

Vue.js

M2i

Alan Piron-Lafleur

Mise en place

C'est un framework frontend qui est relativement facile à prendre en main.
Il permet de découper nos applications en composants réutilisables et de les injecter facilement à des projets existants.

J'utiliserai VSCode pour ce cours, libre à vous de votre IDE.

Créez un dossier coursVueJS sur votre ordinateur, ouvrez le dans votre IDE... c'est parti !

Nous utiliserons les extensions

- ▶ Volar : Facilite le codage en vueJS.
- ▶ Live server : pour refresh automatiquement l'affichage lorsqu'on sauvegarde un fichier.

Le modèle MVVM

Le modèle MVVM : model - view - viewModel est l'architecture logicielle utilisée par Vue.js.

Le M : model

Contient toutes les données liées à la logique métier (base de données, api externes).
Sera en charge de la transmission des données au backend.

Le V : View

C'est l'interface graphique qui fait le lien entre les actions de l'utilisateur et le modèle.
Elle définit où sont placés les éléments graphiques.

Le VM : ViewModel

Fait la transition entre les deux. Un changement dans le modèle (modification du prénom ?) est transmis à la vue par le ViewModel. Un changement sur la vue est transmis au modèle par le ViewModel.



Decouverte

Nous allons commencer avec une série de pratique et d'exercices pour vous montrer Vue.js

Créer un dossier `cours_decouverte` dans votre dossier `CoursVueJS` et placez y un fichier `decouverte-01.html`

Je vous montre comment s'utilise VueJS via le CDN.

Le modèle MVVM

- On tape html:5 pour générer un template de base.
- Rdv sur le site de vueJS -> get started -> quick start pour récupérer le CDN, à mettre dans le <head>
- Voici le code :

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

```
  <h2>Je test {{appName}}</h2>
```

```
</div>
```

```
<script>
```

```
  app = Vue.createApp({
```

```
    data(){
```

```
      return {
```



Du JS de partout

Ecrire entre les accolades s'appelle : utiliser les interpolations de Vue.js
C'est la formation la plus élémentaire de liaison de données du framework.

Il est possible d'écrire du code JS directement dans les accolades.
Je vous montre.

Du JS de partout

Modifier le <h2> comme suit :

```
<h2>Je test {{appName.toUpperCase()}}</h2>
```

Ajouter un paramètre “complexe” et un paragraphe pour tester :

```
return {  
  appName: "mon super projet 1",  
  complexe: false  
}
```

```
<h2>Je test {{appName.toUpperCase()}}</h2>
```

```
<p>Le projet est {{ complexe ? "dur" : "simple" }}</p>
```


Propriétés calculées

Squelette

Pour nos prochains projets, créons un fichier squelette.html

- Copiez collez le fichier decouverte-01.html, nommez le fichier squelette.html
- Enlevez tout ce qui n'est pas générique, votre <body> ressemblera à ça :

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

```
<script>
```

```
  app = Vue.createApp({
```

```
  });
```

```
  app.mount("#app");
```

```
</script>
```

Propriétés calculées

Les fonctions que nous utiliserons dans notre application VueJS ne doivent pas être placées dans le modèle (pas dans `data()`).

Il existe une propriété `computed()` pour cela, c'est le fameux `ModelView`.

Créez un fichier `decouverte-02.html` à partir de `squelette.html`... je vous montre la suite.

Propriétés calculées

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

```
  <h2>{{transformString}}</h2>
```

```
</div>
```

```
<script>
```

```
  app = Vue.createApp({
    data(){
      return {
        string: "Une phrase au hasard"
      }
    },
    computed: {
      transformString(){
```

Exercice

Créez un dossier exercices dans cours_decouverte.

Créez un fichier exercice-01.html et réalisez l’affichage suivant :

“Il est précisément : HH:MM:SS (affichage FR)”

“Il est précisément : HH:MM:SS (affichage US)”

👉 Aide pour coder propre :

- le Model donne les données nécessaires au programme (ici, une date)
- le ModelView utilise la donnée pour la transformer (en français, en US)
- toLocaleTimeString('en-US') est une fonction qui fonctionne !

Correction

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

```
  <p>Il est précisément {{dateFr}} (affichage FR) </p>
```

```
  <p>Il est précisément {{dateUs}} (affichage US) </p>
```

```
</div>
```

```
<script>
```

```
  app = Vue.createApp({
    data(){
      return {
        d: new Date()
      }
    },
    computed: {
```

Data binding



Data binding

Vue.js va nous permettre de binder les attributs de nos éléments html avec des données du modèle. C'est le data binding.

Je vous montre dans un fichier decouverte-03.html

Data binding

```
<div id="app">
   <- on peut enlever le v-bind, ça marche aussi
</div>

<script>
  app = Vue.createApp({
    data(){
      return {
        url: "https://vuejs.org/images/logo.png"
      }
    }
  });
  app.mount("#app");
</script>
```



Data binding

Après le binding sur les attributs, on peut binder du texte brut et html, directement.

A la suite de [decouverte-03.html](#), je vous montre.

Data binding

```
<div id="app">  
  <p v-text="presentation"></p>  
  <p v-html="presentationHTML"></p>  
</div>
```

```
<script>  
  app = Vue.createApp({  
    data(){  
      return {  
        presentation: "Salut, c'est Patrick",  
        presentationHTML: "Salut, c'est <strong>Patrick</strong>"  
      }  
    }  
  });  
  app.mount("#app");  
</script>
```



Data binding

Il est évidemment possible (et souhaitable) d'utiliser des objets JSON provenant du modèle pour les binder dans notre HTML.

Je vous montre.

Data binding

```
<div id="app">  
  <p>Ici c'est <span v-text="personne.prenom"></span>, j'ai <span  
v-text="personne.age"></span> ans</p>  
</div>
```

```
<script>  
  app = Vue.createApp({  
    data(){  
      return {  
        personne : {  
          prenom: "Alan",  
          nom: "PL",  
          age: 31  
        }  
      }  
    }  
  })
```

Modifier des données

Dans Vue.js, il sera simple de modifier l'affichage, après un clic sur un bouton, après le post d'un formulaire, etc ..

Il suffira de changer les propriétés de notre app au moment souhaité.

Voyons cela ensemble.

On met notre app dans une constante :

```
const appVue = app.mount("#app");  
appVue.personne.prenom = "John";
```

Exercice

Créez un fichier html exercice-02.html et réalisez une horloge (qui se met donc à jour automatiquement) :

“Il est précisément : HH:MM:SS”

👉 Aide pour coder propre :

- utilisez la fonction `setInterval()` pour modifier le modèle chaque seconde

Correction

```
<div id="app">
  <p v-html="horloge"></p>
</div>

<script>
  app = Vue.createApp({

    data(){
      return{

        horloge: new Date().toLocaleTimeString()

      }
    }

  });
  const appVue = app.mount("#app");
```

Two way binding

Nous venons de voir le fait de binder des données de notre modèle pour les afficher dans la view.

Nous allons maintenant voir l'inverse : binder un affichage de notre view pour modifier le modèle.

C'est parti dans [decouverte-04.html](#)

Two way binding

```
<div id="app">
  <input type="text" placeholder="Votre nom" v-model="nom">
  Bonjour {{ nom }}
</div>

<script>
  app = Vue.createApp({

    data(){
      return{
        nom: ''
      }
    }

  });
  app.mount("#app");
</script>
```

Two way binding : textarea

Le textarea est un peu spécial, dans le sens où ce qui sera tapé dedans ressortira sans saut de ligne. Je vous montre en copiant/collant le code ci dessous.

```
<div id="app">
  <textarea v-model="unTexte" cols="40" rows="7"></textarea>
  <div>{{ unTexte }}</div>
</div>
```

```
<script>
  app = Vue.createApp({

    data(){
      return{

        unTexte: ""

      }
    }
  })
```

Two way binding : textarea

Pour que l'affichage se passe bien, il faut ajouter à la div qui affiche le texte la propriété CSS : `white-space: pre-line;`

A propos de v-model

Appliqué à un élément html, l'attribut v-model de Vue.js ne remplace pas le même attribut :

- ▶ text et textarea : v-model s'attache à l'attribut value
`<input type="text" v-model="nom" />` *<- modifie l'attribut value de l'input*
- ▶ checkboxes et radiobuttons s'attache à l'attribut checked;
`<input type="checkbox" v-model="isChecked" />` *<- modifie l'attribut checked*
- ▶ select s'attache à l'attribut value de l'option
`<select v-model="selected"> <option>A</option></select>` *<- lorsque l'option sera choisie, selected sera égal à la valeur de l'option, soit A.*

Gestion évènementielle

Utilisation d'événement

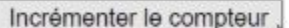
Dans VueJS, et nous l'avons déjà fait, vous pouvez utiliser les directives.

En voici une nouvelle : `v-on:nomévénement`.

Par exemple : `v-on:click` ou `v-on:mousemove`, etc

Exercice pour essayer : Réaliser l'affichage suivant dans un fichier `exercice-03.html`:

Nombre de clics sur le bouton : 7

A rectangular button with a light gray border and a light gray background. It contains the text 'Incrémenter le compteur' in a dark gray font. A mouse cursor is pointing at the bottom right corner of the button.

Je vous montrerai pour le clic sur le bouton

Utilisation d'événement

```
<div id="app">
  <p>Nombre de clics sur le bouton : {{ compteur }}</p>
  <button v-on:click="compteur++">Incrémenter le compteur</button>
// on peut mettre @click plutôt que v-on:click
</div>
<script>
  app = Vue.createApp({
    data(){
      return {
        compteur: 0
      }
    }
  });
  app.mount('#app');
```

Utilisation de l'objet event

Créons un fichier decouverte-05.html pour découvrir l'objet event.

Nous allons afficher les coordonnées du pointer lorsque l'on passe la souris sur une image.

Regardons ensemble.

Utilisation de l'objet event

```
<div id="app">
  
  <div>{{ xy }}</div>
</div>
<script>
  app = Vue.createApp({

    data(){
      return {
        xy: ''
      }
    },
    methods: {
```



Suffixe d'événement

Chaque événement peut être préfixé par un suffixe Javascript. Par exemple, on pourrait faire un `@click.once` pour n'exécuter qu'une fois l'événement.

Je vous montre.

Suffixe d'événement

Dans decouverte-05.html :

```
<div id="app">
  <button @click.once="action">Cliquez ici</button>
  <input type="text" @keyup.enter="action" />
</div>
<script>
  app = Vue.createApp({

    methods: {
      action(){
        console.log("Je suis cliqué !")
      }
    }

  });
```

Suffixe d'événement

Exercice dans `exercice-04.html` :

Affichez une image avec une `width` de 500px.

Au clic gauche sur l'image, passez la `width` à 300px.

Suffixe d'événement

```
<div id="app">
  
</div>
<script>
  app = Vue.createApp({

    data(){
      return {
        width: '300'
      }
    },
    methods: {
```

Les arguments dynamiques

Les arguments dynamiques

Les arguments d'un event peuvent être dynamiquement exprimés, je vous montre comment sur l'exercice 4

Les arguments dynamiques

Sur exercice 04 :

```
<div id="app">
  <select v-model="choix">
    <option value="click">Clic</option>
    <option value="dblclick">Double clic</option>
  </select>
  
</div>

<script>
  app = Vue.createApp({

    data(){
      return{
        width: 500,
```

Directive v-for



Directive v-for

v-for nous permet de boucler dans un tableau. Pour illustrer son utilisation, créons un fichier decouverte-05.html à partir du squelette.html

Je vous montre la suite.

Directive v-for

```
<div id="app">
  <p>Quels langages de prog connaissez vous ?</p>
  <input type="text" v-model="unLang" />
  <input type="button" value="valider" @click="ajouter" />
  <div>{{ langages }}</div>
</div>
<script>
  app = Vue.createApp({
    data(){
      return {
        unLang: '',
        langages: []
      }
    },
    methods: {
```

Directive v-for

Maintenant, plutôt que d'afficher tout le tableau langages, je vais vouloir afficher une liste à puce avec chaque langage, un par un.

Je vous montre.

Directive v-for

A la place de la `<div{{ langages }}</div>`

Je met :

```
<ul>  
  <li v-for="langage in langages">{{ langage }}</li>  
</ul>
```

et si je veux qu'un langage s'ajoute lorsque j'appuie sur "entrer" ?

Je rajoute : `@keyup.enter="ajouter"` sur l'input

Directive v-once

Directive v-once

Nous pouvons nous retrouver dans le cas où nous souhaitons afficher les informations d'une donnée sans pour autant modifier l'affichage dynamiquement.

Par exemple, pour réaliser l'affichage suivant :

Valeur de départ : 15

Valeur finale : XX

<bouton pour incrémenter>

Directive v-once

```
<div id="app">
  <div v-once>Valeur de départ : {{ nombre }}</div>
  <div>Valeur finale : {{ nombre }}</div>
  <button type="button" @click="incrémenter" >Incrémenter</button>
</div>
<script>
  app = Vue.createApp({
    data(){
      return {
        nombre:15
      }
    },
    methods: {
      incrémenter(){
        this.nombre++;
      }
    }
  })
```

Classes conditionnelles

Classes conditionnelles

Nous pouvons binder des classes dynamiquement à un élément HTML. Cela s'écrit :

```
<div :class="{d-none:isOK}">Une div</div>
```

Ainsi, si isOK est égal à false, la classe "d-none" ne sera pas appliquée. Si isOK est égal à true, la classe "d-none" sera appliquée.

Je vous montre un exemple.

Classes conditionnelles

```
<div id="app">
  <p>Quels langages de programmation connaissez-vous ?</p>
  <input type="text" v-model="unLang">
  <input type="button" value="valider" @click="ajouter">

  <ul>
    <li v-for="langage in langages">
      <input type="checkbox" v-model="langage.etatCase"> <span
: class="{rouge:langage.etatCase}">{{ langage.lang }}</span>
    </li>
  </ul>
</div>
<script>
  app = Vue.createApp({
    data(){
```

Les conditions



Les conditions

Il nous est possible d'exprimer des conditions sur nos éléments HTML avec v-if, v-else-if et v-else.

Je vous montre.

Les conditions

Reprenons le code précédent. Si la case est cochée, on veut afficher “le texte HTML est coché”, sinon, “le texte HTML est décoché”.

Si le langage est Javascript et qu’il est coché,, on affichera en plus : “mon langage préféré !”

Voici l’input à modifier :

```
<input type="checkbox" v-model="langage.etatCase"> Le texte {{ langage.lang }}  
<span v-if="langage.etatCase && langage.lang == 'javascript'">est coché, mon  
langage préféré !</span>  
<span v-else-if="langage.etatCase">est coché</span>  
<span v-else>est décoché</span>
```


Fin de la première partie : découverte

A ce stade, vous savez :

Créer une application Vue.js 3

Définir des propriétés dans le modèle et y accéder dans la vue avec une `{{ interpolation }}`

Effectuer un binding bidirectionnel avec `v-model` et `v-bind`

Définir des `methods` au niveau de votre application

Utiliser `v-html` dans un élément pour l'html soit interprété :

`<p>En utilisant la directive `v-html` : </p>`

Gérer les events avec `@event`

Créer des boucles dans la vue avec `v-for`

Fin de la première partie : découverte

Utiliser des arguments dynamiques dans la vue avec `[unArgument]`.

Exemple : `@[eventDynamique] = "fonction"`

Créer des classes conditionnelles avec `:class="uneClasse:unBooleen"`

Fin de la première partie : TP 1

Il est temps d'appliquer tout ce que vous avez vu dans un projet de plus grande ampleur !

C'est l'heure du TP 1 !