VueJS

Les Composants

Maintenant que l'on a vu l'architecture d'un projet VueJS, il est temps d'attaquer le développement. Avant de créer notre premier composant, ou de réutiliser le composant créé, nous allons jouer avec les templates.

Nous allons rester quelques temps sur le composant principal App.vue. Dans un projet complet, vous ne modifierez presque jamais ce composant Mais pour le moment, travaillons dessus. Nous verrons comment créer nos composants et découper notre application plus tard.

Si vous avez un peu inspecté les fichiers, vous avez dû apercevoir un code un peu particulier.

```
{{ msg }}
```

et

```
<HelloWorld msg='Hello world' />
```

Ce balisage particulier est utilisé par différents moteurs de templates (Comme Twig, .Net ou Angular) afin d'afficher le contenue d'une variable. Ici la variable `msg`.

Cette opération est appelée un binding entre la logique et l'affichage :

- Encapsule de petites opérations
- Affiche automatiquement le résultat
- La syntaxe est {{ variableName }}

Enfin, pour afficher cette variable, il faut lui associer une valeur. Pour cela, il existe deux options. Utiliser le `data` ou définir une `props` msg dans le export default de `HelloWorld`.

Avec ces deux morceaux de code, vous venez de voir comment définir une propriété dans une instance de Vue. Et comment afficher cette variable dans le HTML.

Si vous souhaitez avoir accès à une variable à l'intérieur du component `HelloWorld`, ajoutez ceci ligne 36 :

```
data () {
  return {
    age: 30,
    city: 'Lyon',
  }
},
```

Vous aurez alors accès aux variables {{ age }} et {{ city }}.

- Attention `data` est une fonction qui retourne un objet

- `data` est un nom reservé dans cette partie du code logique de VueJS, si vous mettez un autre autre nom, cela ne marchera pas.
- `data` réprésente les variables propres au component.
- `props` est un attribut passé en paramètre de création du component depuis un component parent

Le principe du data-binding est le fait d'associer une variable affichée à l'écran à une variable manipulable en javascript. Les bindings sont des bindings bi-directionnels. Ils conservent la valeur.

A chaque changement dans le code, la valeur est recalculée. Tout comme Angular, ces deux points font toute la force de VueJS.

Les variables sont toujours à la bonne valeur.

Une props quant à elle vient du component parent. Et ne doit pas être modifiée par un component enfant directement. Il est possible d'envoyer un signal au parent avec une nouvelle valeur par exemple. Mais nous verrons celà plus tard.

Maintenant que l'on a vu comment afficher une variable, il est temps de pousser les choses un peu plus loin. Par exemple :

- comment gérer les conditions ;
- Ou gérer les itérations ;
- Ou encore récupérer le contenu d'un input utilisateur ?

Les Directive des composants

Afin de réaliser ces opérations, VueJS possède des attributs HTML particuliers : Des directives.

Préfixées par la lettre `v-`, ces directives nous permettent de créer facilement des pages web :

- v-if
- v-for
- v-model

v-if

Rendu conditionnel avec v-if. Remplacez votre affichage de la variable age par le code suivant :

```
<p v-if="age > 20"> {{ age }} </p>
```

Testez en modifiant la valeur (ou le test) de age.

v-if n'est pas la seule directive de condition :

- v-if

- v-else-if
- v-else

Essayons un exemple un peu plus poussé avec notre variable age.

```
<p v-if="age < 20"> down {{ age }} </p>
<p v-else-if="age > 20 && age < 30"> middle {{ age }} </p>
<p v-else> up {{ age }} </p>
```

Et voilà comment réaliser une condition directement avec VueJS Plus besoin d'accéder à du HTML pour écrire ou supprimer du code. VueJS s'occupe de tout l'affichage. Comme en JavaScript, vous pouvez avoir plusieurs else-if consécutifs.

Il faut aussi noter que le test v-if (entre les guillements avec le mot v-if) est du pur JavaScript. Vous pouvez donc appliquer des fonctions JavaScript

```
v-if="Math.random() === 42"
```

Exercice : 20min

- Remplacez le composant HelloWorld.
  - Retirer l'import
  - Retirer la définition du component
  - Laissez la définition du Helloworld dans le template et regardez la page rendue. Le balise n'est pas visible mais est bien dans le code source de la page.
  - Retirez la balise du template car elle ne servira plus.
- Rajoutez une balise div. Celle-ci doit s'afficher :
  - En testant une variable
  - En testant une valeur fixe
  - En appliquant une fonction JavaScript

v-for

Une deuxième chose que l'on réalise souvent sur du front-end, c'est d'itérer sur un tableau d'éléments :

- Une liste d'utilisateurs
- Une liste de messages
- Une liste de photos

VueJS fournit une directive qui permet de traiter facilement ce cas fréquent; La directive `v-for`.

Dans un premier temps, créons un tableau dans notre objet data.

```
data () {
  return {
    friends: ['riri', 'fifi', 'loulou']
```

```
  }
},
```

Puis ajoutons l'itération au niveau du HTML

```
<ul>
  <li v-for="(friend, index) of friends" :key="index">
    {{ friend }}
  </li>
</ul>
```

De même, vous pouvez utiliser la boucle for... in pour traiter les propriétés des objets. Et vous pouvez travailler sur un tableau d'objets.

Exercice: 30min

- Afficher un tableau de données
- Tester de lire les propriétés d'un tableau d'objets
- Utiliser une condition pour n'afficher que certains éléments

Les évènements

Maintenant que nous avons vu comment traiter l'affichage des données, il est temps d'attaquer les événements en VueJS. Comme en JavaScript, VueJS possède une gestion des événements utilisateurs :

- click
- mouseenter
- mouseout
- ...

Pour détecter et effectuer une opération sur ces événement utilisateur, VueJS possède la directive `v-on`. Cette directive permet d'annoncer que l'on souhaite écouter pour un événement particulier, et ce qui doit être réalisé le cas échéant.

Dans un premier temps, ajoutons la gestion et la détection de l'événement au niveau du HTML. Sur un élément:

```
<button v-on:click="mymethod">Click</button>
```

Dans le component, au même niveau que data, ajoutez la propriété methods. Dans cet objet, vous allez pouvoir définir l'ensemble des méthodes de votre composant

```
methods : {
  mymethod() {
    console.log('clicked');
  },
},
```

Source > exercices/helloWorld/src/components/HelloWorld.vue

L'événement est passé en paramètre de la fonction appelée lors de l'événement. Si vous souhaitez récupérer la valeur, il sufit d'y associer un paramètre.

```
methods : {
  mymethod(event) {
    console.log(event);
  },
},
```

Vous pouvez aussi accéder et modifier les variables de data avec l'écriture this.myVariable

```
methods : {
  mymethod(event) {
    this.age = 30;
  },
},
```

Vous allez souvent utiliser les événements avec VueJS. Comme un bon développeur est content lorsqu'il écrit le moins de code pour un maximum de résultat, vous pouvez utiliser la version rapide pour les événements

```
<button @click="mymethod">Click</button>
```

Exercice Evénements des composants : 20min

Faire un compteur :

- Un bouton pour incrémenter
- Un bouton pour décrémenter
- Tester d'autres événements

Gestion des formulaires

Un dernier point et on aura fait le tour de la gestion des templates avec VueJS. Nous avons déjà vu comme gérer l'affichage, les conditions, les boucles et traiter les événements. Il nous reste à découvrir comment récupérer le contenu des inputs.

Encore une fois, VueJS va nous permettre de gérer ce cas pratique très facilement La directive `v-model` est là pour ça. Cette directive permet de lier / créer une variable qui va être automatiquement liée au contenu de l'input associé.

```
<input type="text" v-model="firstname"/>
```

Il ne vous reste plus qu'à afficher le contenu de la variable pour voir l'effet 'wahou' du binding bi-directionnel en temps réel !

Vous pouvez aussi tester la directive v-model sur d'autres types d'input

Exercice: 40min

Faire une liste de courses :

- Les utilisateurs peuvent ajouter des d'éléments depuis un input
- Si un utilisateur clique sur un élément de la liste, le supprimer
- Ajouter si l'utilisateur appuie sur Enter dans l'input
- Ajouter une validation des données

Source > exercices/courses