Vue.js & Laravel & Blade

Table des matières

1 Introduction

Vue est un framework Javascript qui permet de faciliter la construction d'interfaces frontend, à l'instar de React ou Angular. Il est utilisé notamment par Netflix, Adobe, Alibaba et Gitlab. Ce framework est relativement plus facile à utiliser que ses concurrents, tout en restant très performant. Vue propose une documentation en français : `https://fr.vuejs.org/v2/guide`

2 Mise en place dans Laravel

2.1 Outils par défaut

Laravel intègre Vue.js par défaut. D'autres outils sont également présents par défaut dans Laravel, et nous pourrons nous en servir avec Vue. Tous ces outils sont précompilés dans le fichier *public/js/app.js* :

Outil	Type	Description
Bootstrap	Framework	Fonctionnalités js
jQuery	Bibliothèque js	Facilités js
Popper	Bibliothèque js	Tooltips, popovers
Lodash	Bibliothèque js	Facilités js
Axios	Bibliothèque js	XMLHttpRequest

Remarque : dans ce cours, nous allons utiliser Laravel-mix pour compiler les fichiers sources js. Les modifications devraient donc être apportées dans le dossier resources/js. Nous n'expliquons pas ici comment procéder, veuillez vous référer à la documentation de Laravel : https://laravel.com/docs/5.8/mix

2.2 Préparation de la vue de travail

Nettoyons un peu la vue *welcome.blade.php* :

```
<!doctype html>
<html lang="fr">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>Vue.js</title>
    </head>
<body>
</body>
</html>
```

Pour l'instant, rien ne s'affiche. Incluons nos outils js (dont Vue.js) dans la vue welcome :

```
<body>
    <script src="{{ asset('js/app.js') }}"></script>
</body>
```

En rechargeant la page, on peut constater en console une erreur levée par la bibliothèque Axios :

CSRF token not found...

Cette erreur, qui concerne la sécurité des requêtes de type Ajax, est corrigée en ajoutant une balise meta dans la vue :

```
<title>Vue.js</title>
<meta name="csrf-token" content="{{ csrf_token() }}">
```

2.3 Installation de l'extension "Vue.js devtools" sur le navigateur

Une extension « Vue.js devtools » est disponible pour Chrome et Firefox. Après installation, une icône apparaît à droite de la barre d'adresse. Pour l'instant, est elle grisée :

2.4 Activation de Vue.js et des outils de développement spécifiques à Vue

En fait, une instance de Vue a déjà été créée à la fin script *app.js*, avec le code suivant :

```
const app = new Vue({
    el: '#app'
});
```

Cela signifie que cette instance de Vue va chercher à se monter sur une élément du DOM possédant l'attribut *id="app"*. On pourra d'ailleurs créer plus tard d'autres instances. Ajoutons une div qui corresponde à cette description :

```
<div id="app"></div>
```

Maintenant, l'icône de l'extension est verte.

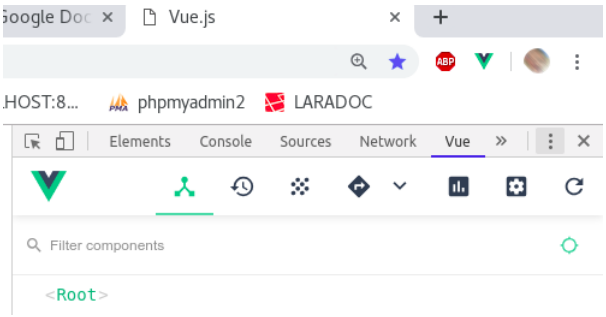
Si vous n'avez pas encore compilé vos sources js avec npm, vous obtenez le message suivant en cliquant sur l'icône verte :

> *Vue.js is detected on this page. Devtools inspection is not available because it's in production mode or explicitly disabled by the author.*

Après avoir exécuté npm install et npm run dev, les scripts js précompilés en mode production sont écrasés avec des scripts en mode développement. Après rafraîchissement, le message change :

> *Vue.js is detected on this page. Open DevTools and look for the Vue panel.*

En affichant les outils de développement du navigateur, on peut observer l'ajout d'un onglet "Vue" :



Pour terminer cette préparation, déplaçons la déclaration de la constante app dans la vue welcome elle-même (en remplaçant const par var, pour garder une cohérence avec le documentation de Vue, et en la supprimant bien du fichier app.js), et exécutons à nouveau npm run dev. On doit obtenir dans l'onglet Vue le même affichage que précédemment.

```
<div id="app"></div>
...
<script>
var app = new Vue({ el: '#app' });
</script>
```

3 Utilisation basique de Vue

A ce stade, nous vous invitons à lire la documentation française de Vue : https://fr.vuejs.org/v2/guide. Ce qui suit est un résumé adapté à l'intégration de Vue dans Laravel, dans le contexte du moteur de gabarits Blade.

3.1 Les « moustaches » pour lier un contenu

Vue effectue, entre autres choses, du rendu de données « classique ». Pour illustrer cette fonctionnalité, nous allons :

1. Déclarer une variable et attribuer une valeur à cette variable,
2. Afficher la valeur de la variable dans un élément HTML,
3. Modifier la valeur de la variable dynamiquement en console.

Pour déclarer et attribuer (1) une variable message, il faut l'ajouter dans l'objet data de l'instance :

```
var app = new Vue({
    el: '#app',
    data: { message: 'Hello Vue!' }
})
```

Remarque : la valeur de la variable peut également être obtenue via une requête Ajax, ou préparée côté serveur avec Blade par exemple :

```
message: {{ $message }}
```

Une fois déclarée, la variable peut être utilisée (2) dans un élément HTML. Mais attention : Vue utilise des "moustaches" identiques à celles de Blade, et le code suivant fait lever une exception par Blade :

```
<div id="app">
    {{ message }}
</div>
```

Heureusement, les concepteurs de Blade ont prévu ce cas de figure : un @ devant les moustaches ouvrantes indique à Blade que l'expression ne doit pas être interprétée, et laissée à Vue (ou à tout autre framework).

```
<div id="app">
    @{{ message }}
</div>
```

Vue lie dynamiquement les éléments du DOM et les données. Le code ci-dessus affiche « Hello Vue ! ». Lorsqu'on tape en console « app.message » (on se souvient que l'on a déclaré une variable *app* qui contient une instance de Vue), c'est également ce message qui s'affiche.

```
> app.message
< "Hello Vue!"
```

Attribuons maintenant en console une autre valeur à notre variable (3). On peut constater que le contenu de la *div* a changé de façon dynamique.

```
> app.message="Wow Vue !"
< "Wow Vue !"
```

3.2 Les types de variables

Vue utilise les types JS standards :

```
message: true // booléen
messages: ['hello1', 'hello2'] // tableau
messages: [ // JSON (Javascript Object Notation)
            { 'id':0, 'content': 'hello0' },
            { 'id':1, 'content': 'hello1' }
          ]
```

Le contenu des tableaux et des JSON est accessible selon la syntaxe JS courante :

```
messages[0] // accès à l'élément du tableau d'index 0
messages[0].content // l'élément du tableau d'index 0 est un JSON
                    // qui possède une propriété content
```

3.3 Les directives

Les directives de Vue sont de nouveaux attributs qui s'ajoutent aux attributs natifs des éléments HTML.

3.3.1 Lier des attributs avec la directive *v-bind*

Les moustaches permettent de lier simplement le contenu d'un élément du DOM avec les variables définies dans Vue. En revanche, cette syntaxe ne peut pas être utilisée pour changer les attributs HTML. Pour cela, il faut utiliser une directive *v-bind*, qui s'utilise à l'intérieur des balises ouvrantes comme un attribut, avec la syntaxe *v-bind :attribut="valeur"*. Voici quelques illustrations :

```
<p v-bind:title="message"></p>
    // affiche le contenu de la variable message au survol.
<p v-bind:id="'par_' + id"></p>
    // Si la variable id vaut "1", l'attribut id estest "par_1"
    // Remarquer la concaténation js avec les guillemets simples.
<input type="text" v-bind:disabled="inputDisabled">
    // Si la variable inputDisabled vaut null, undefined ou false,
    // l'attribut n'est pas inclus dans l'élément input.
<input type="text" v-bind:[attrtype]="inputDisabled">
    // Avec les crochets, on peut également lier les attributs à une variable.
    // Par exemple, attrtype == "disabled"
    // Noter qu'un argument entre crochets doit être en minuscules.
<p :title="message"></p>
    // Ecriture abrégée pour v-bind
```

3.3.2 Affichage conditionnel avec la directive *v-if*

Le code suivant insère ou retire l'élément selon l'état du booléen display :

```
<p v-if="display">Hello</p> // Cette ligne apparaît dans le DOM si display == true
```

3.3.3 Ecouter les évènements avec la directive v-on

Nous verrons plus loin comment définir des méthodes avec Vue. Le code ci-dessous appelle une méthode « doSomething » au clic :

```
<a href="#" v-on:click="doSomething">Lien</a>
```

Comme avec v-bind, on peut lier les attributs à une variable :

```
<a href="#" v-on:[event]="doSomething">Lien</a> // p. ex. avec event = "click"

// Attention, l'écriture abrégée de v-on ne fonctionne pas avec Blade :
<a @:click="doSomething"></a>
```

3.3.4 Générer une liste avec la directive *v-for*

Nous verrons plus loin comment générer une liste avec *v-for*, qui fonctionne comme l'instruction *for* en JS ou *foreach* en PHP :

```
<li v-for="item in items">@{{ item.content }}</li>
...
items: [
    { 'id':0, 'content': 'hello0' },
    { 'id':1, 'content': 'hello1' }
]
```

3.3.5 Générer du HTML avec la directive *v-html*

Les moustaches ont pour particularité d'afficher du texte brut. Le code ci-dessous ne permet donc pas l'interprétation des balises HTML éventuellement contenues dans la chaîne de caractères. La directive v-html résout ce problème.

```
message: '<b>Hello Vue!</b>'
...
<div id="app">
@{{ message }}
    // Les moustaches affichent les caractères bruts : <b>Hello Vue!</b>
<span v-html="message"></span>
    // v-html affiche des caractères gras : Hello Vue!
</div>
```

3.4 Opérations sur les propriétés

Comme l'indique la documentation, on peut utiliser directement la syntaxe JS pour calculer ou modifier la valeur d'une propriété. En termes plus simples, les variables peuvent être calculées ou modifiées avec des fonctions JS à l'intérieur des éléments HTML, et à chaque changement de la propriété, le résultat sera mis à jour.

```
bananas: 0.99
...
<div id="app">@{{ Math.round(bananas) }}
    // la méthode "round" de la classe JS "Math" renvoie la valeur 1
</div>
```

Cette approche a pour inconvénient de compliquer la lecture de la page HTML, notamment avec des opérations longues ou complexes. Mieux vaut alors utiliser les solutions proposées par Vue : les méthodes, les propriétés calculées et les propriétés observées.

3.4.1 Méthodes

Reprenons le calcul ci-dessus en le plaçant dans la méthode *roundBananas* elle-même contenue dans l'objet methods de Vue :

```
@{{ roundBananas() }}
...
var app = new Vue({
    el: '#app',
    data: { bananas: 0.99 },
    methods: {
        roundBananas: function() {
            return Math.round(this.bananas)
        }
    }
});
```

3.4.2 Propriétés calculées

Les méthodes évitent le code verbeux dans les éléments HTML, mais présentent encore un inconvénient : les méthodes sont appelées sur chaque rendu effectué par Vue, ce qui peut être inutilement lourd. C'est pour cela que le framework propose en plus le calcul ou l'observation des propriétés.

A la différence des méthodes, les propriétés calculées sont mises en cache par Vue, et ne sont recalculées que si nécessaire. Dans le code ci-dessous, nous avons remplaçé la méthode par une propriété calculée. Remarquez la suppression des parenthèses à l'intérieur des moustaches : en effet, *roundBananas* n'est plus une méthode, mais bien une propriété dont la valeur est calculée et placée dans le cache avant le rendu. Afin de mieux comprendre le fonctionnement d'une propriété calculée, nous avons ajouté une propriété apples et un affichage du mot « round » en console à chaque exécution de la closure attribuée à *roundBananas*.

```
@{{ roundBananas }} <br> @{{ apples }}
...
var app = new Vue({
    el: '#app',
    data: { bananas: 1.85, apples: 5 },
    computed: {
        roundBananas: function() {
            Console.log('round')
            return Math.round(this.bananas)
        }
    }
});
```

On peut observer en console l'apparition du mot « round » au chargement de la page. En revanche, si on change la propriété *apples* en console, il est intéressant de constater que le mot ne réapparaît pas. Cela est dû à la mise en cache de la valeur de la propriété *bananas*. Comme la propriété *bananas* n'a pas été modifiée, il n'y a pas de raison de la recalculer. Avec une méthode, la propriété *bananas* serait recalculée (et le mot « round » réaffiché) à chaque changement de la propriété *apples*.

De façon générale, il faut préférer les propriétés calculées aux méthodes pour optimiser le rendu.

3.4.3 Propriétés observées

Les propriétés calculées ou les méthodes ne sont pas suffisantes pour gérer les opérations asynchrones d'accès à une API, les opérations très coûteuses en calcul ou avec des données changeantes pour lesquelles il faut réduire la fréquence d'accès. Dans ces cas de figure, il faut utiliser l'option watch qui permet d'oberver les changements d'une propriété. Analysez l'exemple fourni par la documentation officielle de Vue : https://fr.vuejs.org/v2/guide/computed.html#Observateurs