Vue.js : On se lance !

Comme je l'ai dit dans l'article précédent vue.js est une librairie pour créer facilement des interfaces web interactives. Dans cet article nous allons voir dans un premier temps comment l'installer et dans un deuxième temps les concepts clés et un petit exemple de réalisation pour illustrer.

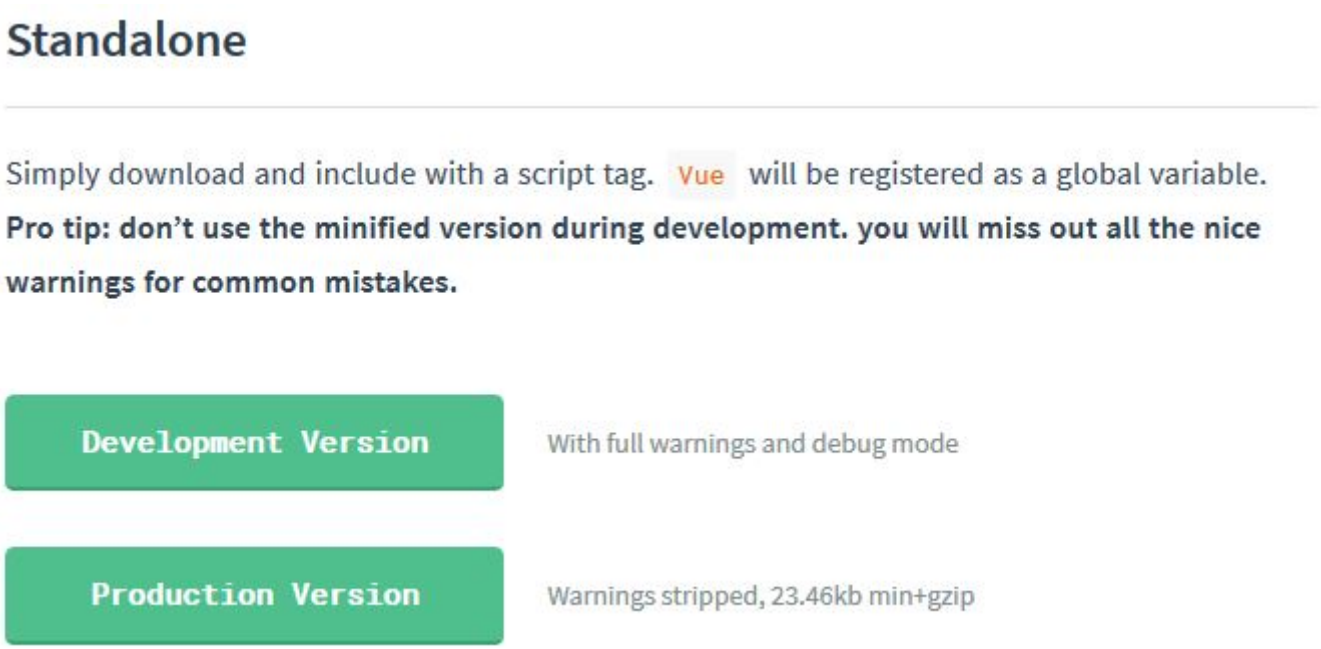
Il existe un site très bien fait auquel vous pouvez vous référer. Mais évidemment tout y est en anglais.

Installation

Pour installer vue.js c'est on ne peut plus simple puisque c'est juste une librairie Javascript. Vous avez donc deux possibilités…

La charger et la mettre sur votre serveur

Il suffit de récupérer la librairie en cliquant sur ce bouton sur le site :



Au moment où j'écris ces lignes la version est la 1.0.16 mais il

est fort probable que vous trouviez une version plus récente.

Vous n'avez plus ensuite qu'à prévoir une balise script :

```
<script src="/js/vue.min.js"></script>
```

Adaptez évidement le chemin selon votre contexte.

Utiliser un CDN

Dans ce cas c'est encore plus simple. vous avez le lien disponible sur le site au même emplacement que vu ci-dessus. Il suffit ensuite de prévoir une balise script :

```
<script
src="http://cdn.jsdelivr.net/vue/1.0.16/vue.min.js"></script>
```

Là aussi il est fort probable que vous aurez une version plus récente.

Le modèle MVVM

Vue.js utilise le modèle **MVVM**, autrement dit **Modèle-Vue-VueModèle**. Voyons ça de plus près. C'est un modèle assez proche du modèle **MVC** (Modèle-Vue-Contrôleur) qui est plus connu, et du **MVP** (Modèle-Vue-Présentation) qui est moins connu. Le but est de bien séparer ce qui est présenté à l'écran (l'interface utilisateur) et la gestion.

Ce modèle a originellement été créé en 2005 par Microsoft pour le WPF et Silverlight. Mais il a ensuite été adopté dans certains frameworks Javascript, comme KnockoutJS.

Le modèle

Le modèle représente les données nécessaires à l'application. Par exemple des données sur des livres : titre, année de parution, auteur, éditeur… Le modèle est chargé de gérer ces données : les mémoriser, les conserver, les restituer. Par contre il n'est pas censé effectuer de traitement sur ces données.

La vue

La vue représente ce qui est accessible à l'utilisateur : l'interface. La principale caractéristique de cette vue c'est qu'elle est interactive. Si l'utilisateur fait une action la vue doit s'adapter en conséquence. Mais ce n'est pas elle qui comporte la logique nécessaire pour le faire, c'est la VueModèle.

La VueModèle

La VueModèle est l'intermédiaire entre la vue et le modèle. Elle transforme les données en éléments d'affichage dans la vue (data-binding) et elle gère les actions de l'utilisateur. La liaison entre la vue et la VueModèle est donc à double sens :

- un changement dans le modèle doit être actualisé dans la vue
- une action dans la vue doit être transmis au modèle

Voici un petit schéma pour illustrer tout ça :

Modèle MVVM dans vue.js

Voyons maintenant comment cela est implémenté dans vue.js.

VueModèle

On va voir ici d'emblée le principal objet de vue.js qui possède le constructeur Vue pour instancier une VueModèle. Donc à la base on va créer un objet :

```
var vm = new Vue({ /* options */ })
```

Cette vue sera associée à un élément du DOM. Je trouve un peu

dommage la confusion générée avec l'appellation « Vue » étant donné que c'est une VueModèle. Mais on va voir à l'usage que ce n'est pas vraiment un souci.

Nous allons voir juste après à quoi correspondent les options transmises.

Vue

J'ai dit ci-dessus que la VueModèle est associée à un élément du DOM. La vue est justement constituée par cet élément du DOM. Il y a deux façons de déclarer cet élément, on peut le transmettre dans les options lorsqu'on crée la VueModèle :

```
var vm = new Vue({
  el: '#element'
});
```

Ou alors on le déclare ensuite :

```
var vm = new Vue();
vm.el = "#element";
```

Modèle

Le modèle est un objet Javascript qu'on va associer à la VueModèle. Là encore on pourra le faire directement dans les options :

```
var vm = new Vue({
  el: '#element',
  data : {}
});
```

Ou ensuite :

```
var vm = new Vue({
  el: '#element'
});
wm.data = {};
```

Voilà les acteurs en place mais pour le moment on ne peut pas en faire grand chose, il va nous falloir des outils pour les mettre

en relation pour créer de l’interactivité.

Les directives

Si vous utilisez AngularJS vous êtes déjà habitué aux directives puisque ici c’est le même principe qui est appliqué, en plus simple.

Une directive est un attribut HTML qui dit de faire une action dans le DOM. Vue.js propose de nombreuses directives et on peut aussi en créer de nouvelles selon les besoins.

Prenons un exemple. Voici la partie HTML :

```
<div id="tuto">
  <p v-text="texte"></p>
</div>
```

Et voici la partie Javascript :

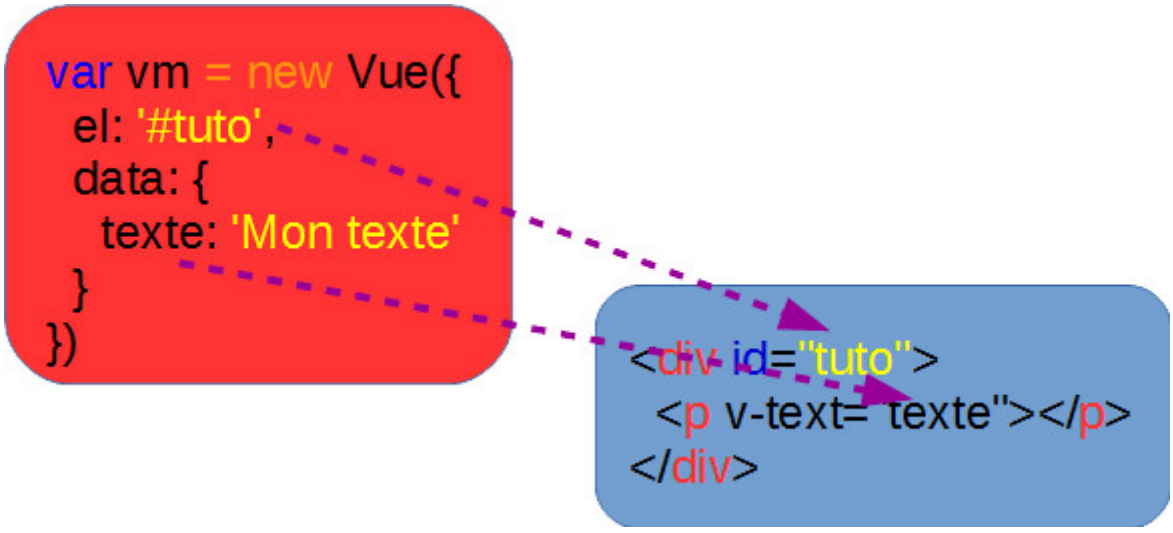
```
var vm = new Vue({
  el: '#tuto',
  data: {
    texte: 'Mon texte'
  }
});
```

On crée une VueModèle vm, on lui associe l’élément du DOM qui a l’identifiant tuto (la vue), et les données data (le modèle).

Dans le HTML on a la directive v-text qui associe la propriété texte du modèle avec le contenu de la balise p du DOM. On va donc avoir comme contenu de la balise la valeur de la propriété texte. Et donc afficher :

Mon texte

Voici une petite illustration des liaisons :



Là où ça devient intéressant c'est que si cette valeur change l'affichage changera également. C'est facile à vérifier en ajoutant cette ligne à la fin du code vu ci-dessus :

```
setTimeout(function(){ vm.texte = 'Mon autre texte'; }, 3000);
```

Maintenant au bout de 3 secondes l'affichage va changer pour :

> *Mon autre texte*

Remarquez comment j'ai référencé la propriété :

```
vm.texte = 'Mon autre texte';
```

On obtient le même résultat que si on avait écrit :

```
vm.$data.texte = 'Mon autre texte';
```

Mais c'est bien plus lisible et pratique ainsi !

Mustache et filtres

Mustache

Rien à voir avec nos moustaches même si justement ce nom vient de la ressemblance antre les accolades et les moustaches (merci **tofdesbois**). Toujours est-il que la syntaxe mustache est devenue une référence pour le templating dans de multiples langages. La base en est une grande simplicité. Reprenons l'exemple vu ci-

dessus mais cette fois en nous passant de la directive :

```
<div id="tuto">
  <p>{{ texte }}</p>
</div>
```

On conserve le même code Javascript et on obtient le même résultat. Le texte ainsi transmis est « échappé », c'est à dire que si vous prévoyez par exemple une balise HTML :

```
data: {
  texte: '<span>Mon texte</span>'
}
```

Le résultat sera rendu en texte :

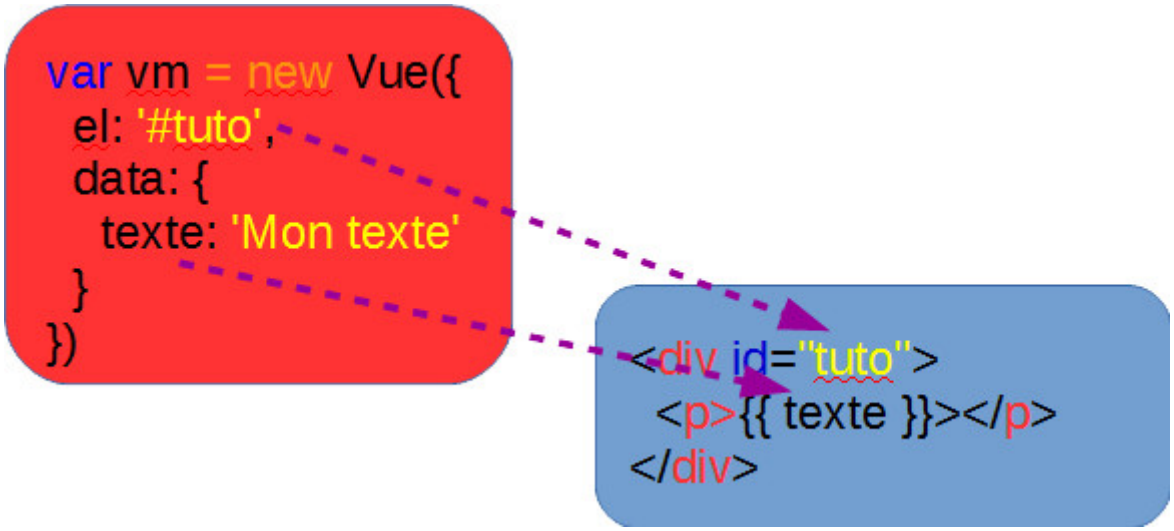
Mon texte

Ceci pour des raisons de sécurité. Si vous tenez absolument à ne pas protéger les données vous pouvez utiliser la syntaxe avec la triple accolade :

```
<p>{{{ texte }}}</p>
```

Mais dans ce cas soyez prudent !

Voici une petite illustration des liaisons :



Les filtres

Un filtre permet d’apporter des modifications aux données avant de les afficher. La meilleure façon de comprendre de quoi il s’agit est de prendre un exemple :

```
<p>{{ texte | uppercase }}</p>
```

On affiche de nouveau notre texte mais cette fois on le veut en capitales. On obtient :

MON TEXTE

Il suffit d’ajouter le signe « | » suivi du nom du filtre. Nous en verrons d’autres dans les prochains articles. Il est aussi possible d’en créer, ce que nous ferons dans la deuxième partie.

Un exemple

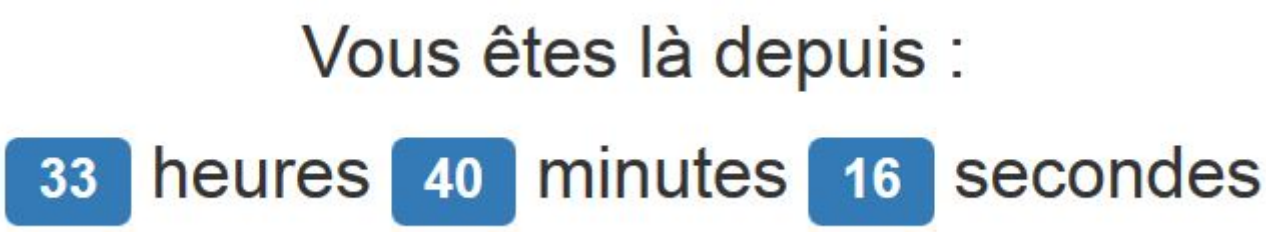
Nous avons à présent suffisamment d’éléments pour construire un premier exemple. Nous allons faire un chronomètre qui se déclenche au chargement de la page et qui affiche les heures, les minutes et les secondes. Donc rien de bien extraordinaire mais on va voir avec quelle facilité on peut le réaliser avec vue.js. Voilà le code complet de la page :

```
<!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 vue.js</title>
                                <link       rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css">
  </head>
```

```
  <body>

    <div class="container">

      <div id="tuto" class="text-center">
        <h1>Vous êtes là depuis :</h1>
        <h1>
           <span class="label label-primary">{{ heures }}</span>
heures
           <span class="label label-primary">{{ minutes }}</span>
minutes
          <span class="label label-primary">{{ secondes }}</span>
secondes
        </h1>
      </div>

    </div>

                                                          <script
src="http://cdn.jsdelivr.net/vue/0.12.8/vue.min.js"></script>

    <script>

      var vm = new Vue({
        el: '#tuto',
        data: {
          heures: 0,
          minutes: 0,
          secondes: 0
        }
      });

      var totalSecondes = 0;
      setInterval(function() {
        var minutes = Math.floor(++totalSecondes / 60);
        vm.secondes = totalSecondes - minutes * 60;
        vm.heures = Math.floor(minutes / 60);
        vm.minutes = minutes - vm.heures * 60;
      }, 1000);

    </script>

  </body>
```

```
</html>
```

Avec cet aspect :



Voyons un peu comment ça fonctionne…

La partie vue est ici :

```
<div id="tuto" class="text-center">
  <h1>Vous êtes là depuis :</h1>
  <h1>
    <span class="label label-primary">{{ heures }}</span> heures
    <span class="label label-primary">{{ minutes }}</span> minutes
     <span class="label label-primary">{{ secondes }}</span>
secondes
  </h1>
</div>
```

On a la DIV englobante avec l'identifiant tuto. On a aussi 3 emplacements « mustache » pour :

- heures,
- minutes,
- secondes

Le reste est juste de la mise en forme boostée par Bootstrap.

La partie Javascript est constituée dans un premier temps par la déclaration de la VueModèle:

```
var vm = new Vue({
  el: '#tuto',
  data: {
    heures: 0,
    minutes: 0,
```

```
    secondes: 0
  }
});
```

On trouve l'élément du DOM désigné par la propriété el. On trouve aussi le modèle avec la propriété data. On trouve au niveau des données les trois entités dont nous avons besoin pour gérer le chronomètre. On sait aussi que si on modifie une valeur dans le modèle ça sera répercuté dans la vue. Il suffit donc de mettre à jour régulièrement ces valeurs pour faire fonctionner le chronomètre.

Cette partie est réalisée avec ce code :

```
var totalSecondes = 0;

setInterval(function() {
  var minutes = Math.floor(++totalSecondes / 60);
  vm.secondes = totalSecondes - minutes * 60;
  vm.heures = Math.floor(minutes / 60);
  vm.minutes = minutes - vm.heures * 60;
}, 1000);
```

On initialise une variable totalSecondes qui va cumuler les secondes qui défilent. Ensuite on met en marche un timer avec un pas de 1 seconde et on actualise à chaque fois les données du modèle.

On n'a pas à se soucier de la manipulation du DOM, ce qui nous incomberait avec juste du Javascript ou avec JQuery. Ici la liaison est automatiquement créée entre le modèle et la vue par la VueModèle.

En résumé

- Vue.js implémente le modèle MVVM.
- Vue.js offre des directives pour agir sur le DOM.
- Vue.js offre la syntaxe mustache et des filtres pour la mise en forme de l'affichage.