Laravel

Projet CKEDitor.

Auteur : ***Berthe Seydou***

18/08/2016

Table des matières

[Intégration CKEditor: 2](#_Toc459308002)

[1. Création de nouveau projet : 3](#_Toc459308003)

[2. Authentification : 3](#_Toc459308004)

[3. Intégration de CKEditor : 3](#_Toc459308005)

[4. Intégration de Laravel File Manager : 5](#_Toc459308006)

[5. Création de la migration pour contenir les post : 6](#_Toc459308007)

[6. Configuration des routes, Controller et les vues pour le Post : 7](#_Toc459308008)

Prérequis : Projet Laravel Table Et Crud.

Github : [CKEditor](https://github.com/ArianeExtern/EtudeLaravel/tree/CKEditor).

# Intégration CKEditor:

Comme dans la plus part des sites Web, un éditeur de contenu joue un rôle très important dans un site web.

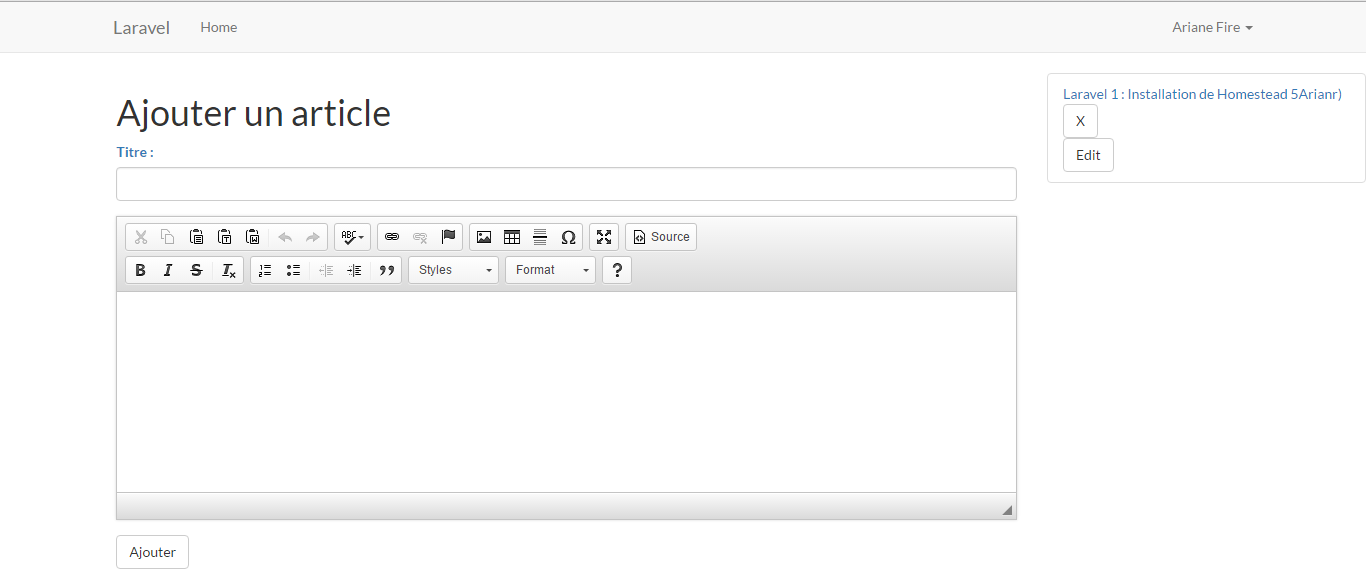
C’est ainsi que dans cette partie nous allons voir comment intégrer un éditeur de contenu dans une Application Laravel.

Pour ce faire, on peut bien partir « **From Scratch** » mais ça ne sert à rien de réinventer la roue étant donné qu’il existe déjà plusieurs composants Laravel qui gère cette tâche, du coup nous allons nous intéresser particulièrement à **CKEditor** à cause de sa facilité de mise en œuvre, de sa flexibilité et de sa bonne documentation.

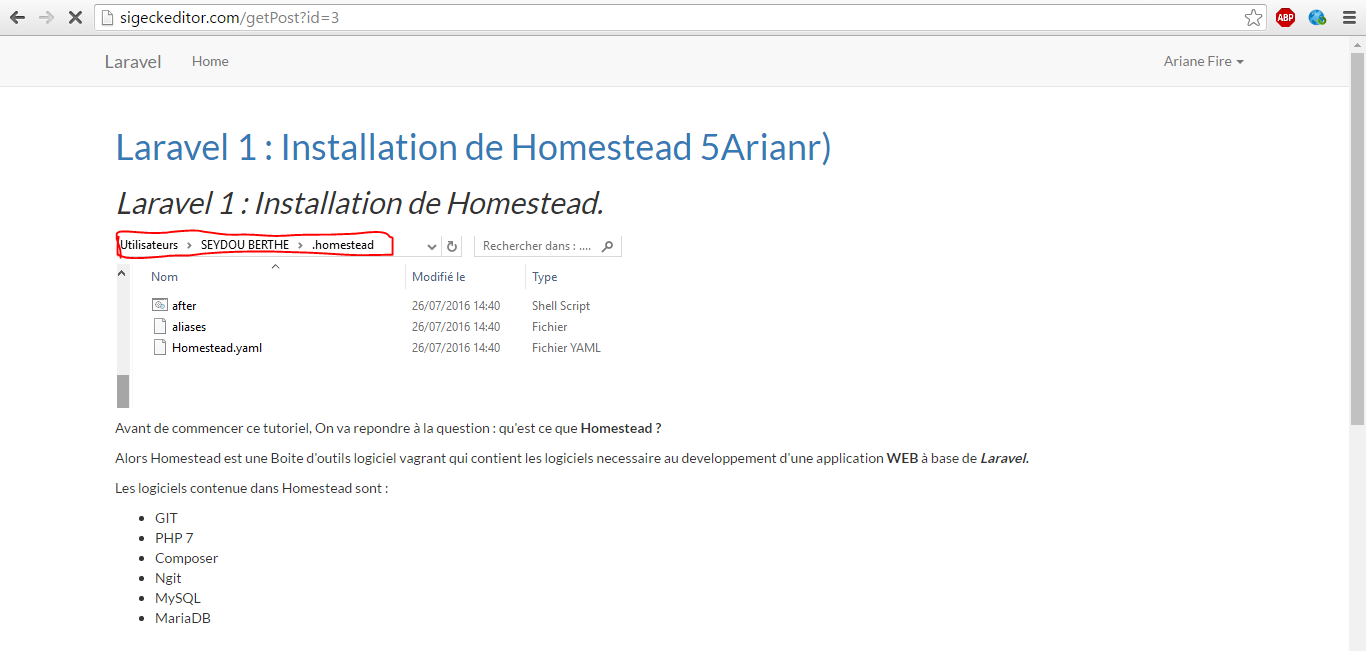
Afin d’aboutir à une application un peu plus pratique, nous allons faire un petit « Editeur de Contenue » c’est-à-dire que l’on va ajouter,et visualiser du contenu et bien évidemment l’ajout se fera à partir de l’éditeur « **CKEditor** » que l’on va intégrer.

Voici ce que l’on aura en fin de ce projet en plus de la page d’accueil, d’inscription et d’authentification :

🡪Page d’ajout de contenu et d’affichage de la liste de contenu (à droite sur l’image):



🡪Page de visualisation de contenu :



Pour atteindre cet objectif, nous allons procéder comme suit :

1. Créer un projet « **CKEditorIntegration** ».
2. Ajouter le composant d’authentification de Laravel.
3. Intégrer **CKEditor** dans l’application.
4. Intégrer le composant « **Laravel FileManger** » pour assurer l’upload et le chargement de fichier.
5. Préparer la base de données pour garder les contenus « Posts ».
6. Configurer les routes et le Controller pour gérer la soumission et l’affichage de nouveau posts.

# Création de nouveau projet :

Commencer par créer un projet nommée « **CKEditorIntegration** », cette nomenclature du projet n’est pas obligatoire, vous pouvez le nommé comme vous le voulez, seulement que vous ayez un nouveau projet Frais.

# Authentification :

Comme Laravel gère déjà bien ça, on utilise le composant d’authentification de Laravel en lançant en ligne de commande étant dans le répertoire racine du projet :

**php artisan make:auth**

On aura ainsi tous les composants nécessaires à l’authentification (Vue, Controller et Migration).

# Intégration de CKEditor :

Comme après la génération des composants d’authentification, on a eu aussi des vues qui ont été générer comme celle affichée après l’authentification « **home.blade.php** » et aussi la page master « **resources/views/layouts/app.blade.php** », c’est deux vues sont ce qui vont nous intéresser plus par la suite.

Pour installer **CKEditor** dans notre application, nous allons procéder comme suit :

1. Installer le package **CKEditor** dans notre application :

Comme tous package Laravel, nous allons utiliser le gestionnaire de package « **composer** » pour installer le package ; Etant placé dans le répertoire du projet, on lance la commande :

**composer require unisharp/laravel-ckeditor**

1. Ajouter les Classes **CKEditor** dans le « Service Provider » de notre application :

Dans le fichier « config/app.php », on ajoute la ligne ci-dessous dans les valeurs du tableau dont la clé est « ‘providers’ => […] » :

**Unisharp\Ckeditor\ServiceProvider::class,**

ce qui ressemblera à

‘porvider’ => [

……..

**Unisharp\Ckeditor\ServiceProvider::class,**

**]**

1. Finalement on publie la ressource en utilisant la commande suivante :

**php artisan vendor:publish --tag=ckeditor**

A ce niveau le composant « **CKEditor** » est prêt à être utiliser.

1. Intégration de **CKEditor** dans la page « **home.blade.php** » :

Pour ce faire commençons par charger le fichier JavaScript de « **CKEditor** » dans la page Master « **resources/views/layouts/app.blade.php** », dans ce fichier on ajoute le code suivant dans la balise **<head>**:

**<script src="/vendor/unisharp/laravel-ckeditor/ckeditor.js"></script>**

A ce niveau on peut afficher CKEditor dans toute nos pages sans problème car le script nécessaire est placé dans la page « master ».

Maintenant dans le fichier « **resources/views/home.balde.php** » ajoutons le code ci-dessous dans la balise <div class= ‘’panel-body’’>….</div> :

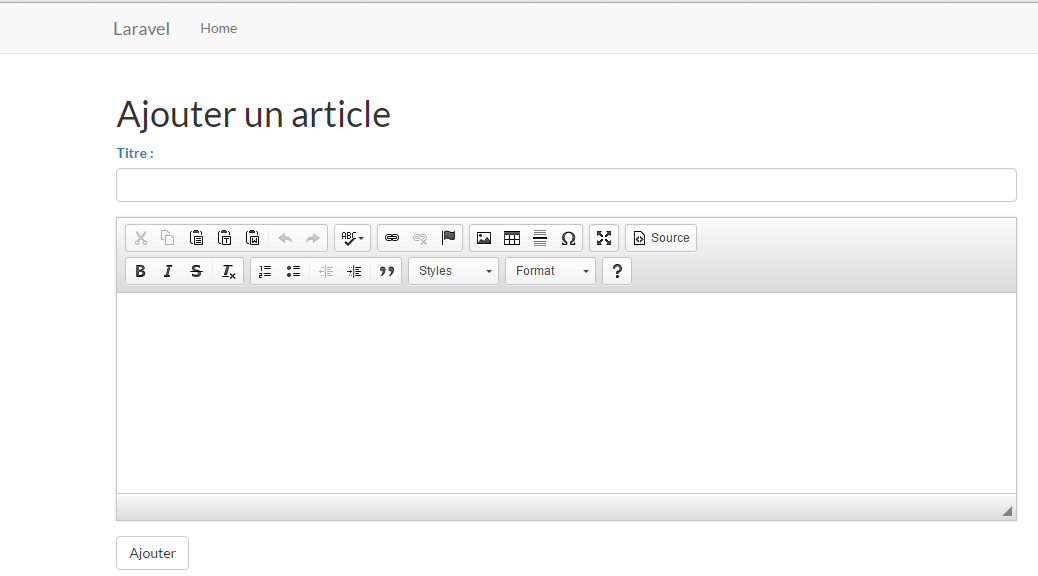
<**div class="row"**>  
 <**div class="col-md-offset-1 col-md-8"**>  
  
 <**h1**>Ajouter un article</**h1**>

<**form method="post" action="**addPost**"**>  
 {!! csrf\_field() !!}  
  
 <**div class="form-group"**>  
 <**label class="text-primary" for="titre"**>Titre : </**label**>  
 <**input type="text" class="form-control" name="titre" id="titre" placeholder="Titre"**/>  
 </**div**>  
  
 <**div class="form-group"**>  
 <**textarea name="textarea" id="textarea"**></**textarea**>  
 </**div**>  
  
 <**div class="form-group"**>  
 <**button class="btn btn-default" type="submit"**>Ajouter</**button**>  
 </**div**>  
</**form**>  
  
<**script**>  
 **CKEDITOR**.replace( **'textarea'**, {  
 **language**: **'fr'**,  
 **uiColor**: **'#DADADA'**,  
});  
</**script**>

</div></div>

Comme on peut le voir, c’est dans la balise <script> que toute la magie s’opère, concrètement ce que le code dans cette balise fait : On remplace le champ <textarea> du formulaire par le composant **CKEditor** en français et la couleur de l’interface du composant est « Gris ».

A ce niveau si on se connecte à l’application dans la page « Home », on voit le formulaire d’ajout de contenu de **CKEditor** comme sur l’image ci-dessous :



A ce niveau vous pouvez d’éditer du contenu mais arrivé au niveau de l’ajout d’une image au contenu ce n’est pas trop pratique du coup on va utiliser un autre composant Laravel pour gérer ça.

# Intégration de Laravel File Manager :

« **Laravel FileManager** » est une application Web gérant l’upload et chargement de fichier. Le choix de ce composant dans notre cas est qui peut jouer ce rôle pour notre **CKEditor**.

Pour intégrer ce composant à notre **CKEditor**, on va suivre les étapes suivantes :

1. Installer le package de « **Laravel File Manager** » : on exécute la commande ci-dessous :

**composer require unisharp/laravel-filemanager**

1. Ajouter « **Laravel File Manager** » et une dépendance dans le Service Provider comme pour « CKEditor », on ajoute dans le fichier « **config/app.php** » la ligne ci-dessous dans les valeurs de la clé ‘provider’ :

**Unisharp\Laravelfilemanager\LaravelFilemanagerServiceProvider::class,**

**Intervention\Image\ImageServiceProvider::class,**

Et dans le même fichier, on ajoute dans la clé ‘alias’ la ligne suivante :

**'Image' => Intervention\Image\Facades\Image::class,**

1. Ensuite on publie ces ressources : en lançant ces deux commandes l’une après l’autre :

**php artisan vendor:publish --tag=lfm\_config**

**php artisan vendor:publish --tag=lfm\_public**

1. A ce niveau un fichier de configuration de «**Laravel  File Manager** » est situé au répertoire « **config/lfm.php** », donc la dernière configuration sera de s’assurer que les ligne suivante ont les valeurs comme ci-dessous :

**'allow\_multi\_user'** => **false**,

Et

**'shared\_folder\_name'** => **'/shares'**,

**NB** : Attention au symbole « **/** » avant « **shares** » ci-dessus, ça doit bien être là quel que soit le nom choisis (« **/shares** » ou «/<votre nom de repertoire> » c’est dans ce répertoire que les images seront stockés à la racine du répertoire « public ».

1. Pour intégrer ce « **File Manager** » dans notre **CKEditor**, on doit intervenir au niveau de la balise <script> qui permet d’afficher le composant **CKEditor** en ajoutant ces lignes de code dans l’objet qui est en 2ième arguments de la méthode CKEDITOR.replace(…), ce qui donne :

**CKEDITOR**.replace( **'textarea'**, {  
 **language**: **'fr'**,  
 **uiColor**: **'#DADADA'**,

//LIGNE DE CODE POUR AJOUTER LARAVEL FILE MANAGER  
 **filebrowserImageBrowseUrl**: **'/laravel-filemanager?type=Images'**,  
 **filebrowserImageUploadUrl**: **'/laravel-filemanager/upload?type=Images&\_token=**{{csrf\_token()}}**'**,  
 **filebrowserBrowseUrl**: **'/laravel-filemanager?type=Files'**,  
 **filebrowserUploadUrl**: **'/laravel-filemanager/upload?type=Files&\_token=**{{csrf\_token()}}**'**});

1. Une dernière modification consiste à éditer le fichier « config/lfm.php » plus précisément la ligne :

'middlewares' => ['auth'],

Doit être comme suit dans le cas où vous utiliser Laravel 5.2 :

'middlewares' => ['auth', ‘web’],

A ce niveau l’édition de contenu marche bien, nous allons donc nous intéresser à la sauvegarde de contenu autrement dit la soumission du formulaire.

# Création de la migration pour contenir les post :

Comme on a pu le remarquer sur le formulaire d’ajout de post possède deux informations essentielles à savoir le titre et le contenu, pour créer cette migration on va lancer la commande suivante :

**php artisan make:migration create\_posts\_table –create=posts**

Ceci va nous créer la migration pour les post, on va donc modifier la méthode « up() » de la migration comme suit :

**public function** up()  
{  
 Schema::*create*(**'posts'**, **function** (Blueprint $table) {  
 $table->increments(**'id'**);  
 $table->string(**'titre'**);  
 $table->text(**'body'**);  
 $table->timestamps();  
 });  
}

Ensuite nous faisons la migration avec la commande : « **php artisan migrate** ».

Finalement on crée le Model qui va nous permettre d’interagir avec la table « posts », on lance la commande : **« php artisan make:model Post** ».

# Configuration des routes, Controller et les vues pour le Post :

La base de données étant prêt pour les « posts », nous allons configurer les routes pour gérer la soumission du formulaire et aussi l’affichage d’un post.

1. Modifier le contenu du fichier « **routes.php** » pour y ajouter les 2 routes suivantes :

Route::*post*(**'addPost'**, **'HomeController@store'**);  
Route::*get*(**'getPost'**, **'HomeController@getPost'**);

Il faudra aussi modifier la route d’accès à la route « **/home** » comme suit :

Route::*get*(**'/home'**, **function** () {  
 **return** view(**'pages.home'**, [**'posts'** => \App\Post::*all*()]);  
});

Le but est de retourner la liste de tous les « posts » pour les affichée.

A partir de ces routes, on va donc intervenir au niveau du Controller « **HomeController** » pour y ajouter les 2 méthodes « **store() & getPost()**» :

**public function** store(){  
 $p = request()->all();  
 Post::*create*([**'titre'** => $p[**'titre'**], **'body'** => $p[**'textarea'**]]);  
 **return** redirect(**'/home'**);  
}  
  
**public function** getPost(){  
 **return** view(**'show'**, [**'post'** => Post::->find(request()->get(**'id'**))]);  
}

On remarque que la méthode « **getPost()** » retourne une vue nommé « show => show.blade.php », en effet cette vue permet d’afficher un « post » qui lui est passé en paramètre. Créer une vue « **show.blade.php** » dans le répertoire « **resources/views** » et mettre le contenu suivant :

**@extends('layouts.app')  
  
@section('content')** <**div class="row"**>  
 <**div class="col-md-offset-1 col-md-10"**>  
 <**h1 class="text-primary"**>{{$post->**titre**}}</**h1**>  
 {!! $post->**body** !!}  
 </**div**>  
 </**div**>  
  
 <**hr**/>

@endsection

Finalement on va ajouter un bout de code dans la vue « home.blade.php » pour qu’il puisse afficher la liste des « posts » et lors du click afficher le post en question.

Ainsi dans le fichier « **home.blade.php** » on ajoute dans la section « **@section(‘content’)** » juste après la balise <div> contenant le formulaire.

<**div class="col-md-3"**>  
 <**ul class="list-group"**>  
 **@forelse(**$posts **as** $post**)** <**li class="list-group-item"**>  
 <**a href="/getPost?id=**{{$post->**id**}}**"**>{{$post->**titre**}}</**a**>

</**li**>  
 **@empty** <**span**>Aucun post pour le moment.</**span**>  
 **@endforelse** </**ul**>  
</**div**>

Ces lignes de code ci-dessous permettent d’afficher la liste des « posts ».

A ce niveau tous devra marcher très bien, on peut ajouter du contenu et le visualiser après.