PHP - Laravel : Vue.js & l'Upload d'un fichier

Ce TP viens suite à une demande faites par des étudiants au cours précédent et vous serez très important pour les deux équipes dans vos projets.

Le TP se décline en deux parties :

PARTIE 1 : Installer Vue.js 2 dans Laravel

L'objectif n'est pas ici d'apprendre le framework/la bibliothèque Vue.js mais d'arriver à l'installer et après, pouvoir l'utiliser de façon basique dans vos projets.

Deux méthodes expliquées pour télécharger VueJs en utilisant npm, le compiler avec Laravel Mix puis l'intégrer pour l'utiliser dans un projet Laravel >= 6.x

 # Introduction

Vue.js est un framework JavaScript open-source utilisé pour construire des interfaces utilisateur et des applications web monopages (SPA).

On le trouve intégré par défaut dans les projets Laravel de version < 6 tels qu'on peut le voir au fichier package.json de la version 5.8 :

```
{
   "devDependencies": {
      ... ,
      "vue": "^2.5.17"
   }
}
```

Avec aussi le composant /resources/js/components/ExampleComponent.vue et le code source suivant au fichier /resources/js/app.js :

```
// Importation de vue.js
window.Vue = require('vue');

// Enregistrement du composant ExampleComponent.vue
Vue.component('example-component', require('./components/ExampleComponent.vue').default);

// L'instance de l'application Vue
const app = new Vue({
   el: '#app',
});
```

... Puis à partir de la version 6.x de Laravel, l'échafaudage de l'interface utilisateur (Bootstrap / Vue) a été extrait dans le package laravel/ui pour permettre à l'échafaudage de l'UI d'être développé et versionné séparément du framework principal.

A la suite de ces changements, aucun code Bootstrap / Vue n'est présent dans l'échafaudage par défaut comme on peut le voir au fichier /resources/js/app.js d'un projet Laravel 8.x :

```
require('./bootstrap');
```

Dans ce guide, nous voulons voir deux méthodes pour télécharger, configurer et utiliser Vue js dans un projet Laravel en utilisant npm.

Méthode 1

Commençons par la méthode classique que je résume en trois étapes :

#1. Télécharger Vue.js 2 dans Laravel

Avant de poursuivre, nous devons nous assurer d'avoir Node.js et Npm installés dans notre ordinateur. Nous pouvons vérifier qu'ils sont installés et accessibles à travers la console en exécutant les commandes npm -v et node -v pour voir leurs versions

Nous pouvons obtenir la dernière version de Node.js et Npm en utilisant l'installeur officiel de Node.js (https://nodejs.org/en/download/) pour les plateformes Windows, MacOS et Linux.

Dans un nouveau projet Laravel, nous avons les dépendances (modules Node) suivantes au fichier package.json :

```
{
   "devDependencies": {
      "axios": "^X.xx",
      "laravel-mix": "^X.x.X",
      "lodash": "^X.xx.xx",
      "postcss": "^X.x.xx"
   }
}
```

Pour les installer, on exécute la commande npm suivante pour les télécharger dans le répertoire /node_modules :

```
npm install
```

Avant de télécharger vue.js, nous pouvons voir les tags de distributions dist-tags disponibles en exécutant la commande :

```
npm view vue
```

Comme il nous manque vue.js, nous pouvons l'installer (télécharger) en exécutant la commande npm suivante :

```
npm install vue
```

Cela importe Vue dans le répertoire /node_modules. Le fichier package.json devient :

```
{
   "devDependencies": {
      "axios": "^X.xx",
      "laravel-mix": "^X.x.x",
      "lodash": "^X.xx.xx",
      "postcss": "^X.x.xx"
   },
   "dependencies": {
         "vue": "^2.x.xx" //
   }
}
```

#2. Compiler et intégrer Vue.js dans Laravel

Une fois vue.js téléchargé dans le répertoire /node_modules, nous pouvons l'intégrer dans le projet pour l'utiliser. Pour ce faire, éditons le fichier /resources/js/app.js de la manière suivante :

- Importons Vue
- Enregistrons un composant « MonComponent.vue » (Composant monofichier)
- Initialisons Vue sur l'élément d'identifiant #app

Nous devons donc avoir le code suivant au fichier resources/js/app.js :

```
require('./bootstrap');

// 1. Importation de vue.js
import Vue from "vue"

// 2. Enregistrement du composant "MonComponent.vue"
Vue.component(
   "mon-component",
   require("./components/MonComponent.vue").default
)

// 3. L'instance de l'application Vue
const app = new Vue({
   el : "#app"
});
```

Comme nous avons enregistré le composant Vue « MonComponent.vue », nous devons créer le fichier /resources/js/components/MonComponent.vue où décrire le template, le style et le script

```
<template>
   <!-- Le template -->
   <section class="container" >
      <h1>Mon composant MonComponent.vue</h1>
      <p>Clicks : <strong>{{ clicks }}</strong></p>
      <button @click="ajouteClick" >clique sur moi</button>
   </section>
</template>

<style>
   /* le style */
   .container {
      max-width: 960px;
      margin: 0 auto; padding: 40px;
      text-align: center;
      font-family: calibri, arial;
   }
</style>

<script>
   // Le script
   export default {
      name : "mon-component",
      data () {
         return {
            clicks : 0
         }
      },
      methods : {
         // Incrémentation de "clicks"
         ajouteClick () {
            this.clicks++;
         }
      }
   }
</script>
```

Nous devons aussi appeler la méthode mix.vue() au fichier webpack.mix.js pour installer les dépendances requises pour le support de fichiers *.vue de Vue.js :

```
const mix = require('laravel-mix');

mix.js('resources/js/app.js', 'public/js').vue()
   .postCss('resources/css/app.css', 'public/css', [
      //
   ]);
```

Il ne reste plus qu'à exécuter la commande npm run suivante pour compiler les fichiers JavaScript et CSS avec webpack :

```
npm run dev
```

Pour la production (version minifié des fichiers) :

```
npm run production
```

Le chemin du fichier JavaScript compilé est /public/js/app.js

#3. Utiliser Vue.js sur une vue (template blade)

Maintenant que nous avons les assets JavaScript compilés dans le fichier /public/js/app.js, nous pouvons l'inclure sur une vue (template Blade) pour utiliser de l'application Vue.

Voici une vue /resources/views/welcome.blade.php où on importe le fichier /public/js/app.js et on intègre le composant <mon-component> dans l'élément <section> d'identification #app :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Mon app. VueJs - Laravel</title>
</head>
<body>

<!-- L'élément #app -->
<section id="app" >

    <!-- Le composant "MonComponent.vue" -->
    <mon-component></mon-component>

</section>

<!-- Le script compilé /public/js/app.js -->
<script src="{{ asset('js/app.js') }}" ></script>
</body>
</html>
```

A ce niveau, nous avons terminé la mise en place du framework JavaScript Vue dans le projet Laravel et pouvons l'utiliser aisément.

Méthode 2

La seconde méthode pour intégrer Vue.js dans un projet Laravel consiste à utiliser le package laravel/ui :

#Installer Vue.js avec laravel/ui

A partir de la version 6.x de Laravel, l'échafaudage Bootstrap / Vue a été extrait dans le package laravel/ui que nous pouvons installer dans un projet Laravel en exécutant la commande composer suivante :

```
composer require laravel/ui
```

Une fois le package laravel/ui installé dans le projet, nous pouvons installer l'échafaudage Vue.js en exécutant la commande artisan suivante :

```
php artisan ui vue
```

Cette commande effectue les opérations suivantes :

- Création du fichier (composant Vue) /resources/js/components/ExampleComponent.vue
- Mise à jour des dépences (modules Node) au fichier package.json
- Création du fichier /resources/js/app.js où sont décrit l'importantion de vue.js, l'enregistrement du composant ExampleComponent.vue et l'instance de l'application Vue.

Nous pouvons aussi ajouter l'échafaudage Bootstrap :

```
php artisan ui bootstrap
```

Il ne reste plus qu'à exécuter la commande npm pour télécharger les dépendances dans le répertoire /node_modules :

```
npm install
```

Puis compiler les fichiers CSS et JavaScript :

```
npm run dev
```

#Conclusion

Nous avons vu dans ce guide deux méthodes pour mettre en place la bibliothèque JavaScript Vue.js dans un projet Laravel en utilisant les commandes npm :

La première en 3 étapes, un peu plus manuelle, où nous avons monté toute l'implémentation nous-même

La seconde où nous nous sommes faits aider avec le package laravel/ui qui génère les fichiers en exécutant les commandes artisan ui

A vous maintenant d'utiliser l'une de ces deux pour installer Vue dans Laravel

PARTIE 2 : Uploader un fichier avec une barre de progression

L'objectif de cette partie est d'arrivé à mettre en place un système d'upload de fichier avec une barre de progression pour suivre l'avancement de l'upload dans un projet Laravel - Vue.js

#Introduction

Une barre de progression peut être définie comme un composant qui indique l'état d'avancement d'une opération. Elle peut se présenter de la forme d'une horizontale non colorée, qui se colore au fur et à mesure de l'avancement de l'opération avec le pourcentage de le progression indiqué…

Nous voulons voir dans ce guide comment mettre en place un formulaire d'upload de fichier avec une barre de progression pour suivre l'état d'avancement d'envoi d'un fichier vers le serveur en utilisant le framework JavaScript Vue.js dans un projet Laravel.

Vue.js va nous permettre d'envoyer le fichier vers le serveur via une requête Ajax (Asynchronous JavaScript + XML) avec Axios et suivre la progression de l'upload.

Vous pourrez vérifier la version de Vue.js et Axios en exécutant la commande **npm ls vue axios**

Nous allons monter notre système en commençant par le coté backend Laravel avec les routes et leurs actions puis nous reviendrons au frontend pour le composant Vue.js.

##Laravel : Les routes, le contrôleur et la vue d'upload de fichier

Pour faire fonctionner le système d'upload de fichier au niveau du client comme au niveau du serveur, nous avons besoin de :

1. Deux routes
2. Un contrôleur
3. Une vue (template Blade)

1. Les deux routes

Nous allons utiliser les routes suivantes pour le système d'upload :

1.« /upload » (GET) pour présenter la page du formulaire d'upload de fichier. Elle sera gérée par l'action « show » du contrôleur UploadFileController.php

2.« /upload » (POST) pour sauvegarder le fichier envoyé au niveau du serveur. Elle sera gérée par l'action « store » du contrôleur UploadFileController.php

Générons le contrôleur UploadFileController.php pour ces actions en exécutant la commande artisan suivante :

```
php artisan make:controller UploadFileController
```

Cette commande crée le fichier /app/Http/Controllers/UploadFileController.php que nous allons compléter au point suivant.

Nous pouvons maintenant définir les routes « upload » (GET et POST) au fichier /routes/web.php :

```
<?php

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

// Afficher le formulaire
Route::get("upload", [ UploadFileController::class, "show" ]);

// Enregistrer le fichier uploadé
Route::post("upload", [ UploadFileController::class, "store" ]);
```

2. Le contrôleur

Le contrôleur /app/Http/Controllers/UploadFileController.php décrit les actions « show » et « store » de routes « upload » de la manière suivante :

- « show » retourne la vue /resources/views/show.blade.php qui va présenter le formulaire d'upload avec la barre de progression
- « store » valide le fichier envoyé au serveur puis l'enregistre dans le répertoire /storage/images/

Le code source du contrôleur /app/Http/Controllers/UploadFileController.php :

```
<?php

namespace App\Http\Controllers;
use Illuminate\Http\Request;

class UploadFileController extends Controller
{
        // Affiche le formulaire d'upload
        public function show () { return view("show"); }

        // Valide puis enregistre le fichier envoyé
         public function store (Request $request) {
                // La validation
                $this->validate($request, ["picture" => "bail|required|image|max:1024"]);

                // On enregistre l'image dans le répertoire "images"
                $chemin = $request->picture->store("images");

                // On retourne la réponse en JSON avec le chemin de l'image
                return response()->json(["chemin" => $chemin]);
   }
}
```

3. La vue (template Blade)

L'action « show » du contrôleur UploadFileController.php retourne la vue /resources/views/show.blade.php (template Blade). Cette vue peut se présenter de la manière suivante en y intégrant le composant Vue <upload-file> que nous allons créer au point suivant :

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
   <head>
      <meta charset="utf-8">
      <meta name="viewport" content="width=device-width, initial-scale=1">
      <title>Laravel - Vue.js - Upload + Progress bar</title>
   </head>
   <body>
      <div id="app">
         <h1>Laravel - Vue.js : Upload + Barre de progression</h1>

         <!-- Le composant Vue d'upload de fichier -->
         <upload-file></upload-file>
      </div>

   <!-- Le script app.js -->
   <script type="text/javascript" src="{{ asset('js/app.js') }}" ></script>
   </body>
</html>
```

#Vue.js : Le composant d'upload et la barre de progression

Le composant Vue <upload-file> que nous avons intégré sur la vue show.blade.php va contenir :

- Le formulaire d'upload de fichier et la barre de progression dans le <template>
- La logique d'upload et la gestion de la barre de progression dans le <script>.

Créons le fichier UploadFileComponent.vue pour ce composant dans le répertoire /resources/js/components/

Il faut ensuite enregistrer le composant au fichier /resources/js/app.js

```
require('./bootstrap');
import Vue from "vue"

// Le composant d'upload de fichier
Vue.component(
        "upload-file",
        require("./components/UploadFileComponent.vue").default
)

// L'instance Vue
const app = new Vue({ el : "#app"});
```

Puis exécuter la commande npm run dev pour compiler les fichiers JavaScript dans /public/js/app.js.

Voici le code source complet (<template> et <script>) du composant /resources/js/components/UploadFileComponent.vue que nous allons commenter juste après :

```
<template>

        <!-- Le formulaire d'upload avec la référence "upload_form" -->
        <form v-on:submit.prevent="upload" enctype="multipart/form-data" ref="upload_form" >

                <label for="picture" >Séléctionnez une image</label><br>

                <input type="file" id="picture" name="picture" required accept=".jpg,.png,.gif" >

                <!-- La barre de progression -->
                <div>
                        <progress v-bind:value="pourcentage" max="100" >{{ pourcentage }}
%</progress>
                        </span>{{ (pourcentage > 0) ? pourcentage + ' %' : '' }}
                </div>

                <button type="submit" >Uploader</button>
        </form>

</template>
```

```
<script>
export default {

   data () {
        return {
           // Pourcentage pour la barre de progression
           pourcentage : 0
        }
   },

   methods : {

        // Upload l'image
        upload () {

           // 1. L'objet "formData" avec la référence du formulaire
           let formData = new FormData(this.$refs.upload_form)

           // 2.  Si un fichier est sélectionné = il possède un nom
           if (formData.get('picture').name) {
```

```
                // 3. La configuration pour la requête Ajax avec axios
                let config = {
                    /*   Lors de la progression de l'upload "onUploadProgress", on met à jour
                         le $pourcentage en divisant les bytes envoyées "progressEvent.loaded"
                         par les bytes attendues "progressEvent.total"
                    */
                    onUploadProgress : progressEvent => {
                this.pourcentage = Math.round((progressEvent.loaded * 100) / progressEvent.total)
                    }
                }

            // 4. On envoie la requête Ajax via axios avec les données du formulaire
            axios.post('upload', formData, config).then((data) => {

                // 5. Si l'image est uploadé
                if (data.data.chemin) {
                    // On réinitialise le formulaire et le pourcentage
                    this.resetForm()
                    alert("Image uploadé")
                }
            })
            .catch((error) => {
                this.resetForm()
                alert(error)
            })

        } else {
            alert("Sélectionnez le fichier à uploader")
        }
    },

        // Réinitialise le formulaire et le pourcentage
        resetForm () {
            this.$refs.upload_form.reset()
            this.pourcentage = 0
        }
    }
}
</script>
```

La logique du composant Vue /resources/js/components/UploadFileComponent.vue peut se résumer en ces six points :

1. La donnée pourcentage initialisé à « 0 » représente le niveau de progression de l'upload :

```
data () {
        return {
                pourcentage : 0
        }
},
```

Sa valeur est liée (data binding) à l'attribut « value » de la barre de progression <progress> :

```
<progress v-bind:value="pourcentage" max="100" >{{ pourcentage }} %</progress>
```

2. Lorsqu'on clique sur le bouton « Uploader » après avoir choisi un fichier, le formulaire envoyé est géré par la méthode upload() :

```
<form v-on:submit.prevent="upload" enctype="multipart/form-data" ref="upload_form" >
```

3. Les données du formulaire sont récupérées dans la méthode upload() via l'instance FormData à laquelle on transmet la référence du formulaire « upload_form » :

```
let formData = new FormData(this.$refs.upload_form)
```

4. On configure la méthode onUploadProgress(progressEvent) de la requête Ajax d'axios pour suivre la progression de l'upload et mettre à jour le pourcentage :

```
let config = {
        onUploadProgress : progressEvent => {
                this.pourcentage = Math.round((progressEvent.loaded * 100) / progressEvent.total)
        }
}
```

5. On envoie la requête XHR avec les données du formulaire et la configuration « onUploadProgress » :

```
axios.post('upload', formData, config).then((data) => {
        // ...
}).catch((error) => {
        // ...
})
```

6. Si la requête échoue ou réussit, on réinitialise le formulaire et le pourcentage en appelant la méthode resetForm() :

```
resetForm () {
        this.$refs.upload_form.reset()
        this.pourcentage = 0
}
```