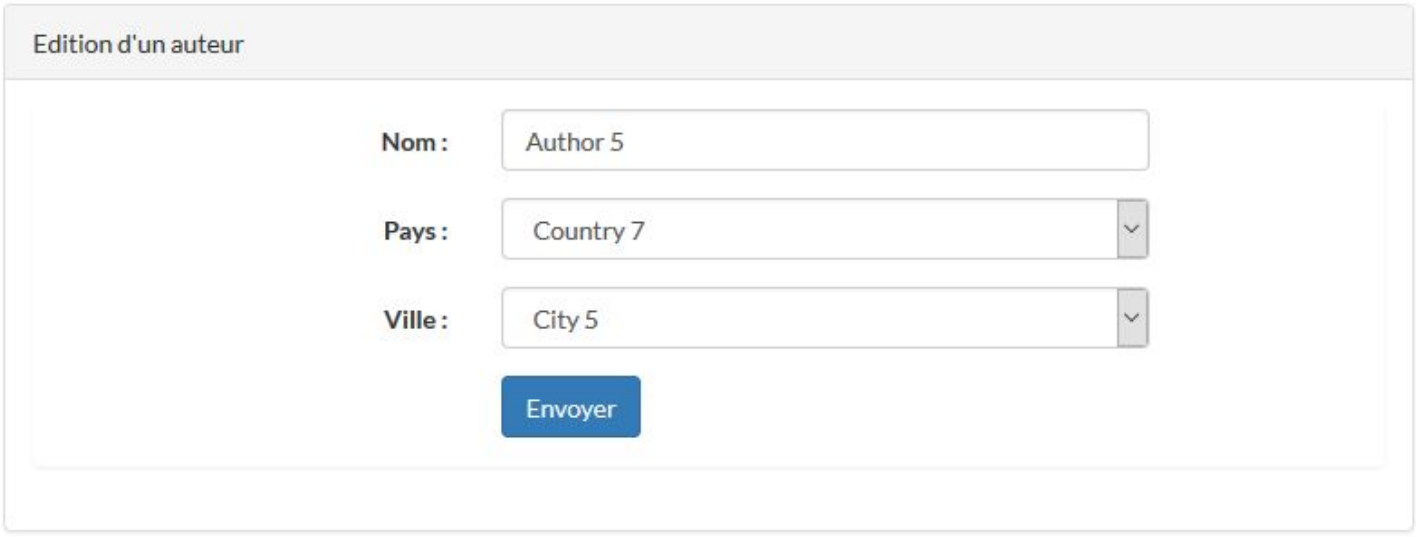
```
         */
        public function authors()
        {
                return $this->hasMany('App\Models\Author');
        }

}
```

Le modèle Author

```
<?php namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Author extends Model {

        public $fillable = ['name', 'city_id'];

        /**
         * One to Many relation
         *
                                                 *     @return
Illuminate\Database\Eloquent\Relations\BelongsTo
         */
        public function city()
        {
                return $this->belongsTo('App\Models\City');
        }

}
```

Le problème

On veut réaliser le formulaire de modification des auteurs qui se présente ainsi :



Au chargement du formulaire on doit avoir :

- le nom de l'auteur
- la liste complète des pays et le bon pays sélectionné
- la liste des villes du pays sélectionné avec la ville de l'auteur sélectionné.

Si on change de pays la liste des villes doit se synchroniser avec le nouveau pays sélectionné. C'est l'objet même de cet article. On va procéder en Ajax pour actualiser cette liste.

Tout ça n'est pas trop compliqué à réaliser mais… il y a aussi la validation à prendre en compte !

Si il y a un souci de validation le formulaire est renvoyé et là on aimerait bien se retrouver avec :

- le nom de l'auteur s'il a changé, ça c'est du classique
- le pays sélectionné, ça c'est déjà moins classique
- la ville sélectionnée avec toutes les villes du pays dans la liste, ça ça va nous poser un problème un peu plus délicat encore parce que la liste est générée dynamiquement.

Les routes

On commence par le plus simple, les routes :

```
Route::get('cities/{id}', 'TestController@cities');
```

```
Route::group(['middleware' => 'web'], function () {

 Route::resource('test', 'TestController', ['only' => [
 'edit', 'update'
 ]]);

});
```

On crée une route pour répondre à la synchronisation des villes : **cities/{id}**, avec l'id du pays. Notez que je n'ai pas mis cette route dans le groupe du middleware « web », on peut la considérer comme une API. On pourrait d'ailleurs utiliser le middleware « api » pour l'occasion.

On crée une ressource avec juste **edit** et **update**.

Si tout se passe bien ça donne ça :

```
+-----------+------------------+-------------+-----------------------------------------------+------------+
| Method    | URI              | Name        | Action                                        | Middleware |
+-----------+------------------+-------------+-----------------------------------------------+------------+
| GET|HEAD  | cities/{id}      |             | App\Http\Controllers\TestController@cities    |            |
| PUT|PATCH | test/{test}      | test.update | App\Http\Controllers\TestController@update    | web        |
| GET|HEAD  | test/{test}/edit | test.edit   | App\Http\Controllers\TestController@edit      | web        |
+-----------+------------------+-------------+-----------------------------------------------+------------+
```

Le contrôleur

Voici le contrôleur :

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;
use App\Http\Controllers\Controller;

use App\Models\Country;
use App\Models\City;
use App\Models\Author;

class TestController extends Controller
{
```

```
    /**
     * Show the form for editing the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function edit($id)
    {
        // Récupération des informations pour le formulaire
        $author = Author::with('city.country')->find($id);
        $countries = Country::all();

        // Envoi du formulaire
        return view('edit', compact('author', 'countries'));
    }

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, $id)
    {
        // Validation
        $this->validate($request, [
            'name' => 'required|max:255'
        ]);

        // Mise à jour de l'auteur
        $author = Author::find($id);
        $author->name = $request->name;
        $author->city_id = $request->city;
        $author->save();

        // Redirection sur le formulaire
         return redirect(route('test.edit', $id))->with('success',
'L\'auteur a bien été mis à jour !');
    }

    /**
     * Get country's cities.
```

```
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function cities($id)
    {
        // Retour des villes pour le pays sélectionné
        return City::whereCountryId($id)->get();
    }
}
```

On a 3 méthodes pour les 3 routes.

edit

Là il faut récupérer les informations dans la base :

```
$author = Author::with('city.country')->find($id);
$countries = Country::all();
```

L'auteur avec le pays parce qu'on a besoin de l'id de celui-ci.

Tous les pays pour remplir la liste.

On envoie tout ça dans la vue **edit** :

```
return view('edit', compact('author', 'countries'));
```

update

Là on commence par la validation :

```
$this->validate($request, [
    'name' => 'required|max:255'
]);
```

J'ai juste demandé la présence du nom et limité sa taille à 255 caractères.

Si la validation passe on met à jour l'auteur :

```
$author = Author::find($id);
$author->name = $request->name;
$author->city_id = $request->city;
```

```
$author->save();
```

Et pour finir on redirige sur le même formulaire avec un message en session flash :

```
return  redirect(route('test.edit',  $id))->with('success', 'L\'auteur a bien été mis à jour !');
```

cities

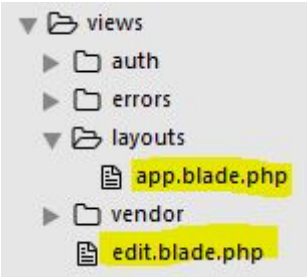
Là c'est la partie API, on renvoie toutes les villes qui correspondent à l'id du pays transmis :

```
return City::whereCountryId($id)->get();
```

Vous voyez que côté Laravel on garde l'esthétique propre à ce framework !

Les vues

On va avoir un template et une vue :



Le template

Là c'est du grand classique :

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
     <meta name="viewport" content="width=device-width, initial-scale=1">
```

```
    <title>Test de liste dynamique</title>

                                                                <link
href="https://fonts.googleapis.com/css?family=Lato:100,300,400,700
" rel='stylesheet' type='text/css'>
                                                                <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstra
p.min.css" rel="stylesheet">

    <style>
        body { Lato'; }
    </style>
</head>

<body>
    <br>
    @yield('content')

                                                            <script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.4/jquery.mi
n.js"></script>

    @yield('scripts')
</body>
</html>
```

La vue

C’est là que se joue le plus délicat. Voici le code complet avec le Javascript intégré :

```
@extends('layouts.app')

@section('content')
<div class="container">
    <div class="row">
        <div class="col-md-10 col-md-offset-1">
            <div class="panel panel-default">
                    <div class="panel-heading">Edition d'un
auteur</div>

                <div class="panel-body">
                    @if (session()->has('success'))
```

```
                        <div class="alert alert-success"
role="alert">{{ session('success') }}</div>
                    @endif
                            <form method="POST" action="{{
route('test.update', $author->id )}} " accept-charset="UTF-8"
class="form-horizontal panel">
                        {!! csrf_field() !!}
                          <input name="_method" type="hidden"
value="PUT">

                        <div class="form-group ">
                             <label for="name" class="col-md-4
control-label">Nom :</label>
                            <div class="col-md-6">
                                  <input class="form-control"
name="name"  type="text"  id="name"  value="{{  old('name',
$author->name) }}">
                               @if ($errors->has('name'))
                                   <span class="help-block">
                                                <strong>{{
$errors->first('name') }}</strong>
                                   </span>
                               @endif
                            </div>
                        </div>
                        <div class="form-group">
                             <label for="country" class="col-md-4
control-label">Pays :</label>
                            <div class="col-md-6">
                                      <select name="country"
id="country" class="form-control">
                                      @foreach($countries as
$country)
                                            <option value="{{
$country->id }}">{{ $country->name }}</option>
                                      @endforeach
                                  </select>
                            </div>
                        </div>

                        <div class="form-group">
                             <label for="city" class="col-md-4
control-label">Ville :</label>
```

```
                            <div class="col-md-6">
                                 <select name="city" id="city"
class="form-control"></select>
                            </div>
                        </div>

                        <div class="form-group">
                            <div class="col-md-6 col-md-offset-4">
                                 <button type="submit" class="btn
btn-primary">
                                    Envoyer
                                </button>
                            </div>
                        </div>

                    </form>
                </div>
            </div>
        </div>
    </div>
</div>
@endsection

@section('scripts')
<script>
$(function() {

    // Récupération des id pour pays et ville
    var country_id = {{ old('country', $author->city->country->id)
}};
    var city_id = {{ old('city', $author->city->id) }};

    // Sélection du pays
    $('#country').val(country_id).prop('selected', true);
    // Synchronisation des villes
    cityUpdate(country_id);

    // Changement de pays
    $('#country').on('change', function(e) {
        var country_id = e.target.value;
        city_id = false;
        cityUpdate(country_id);
    });
```

```
    // Requête Ajax pour les villes
    function cityUpdate(countryId) {
          $.get('{{ url('cities') }}/'+ countryId + "'",
function(data) {
           $('#city').empty();
           $.each(data, function(index, cities) {
               $('#city').append($('<option>', {
                   value: cities.id,
                   text : cities.name
               }));
           });
           if(city_id) {
               $('#city').val(city_id).prop('selected', true);
           }
        });
    }
});
</script>
@endsection
```

J’ai utilisé du Html classique pour cet exemple sans passer par LaravelCollective.

Le remplissage des pays se fait directement avec Blade :

```
<select name="country" id="country" class="form-control">
    @foreach($countries as $country)
          <option value="{{ $country->id }}">{{ $country->name
}}</option>
    @endforeach
</select>
```

Par contre les villes sont vides au départ puisqu’on va les remplir de façon dynamique :

```
<select name="city" id="city" class="form-control"></select>
```

Le Javascript

Au chargement il nous faut les id du pays et de la ville, soit directement issus de la base, soit les anciennes valeurs issues de la validation :

```
var country_id = {{ old('country', $author->city->country->id) }};
var city_id = {{ old('city', $author->city->id) }};
```

Ensuite on sélectionne le pays et on commande la synchronisation des villes :

```
// Sélection du pays
$('#country').val(country_id).prop('selected', true);
// Synchronisation des villes
cityUpdate(country_id);
```

Synchronisation des villes

Voyons la routine de synchronisation des villes :

```
// Requête Ajax pour les villes
function cityUpdate(countryId) {
    $.get('{{ url('cities') }}/'+ countryId + "'", function(data)
{
        $('#city').empty();
        $.each(data, function(index, cities) {
            $('#city').append($('<option>', {
                value: cities.id,
                text : cities.name
            }));
        });
        if(city_id) {
            $('#city').val(city_id).prop('selected', true);
        }
    });
}
```

On a une classique requête **GET** gérée avec **jQuery**. On commence par vider la liste des villes :

```
$('#city').empty();
```

Puis avec une boucle on crée la nouvelle liste :

```
$.each(data, function(index, cities) {
    $('#city').append($('<option>', {
        value: cities.id,
        text : cities.name
    }));
});
```

Pour finir si on est en situation de retour de validation on sélectionne la bonne ville :

```
if(city_id) {
    $('#city').val(city_id).prop('selected', true);
}
```

Changement de pays

Si on change de pays il faut aussi commander la synchronisation des villes :

```
$('#country').on('change', function(e) {
    var country_id = e.target.value;
    city_id = false;
    cityUpdate(country_id);
});
```

Et tout devrait fonctionner !

Conclusion

Ce n’est pas la seule façon de traiter cette situation mais elle me paraît simple et lisible. Une autre façon plus élégante consisterait à faire le traitement de synchronisation uniquement côté client mais ça imposerait d’envoyer aussi toutes les villes dès le départ. Il faudrait alors peut-être oublier jQuery et adopter par exemple Vue.js.