

Formation Vue.js 3

Module de 3 jours (21 heures)

Apprendre à créer une application Web dynamique
et performante avec le framework Vue.js 3

Formateur : Jérémy Dumaye

Nous restons ouverts, nous proposons nos **formations à distance**

PRENEZ LE TEMPS D'APPRENDRE

Les formations recommandées par les développeurs pour les développeurs

NOS FORMATIONS



OBJECTIFS DE LA FORMATION

Organisation journée de formation :

- **9h** : début de la journée
- **11h – 11h15** : pause matin
- **12h30 – 14h** : pause déjeuner
- **16h – 16h15** : pause après-midi
- **17h30** : fin de la journée



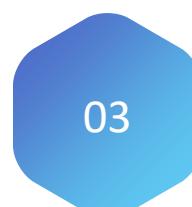
Une application web simple avec Vue.js 3

Introduction de la formation, présentation, préparation setup, bases techniques et premiers exercices avec Vue.js



Une application faite de composants

Rôle d'un composant, et de ses attributs, mise en place des composants, Vue CLI, et exercices



Routeur, plugins et API

Mise en place du routeur pour créer les routes, extensions plugins, extension API, et exercices

Projet fil rouge des 3 jours :

Création d'un annuaire de films permettant de lister les films, et pouvoir créer, modifier, et supprimer un film. Gestion également des catégories de chaque film.

Vue.js en quelques mots

Vue.js est un framework évolutif, permettant de créer des **applications web globales ou partielles**, et a été créé en février 2014 par Evan You, ancien ingénieur de Google ayant notamment travaillé avec Angular.js. Une véritable réussite puisque Vue.js est devenu très populaire rapidement.

C'est un **framework totalement open source**, et surtout totalement indépendant et entièrement piloté par sa communauté.

Vue.js **permet le data binding** (les données et le DOM sont couplés et réactifs aux changements). L'application **s'organise par composant** pour permettre de tout bien organiser.

Grâce à la puissance de sa gestion des routes, et à sa gestion des données via API, Vue.js permet de **créer facilement une Single Page Application (SPA)**. Son intérêt est d'éviter le chargement d'une nouvelle page à chaque action demandée, et de fluidifier ainsi l'expérience utilisateur. Les composants joueront un rôle essentiel pour l'affichage des différentes données liées aux différentes routes créées pour fluidifier l'expérience utilisateur et la vitesse de chargement.

Pour développer avec Vue.js, il va donc falloir avoir des bases en **JavaScript, et HTML/CSS**.

Une documentation très complète est disponible ici :

<https://v3.vuejs.org/guide/introduction.html>

Rappel rapide utilisation JavaScript

Avant de démarrer la formation Vue.js, petit point rapide sur les différentes syntaxes JavaScript à bien connaître et utiliser. Des syntaxes qui seront globalement utilisées durant notre formation Vue.js 3.

JavaScript : let & const

La déclaration d'une variable JavaScript ne se fera plus avec `var`, mais avec `let` et `const` durant notre formation. Deux déclarations permettant d'avoir un **stockage de données en local, et non global** comme `var`.

```
// On entre dans un bloc
if (true) {
|   let vueVariable = 'Vue.js';
}

console.log(vueVariable) // Erreur de syntaxe car vueVariable est indéfini
```

```
// on ne redéfinit pas une même variable
let secondVariable = 'Vue.js';
let secondVariable = 'Vue.js 3'; // Erreur de syntaxe car secondVariable déjà indéfini
```

```
const vueVariable = 'Vue.js';
vueVariable = 'Vue.js 3'; // Erreur de syntaxe car on ne peut pas assigner une autre valeur à une const
```

Rappel rapide utilisation JavaScript

JavaScript : Chaîne de caractères avec le signe accent grave `

Des variables ou opérations peuvent être interprétées directement dans une chaîne de caractères grâce au signe accent grave ` (Alt gr + 7). Pour utiliser des variables ou opérations à l'intérieur il faudra **utiliser des accolades** précédées du signe \$.

```
// Chaîne de caractères avec utilisation de variable
const maVariable = `Bonjour ${prenom} et bienvenue dans votre formation !`
```

JavaScript : Modules export import entre fichier JS

Des modules de scripts peuvent être exporté ou importé dans d'autres fichiers JavaScript.

```
// Export d'une variable présente dans le fichier base.js
export const name = 'Vue JS';

// Import de la variable du fichier base.js dans un fichier formation.js
import { name } from './base.js';
console.log(name) // La console affichera bien 'Vue JS'
```

Rappel rapide utilisation JavaScript

JavaScript : Utiliser les promesses avec then catch

Pour la gestion notamment des appels API, les promesses sont utilisées pour bien gérer les opérations asynchrones, en garantissant le déroulement dans un ordre voulu, et avec une gestion d'erreurs. Une fois l'appel à l'API réalisé, on récupère la réponse valide grâce à **then()** si tout est ok, ou sinon on récupère la réponse d'erreurs grâce à **catch()**.

```
// Un exemple d'appel API grâce à axios par exemple et utilisation du then et catch
axios
  .get('https://api.coindesk.com/v1/bpi/currentprice.json')
  .then(response => (this.info = response.data.bpi))
  .catch(error => console.log(error));
```

Rappel rapide utilisation JavaScript

JavaScript : Utilisation de Object.assign() et de ...

Régulièrement, nous avons besoin de créer une copie superficielle d'un objet, pour pouvoir réaliser une action précise. C'est là qu'intervient **Object.assign()** :

```
/* Exemple utilisation objet copié */
const originalObject = {
  nom: 'Dumaye',
  prenom: 'Jérémy',
  poste: 'Formateur Vue.js'
}
const copiedObject = Object.assign({}, originalObject)

originalObject.nom = 'Duje'

console.log(originalObject.nom) //Duje
console.log(copiedObject.nom) //Dumaye
```

Il est également possible d'utiliser la syntaxe **...** liée à un objet pour avoir le même rendu :

```
poste: 'Formateur Vue.js'
}
const copiedObject = {...originalObject}

originalObject.nom = 'Duje'
```

1- Une application web simple avec Vue.js 3

Setup de la formation Vue.js 3

Chaque point théorique de cette formation sera **mis en pratique** grâce à des exercices. Les exercices sont disponibles dans le **projet Github**. Un accès au projet sera donné pendant la formation (*pensez bien à donner votre identifiant Github*).

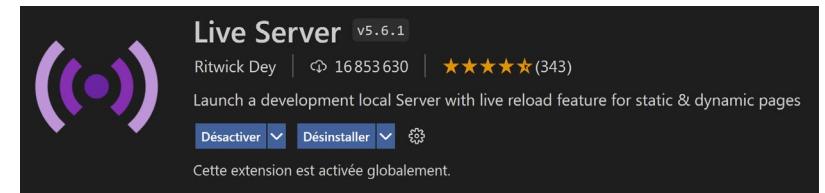
La solution de tous les exercices présentés sera également disponible dans une branche dédiée sur le projet Git partagé après chaque journée pour pouvoir revoir les points non compris.

Il faudra donc :

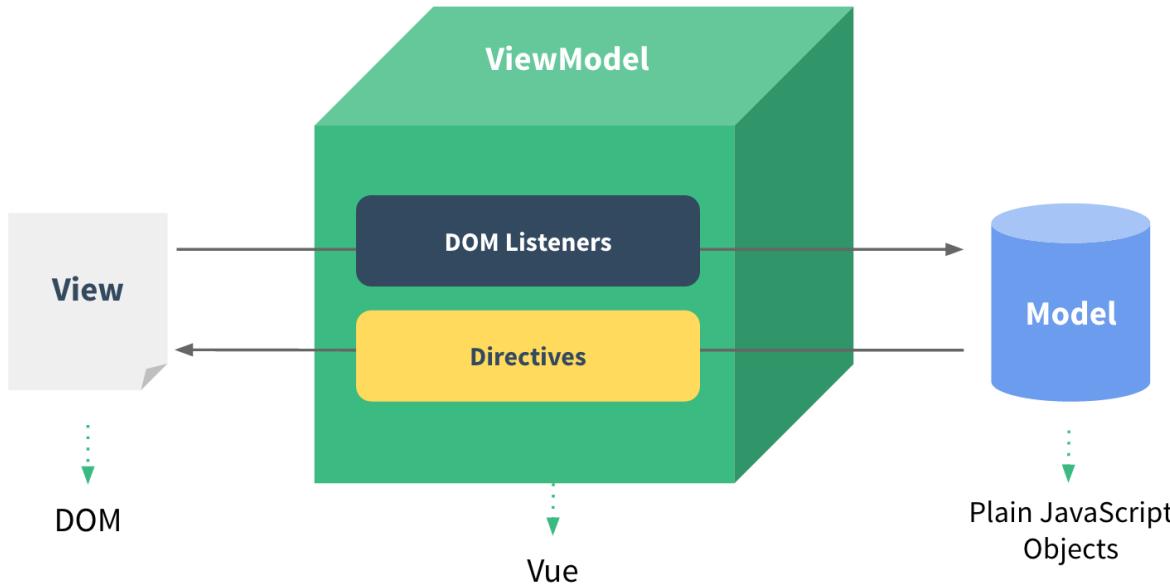
- Faire un fork du repository Github
- Faire un clone de votre fork
- Ajouter le repo comme remote supplémentaire pour récupérer les mises à jour quotidienne
- Chaque exercice aura un fichier readme pour pouvoir avoir le rappel des instructions.

Durant cette formation, l'éditeur **Visual Studio Code** sera utilisé.

Deux extensions Visual Studio Code seront nécessaires et à installer : l'extension **Vetur**, et l'extension **Live Server**.



Le modèle MVVM utilisé par Vue.js



Vue.js est un framework basé sur le **modèle MVVM**. Model View ViewModel, proche du modèle MVC dans sa composition.

Le but étant de séparer ce qui est affiché sur l'écran de l'interface utilisateur, de la logique programmatique. Il y a donc en premier le Model qui représente les données nécessaires à l'application, la View représente l'interface accessible à l'utilisateur et qu'il voit à l'écran, et le ViewModel fait l'interface entre le Model et la View en écoutant les actions de l'utilisateur pour le transmettre au Model, ou en récupérant les données du Model pour les transmettre à la View.

1- Une application web simple avec Vue.js 3

Mise en place de Vue.js 3

Nous allons donc mettre en place Vue.js dans un fichier index.html en nous basant sur l'implémentation expliquée dans la documentation officielle : <https://v3.vuejs.org/guide/introduction.html>

Objectif de cette mise en place :

- Ajouter le framework Vue.js 3 dans le fichier index.html grâce au CDN

```
<script src="https://unpkg.com/vue@3"></script>
```

Without Build Tools

To get started with Vue without a build step, simply copy the following code into an HTML file and open it in your browser:

```
<script src="https://unpkg.com/vue@3"></script>
```

- Créer notre première structure de code Vue.js
- Afficher sur la page notre premier « Hello World ! »

1- Une application web simple avec Vue.js 3

Mise en place de Vue.js 3

```
<body>
  <div id="app">
    <h2>{{ titre }}</h2>
  </div>

  <!-- Mise en place de Vue.js 3 via le CDN -->
  <script src="https://unpkg.com/vue@next"></script>

  <!-- Le code Vue.js -->
  <script>
    const app = Vue.createApp({
      data() {
        return {
          titre: 'Hello world'
        }
      },
      app.mount('#app')
    })
  </script>
</body>
```

- 1- On ajoute le framework Vue.js 3 via le CDN
 - 2- On crée une div d'ID #app qui servira d'afficher le rendu de notre application Vue.js
 - 3- On crée notre application, dans une variable const nommée app, grâce à la méthode `createApp` de Vue
 - 4- On définit nos premières datas
 - 5- On monte notre application pour pouvoir afficher ce que l'on souhaite dans la div #app grâce à `mount('#app')`
 - 6- À l'intérieur de la div #app représentant la balise parente de notre application, on ajoute les données à afficher provenant des datas de notre Model Vue.js
- Dans l'exemple ici on affiche la valeur de titre grâce aux doubles moustaches `{{ titre }}`

Le Model dans Vue.js 3

Le Model est l'ensemble des données fournies au ViewModel dans la méthode data().

```
data() {
  return {
    titre: 'Hello world'
  }
}
```

Ici, le Model comprend juste titre qui a comme valeur 'Hello world'. D'autres valeurs peuvent être ajoutées en dessous directement avec différents types possibles :

```
data() {
  return {
    titre: 'Hello world',
    texte: 'Ceci est mon premier texte',
    age: 42,
    clicBtn: false,
    personne: {
      prenom: 'Jérémie',
      nom: 'Dumaye'
    },
    competences: ['dev', 'formation']
  }
}
```

The diagram consists of five blue arrows pointing from specific data properties to their corresponding type labels. The properties are: 'titre', 'texte', 'age', 'prenom', and 'competences'. The labels are: '1- Type String', '2- Type Number', '3- Type Boolean', '4- Type Object', and '5- Type Array'.

- 1- Type String
- 2- Type Number
- 3- Type Boolean
- 4- Type Object
- 5- Type Array

La View dans Vue.js 3

L'affichage de la View dans Vue.js se situe donc à l'intérieur de la div montée grâce au ViewModel (ici dans notre div #app).

Les données du Model s'affichent grâce à la double accolade dans la View.

```
<div id="app">
|   <h2>{{ texte }}</h2>
</div>
```

Je demande à la View d'afficher ma variable texte définie en amont dans mon application Vue.js

À l'intérieur des doubles accolades, je peux réaliser du code JavaScript si besoin particulier, comme :

```
<div id="app">
|   <h2>{{ texte.toUpperCase() }}</h2>
</div>
```

J'utilise la méthode JavaScript toUpperCase() pour définir que mon texte s'affichera en majuscule

```
<div id="app">
|   <h2>{{ texteValide ? 'Texte valide !' : 'Texte non valide !' }}</h2>
</div>
```

Pour ma variable texteValide qui sera un booléen true ou false, j'utilise une condition ternaire pour afficher un texte si c'est true ou un autre si c'est false

Bien entendu, nous verrons par la suite qu'il est souvent plus propre de passer par d'autres propriétés Vue pour nous éviter d'ajouter trop de code JavaScript dans la View

Le ViewModel dans Vue.js 3

Le ViewModel est la base de la construction de l'application, qui fera la passerelle entre le Model et la View.

```
const app = Vue.createApp({
  data() {
    return {
      titre: 'Hello world'
    }
  },
  app.mount('#app');
```

Le ViewModel est donc la création de la structure de l'application pour pouvoir avoir des interactions avec le Model et la View. Ici, de base on a donc la propriété data() qui permet de récupérer les datas provenant du Model.

Le ViewModel a également plein d'autres propriétés possibles, les plus utilisées étant les propriétés methods, et computed. Il y en a également d'autres comme watch, mounted, etc. qu'on aura le temps de voir plus tard.

```
const app = Vue.createApp({
  data() {
    return {
      age: 42,
      count: 4
    }
  },
  methods: {
    increment() {
      this.count++
    }
  },
  computed: {
    ageLegal() {
      return this.age > 17 ? 'Authorisé' : 'Non autorisé'
    }
  }
});
app.mount('#app');
```

Pour définir la propriété methods, on crée un objet avec à l'intérieur autant de fonctions nécessaires à une action. Une méthode peut avoir des paramètres en appel.

Pour appeler une méthode dans la View, il suffit de faire {{ increment() }} par exemple.

La propriété computed sera pour créer des fonctions calculées. Pour afficher une propriété computed dans la View, il suffit de faire {{ ageLegal }} Un computed est en get de base, mais peut aussi avoir un setter que nous verrons plus tard.

Exercice numéro 1

01

Pour ce premier exercice, le but sera de mettre en place Vue.js 3 via le CDN, de monter notre application dans la variable const app et d'afficher à l'écran la date et l'heure actuelle.

Astuce : l'affichage de la date et de l'heure se fera grâce à une instance de l'objet JavaScript Date()

Le data-binding dans Vue.js

Le data binding permet d'interagir avec les données du Model vers la View ou depuis la View grâce au ViewModel.

Il existe le **One Way Binding** et le **Two Ways Binding**.

Le One Way Binding pour passer une valeur du Model pour un rendu dans la View :

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

  <!-- Mise en place de Vue.js 3 via le CDN --&gt;
  &lt;script src="https://unpkg.com/vue@next"&gt;&lt;/script&gt;

  <!-- Le code Vue.js --&gt;
  &lt;script&gt;
    const app = Vue.createApp({
      data() {
        return {
          logoVueUrl: 'https://v3.vuejs.org/logo.png'
        }
      },
      app.mount('#app');
    &lt;/script&gt;
&lt;/body&gt;</pre>
```

*Le Model contient la data de l'url du logo de Vue.js à ajouter. Pour pouvoir définir un attribut HTML dynamiquement (ici donc l'attribut src de la balise img), on ajoute la propriété **v-bind** devant. Ce qui donne **v-bind:src***

A noter :

- Il ne faut pas mettre les doubles accolades pour afficher la variable **logoVueUrl**
- Un affichage raccourci est possible en enlevant le **v-bind** du **v-bind:src** et en ne laissant donc que le **:src**

Le data-binding dans Vue.js

Le Two Ways Binding est le fait d'utiliser une variable du Model en la modifiant et en affichant sa nouvelle valeur:

```
<body>
  <div id="app">
    <input type="text" v-model="texte">

    <p>Le texte tapé est : {{ texte }}</p>
  </div>

  <!-- Mise en place de Vue.js 3 via le CDN -->
  <script src="https://unpkg.com/vue@next"></script>

  <!-- Le code Vue.js -->
  <script>
    const app = Vue.createApp({
      data() {
        return {
          texte: ''
        }
      };
      app.mount('#app');
    </script>
</body>
```

Le Model contient la data du texte. Ce texte va être mis à jour lorsque l'utilisateur remplira l'input dans la page. Le Model sera alors modifié avec la nouvelle valeur. Le Two Ways Binding est le fait d'enregistrer la valeur, mais aussi d'afficher cette nouvelle valeur en live dans la page.

*Pour pouvoir réaliser cela, on ajoute la propriété **v-model** en attribut de la balise HTML avec en valeur la variable à modifier. La modification peut être ensuite affichée en live dans la page via cette même variable.*

*Le **v-model** aura un paramétrage un peu différent lorsqu'il sera utilisé avec des composants. Notions qui seront vues plus tard dans la formation.*

Le data-binding dans Vue.js

Au niveau des formulaires, le data-binding s'utilise comme cela :

Pour les input type checkbox :

```
<body>
  <div id="app">
    <p>Listez les langages que vous connaissez :</p>
    <input id="html" type="checkbox" v-model="valuesCheckbox" value="HTML">
    <label for="html">HTML</label>

    <input id="css" type="checkbox" v-model="valuesCheckbox" value="CSS">
    <label for="css">CSS</label>

    <input id="js" type="checkbox" v-model="valuesCheckbox" value="JavaScript">
    <label for="js">JavaScript</label>

    <p>Vous connaissez les langages suivants : {{ valuesCheckbox.join(', ') }}</p>
  </div>

  <script src="https://unpkg.com/vue@next"></script>

  <script>
    const app = Vue.createApp({
      data() {
        return {
          valuesCheckbox: []
        }
      },
      app.mount('#app')
    })
  </script>
</body>
```

Pour pouvoir lister et retenir les valeurs cochées pour les inputs checkbox, il est nécessaire d'avoir un valeur sous forme de tableau provenant du Model

Bien entendu, l'ensemble de ces checkbox ont le même thème et traitent du même sujet, donc le v-model devra être le même. Il suffira d'utiliser l'attribut HTML value pour définir la valeur à retenir ensuite dans le Model

1- Une application web simple avec Vue.js 3

Le data-binding dans Vue.js

Au niveau des formulaires, le data-binding s'utilise comme cela :

Pour les input type radio :

```
<body>
  <div id="app">
    <p>Votre niveau :</p>
    <input id="debutant" type="radio" v-model="valueRadio" value="Débutant">
    <label for="debutant">Débutant</label>

    <input id="intermediaire" type="radio" v-model="valueRadio" value="Intermédiaire">
    <label for="intermediaire">intermédiaire</label>

    <input id="expert" type="radio" v-model="valueRadio" value="Expert">
    <label for="expert">Expert</label>

    <p>Votre niveau est : {{ valueRadio }}</p>
  </div>

  <script src="https://unpkg.com/vue@next"></script>

  <script>
    const app = Vue.createApp({
      data() {
        return {
          valueRadio: ''
        }
      },
      app.mount('#app');
    </script>
</body>
```

Presque pareil donc pour les inputs radio, sauf que cette fois-ci, vu que un bouton radio n'autorise qu'un choix unique, le modèle ne doit pas être un tableau de données, mais une chaîne de caractères.

Comme pour l'input checkbox, les input radio étant de même thème et traitant le même sujet, il faudra bien mettre le même v-model pour retenir la valeur donnée dans le Model.

Exercice numéro 2

02

Créer une nouvelle application Vue.js avec :

- Une **balise select et 3 options** permettant de choisir une compétence web préférée
- Un **textarea** permettant d'entrer un texte pour expliquer pourquoi on aime ce langage
- Une **div** en dessous permettant d'afficher le langage préféré choisi, et le texte d'explication.

Une fois l'application fonctionnelle, il faudra trouver la propriété CSS nécessaire à ajouter qui permettra d'afficher le texte **en prenant en compte les retours à la ligne** effectués dans le textarea.

Les conditions avec Vue.js

Les conditions if, else, else if, sont bien entendus possibles pour construire sa View. Il suffira d'utiliser les propriétés **v-if**, **v-else-if**, ou **v-else** en attribut des balises HTML.

```
<body>
  <div id="app">
    <p v-if="valueOneSubmit">Valeur 1 envoyée</p>
    <p v-else>Valeur 1 non envoyée</p>

    <p v-if="valueTwoSubmit">Valeur 2 envoyée</p>
    <p v-else>Valeur 2 non envoyée</p>

    <p v-if="result > 50">Le résultat est supérieur à 50</p>
    <p v-else-if="result > 40">Le résultat est supérieur à 40</p>
    <p v-else-if="result > 30">Le résultat est supérieur à 30</p>
    <p v-else-if="result > 20">Le résultat est supérieur à 20</p>
    <p v-else>Le résultat est inférieur ou égal à 20</p>
  </div>

  <script src="https://unpkg.com/vue@next"></script>

  <script>
    const app = Vue.createApp({
      data() {
        return {
          valueOneSubmit: true,
          valueTwoSubmit: false,
          result: 20
        }
      },
      app.mount('#app');
    </script>
</body>
```

Nous avons placé des conditions pour définir si la valeur 1 a été envoyée, ou la valeur 2. Si true, on passe dans le **v-if**, sinon on passe dans le **v-else**.

Pareil pour le résultat, avec un exemple permettant de travailler avec le **v-if**, **v-else-if**, et **v-else**.

A noter que les balises HTML qui ne répondent pas aux conditions demandées ne seront tout simplement pas rendus dans le DOM.

Si l'on souhaite afficher ou ne pas afficher une balise HTML en fonction d'une valeur, mais garder la balise dans le DOM, on utilise **v-show** qui ajoutera un simple style `inline display:none` si la condition renvoi `false`.

Utilisation de v-text ou v-html

Pour afficher des données dans la View, nous avons vu que cela se faisait grâce à la double accolade `{{ maVariable }}`. Nous pouvons également utiliser une propriété `v-text` ou `v-html` qui s'ajoute dans les attributs d'une balise directement. L'avantage est de pouvoir ajouter du code HTML directement dans le View par exemple, ce qui n'est pas possible avec des doubles accolades simples. Comme son nom l'indique, `v-text` sera pour afficher du texte brut dans le DOM, et `v-html` affichera du texte HTML.

```
data() {
  return {
    texteBrut: 'Ceci est un texte brut',
    texteHtml: '<p>Ceci est un <strong>texte HTML</strong></p>',
  }
}
```

Le Model ici a deux variables de texte, dont une avec des balises HTML à l'intérieur

```
<div v-text="texteBrut"></div>
<div v-html="texteBrut"></div>

<div v-text="texteHtml"></div>
<div v-html="texteHtml"></div>
```

Ceci est un texte brut
Ceci est un texte brut

<p>Ceci est un texte HTML</p>

Ceci est un **texte HTML**

Nous voyons donc bien la différence, notamment pour la variable `texteHtml` qui via `v-text` reste en texte brut, et via `v-html` s'affiche bien en HTML et transforme les balises dans le DOM.

Les événements v-on en Vue.js

Autre point très utilisé dans Vue.js : les événements **v-on**. Le plus connu étant le **v-on:click** permettant d'ajouter une action sur le clic d'un élément HTML précis, sur un bouton par exemple. Vue.js a également une syntaxe raccourcie pour cette propriété, nous pouvons soit utiliser le **v-on:** ou alors le **@**, ce qui donne soit **v-on:click** ou **@click** pour l'exemple d'un événement au clic.

```
<body>
  <div id="app">
    <p>Formation Vue.js</p>
    <button @click="produit++">Ajouter au panier</button>

    <p v-if="produit > 0">Nombre de formation Vue.js au panier : {{ produit }}</p>
  </div>

  <script src="https://unpkg.com/vue@next"></script>

  <script>
    const app = Vue.createApp({
      data() {
        return {
          produit: 0
        }
      });

      app.mount('#app');
    </script>
  </body>
```

*Le Model a une variable pour le nombre de produit ajouté au panier. L'événement **@click** a été défini et permet d'ajouter +1 au produit à chaque clic.*

La phrase indiquant le nombre de produit au panier s'affiche dès qu'il y a un produit.

Les événements v-on en Vue.js

Chaque événement Vue.js peut avoir des attributs supplémentaires permettant d'être encore plus précis dans le type d'événement que l'on souhaite mettre en place. Voici les plus utilisés :

```
<!-- Evénement déclenché au clic une seule fois -->
<a href="" @click.once="actionClic()">...</a>
<!-- Evénement n'ira pas vers le href prévu -->
<a href="" @click.prevent="actionClic()">...</a>
<!-- Propagation de l'événement arrêtée -->
<a href="" @click.stop="actionClic()">...</a>
<!-- On peut également enchaîner les événements -->
<a href="" @click.prevent.stop="actionClic()">...</a>
```

Il existe aussi d'autres événements comme `@keydown`, événement quand une touche est appuyée, `@keyup` quand une touche est relâchée, `@keypress` quand elle est appuyée puis relâchée, etc.

Ces événements ont également des attributs supplémentaires, comme par exemple :

```
<!-- Evénement déclenché à la touche entrée relâchée -->
<input type="text" @keyup.enter="actionClic()">
<!-- Evénement déclenché quand alt + entrée relâchée -->
<input type="text" @keyup.alt.enter="actionClic()">
<!-- Evénement déclenché au clic gauche de la souris -->
<a href="" @click.left="actionClic()">...</a>
<!-- Evénement déclenché au clic droit de la souris -->
<a href="" @click.right="actionClic()">...</a>
<!-- Evénement déclenché au clic milieu de la souris -->
<a href="" @click.middle="actionClic()">...</a>
```

D'autres disponibles dans la documentation en ligne : <https://v3.vuejs.org/guide/events.html#event-modifiers>

Utilisation de computed dans Vue.js

Nous avons vu tout à l'heure dans l'exercice 1 qu'il était possible d'avoir des propriétés calculées dans Vue grâce à computed. Voici plus de précisions.

```
<div id="app">
  <h2>Il y a {{ countUsers }} inscrits au programme !</h2>
</div>

<!-- Mise en place de Vue.js 3 via le CDN --&gt;
&lt;script src="https://unpkg.com/vue@next"&gt;&lt;/script&gt;

<!-- Le code Vue.js --&gt;
&lt;script&gt;
  const app = Vue.createApp({
    data() {
      return {
        users: ['user 1', 'user 2', 'user 3', 'user 4']
      }
    },
    computed: {
      countUsers() {
        return this.users.length;
      }
    }
  );
  app.mount('#app');
&lt;/script&gt;</pre>
```

*L'avantage d'utiliser des propriétés calculées grâce à **computed** c'est au niveau du recalcule de la data. En effet, une fonction **computed** sera en cache et n'aura pas besoin d'être recalculée à chaque fois, sauf quand une des données du model utilisées dans la fonction **computed** change, cela bascule le recalcule de la fonction **computed**.*

*Ici donc, la propriété permettant de calculer le nombre d'utilisateurs ne sera jamais recalculée si le tableau des **users** ne change pas.*

*De son côté, une méthode **methods** Vue.js sera ré-exécutée à chaque fois malgré qu'elles puissent donner le même résultat.*

A bien choisir donc.

1- Une application web simple avec Vue.js 3

Utilisation des methods Vue.js

Nous avons vu tout à l'heure dans l'exercice 1 qu'il était possible d'avoir des propriétés calculées dans Vue grâce à computed. Nous allons maintenant voir la propriété methods de Vue permettant de réalisés des actions précises.

```
<body>
  <div id="app">
    <label for="prenom">Entrez votre prénom :</label>
    <input type="text" id="prenom" v-model="prenom">
    <button @click="affichageBienvenue()">Valider</button>
  </div>

  <!-- Mise en place de Vue.js 3 via le CDN -->
  <script src="https://unpkg.com/vue@next"></script>

  <!-- Le code Vue.js -->
  <script>
    const app = Vue.createApp({
      data() {
        return {
          prenom: ''
        }
      },
      methods: {
        affichageBienvenue() {
          if (this.prenom) {
            alert(`Bonjour et bienvenue ${this.prenom} !`);
          } else {
            alert(`Merci de rentrer votre prénom !`);
          }
        }
      });
    app.mount('#app');
  </script>
</body>
```

*Le Model a en mémoire la donnée **prenom** qui permettra de lier la valeur entrée dans l'input.*

*La méthode **affichageBienvenue()** va faire un **alert()** pour dire la bienvenue au prénom qui aura été renseigné avant par l'utilisateur et déclenché grâce à l'événement **@click**.*

*Nous avons accès à toutes les données du Model depuis chaque méthode grâce à la propriété **this**.*

*Une simple condition a été ajoutée dans la méthode pour s'assurer que **this.prenom** soit bien rempli.*

Exercice numéro 3

03

Créer une nouvelle application Vue.js avec :

- Un titre « Traduction du mot Bienvenue »
- Trois boutons permettant au clic de traduire le mot Bienvenue en anglais, espagnol, et italien.

Astuce : il faudra utiliser une méthode avec un argument.

Exercice numéro 4

04

Créer une nouvelle application Vue.js avec :

- Une image présentant une photo au choix et un attribut width sur l'image avec une valeur défini par le Model
- Une méthode permettant au clic avec le bouton de la souris gauche sur l'image d'ajouter 100 en largeur, au clic droit de la souris sur l'image d'enlever 100, et au clic avec le bouton du milieu de la souris sur l'image, remettre à la valeur d'origine

Style et class binding dans Vue.js

Le data binding fonctionne également avec les class HTML ou les style inline HTML. Comme vu tout à l'heure, l'attribut **v-bind:** ou simplement **:** est à utiliser sur **v-bind:class**, ou **:class**, et **v-bind:style**, ou **:style**.

```
<body>
  <div id="app">
    <div :class="{'active-link': isActive, 'first-bloc': isFirst}"></div>
    <div :style="{'backgroundColor': redColor, 'color': whiteColor}">Test texte</div>
  </div>

  <script src="https://unpkg.com/vue@next"></script>

  <script>
    const app = Vue.createApp({
      data() {
        return {
          isActive: true,
          isFirst: true,
          redColor: '#c02626',
          whiteColor: '#fff'
        }
      },
      app.mount('#app')
    })
  </script>
</body>
```

*Le style et class binding se comporte de la même manière. Un tableau d'objet est ouvert avec en première propriété le nom de la class ou du style CSS, et en valeur une data du Model qui renverra donc **true** ou **false** pour afficher ou non la classe, et renverra une valeur CSS pour remplir la propriété du style CSS choisie.*

Les boucles v-for dans Vue.js

Pour parcourir des tableaux de données, que ce soit de simple tableaux, ou des tableaux d'objet, la propriété **v-for** est utilisée pour réaliser une boucle afin de récupérer ces données et de les lister.

```
<body>
  <div id="app">
    <!-- Listing d'une tableau de données --&gt;
    &lt;ul&gt;
      | &lt;li v-for="langage in langagesWeb"&gt;{{ langage }}&lt;/li&gt;
    &lt;/ul&gt;
    <!-- Listing d'une tableau de données avec l'index --&gt;
    &lt;ul&gt;
      | &lt;li v-for="(langage, index) in langagesWeb"&gt;{{ index + 1 }} - {{ langage }}&lt;/li&gt;
    &lt;/ul&gt;
  &lt;/div&gt;

  &lt;script src="https://unpkg.com/vue@next"&gt;&lt;/script&gt;

  &lt;script&gt;
    const app = Vue.createApp({
      data() {
        return {
          langagesWeb: ['PHP', 'HTML', 'CSS', 'Vue.js']
        }
      };
      app.mount('#app');
    &lt;/script&gt;
&lt;/body&gt;</pre>
```

*Pour parcourir les éléments d'un tableau de données présent dans le Model, il faut donc utiliser **v-for** directement sur l'élément qui sera répété, ici l'élément ****.*

*La syntaxe est **v-for='(data, index)** in **datas**' où **datas** est le nom de la valeur provenant du Model. L'utilisation de la propriété **index** permet de parcourir les index du tableau. Attention, un index commence toujours à 0.*

Information : pour le moment nous n'avons pas besoin de l'attribut **key** avec le **v-for**, mais il sera obligatoire dans une structure autre que l'implémentation de Vue.js avec le CDN

Exercice numéro 5

05

Créer une nouvelle application Vue.js avec :

- Un champ label et input demandant « Quels langages de programmation connaissez-vous ? »
- Un bouton permettant au clic d'ajouter la valeur entrée de le champ de l'input dans un tableau de données, en faisant en sorte d'afficher un message si la valeur est vide
- Avoir un listing en live en dessous qui affiche la liste des langages choisis

Exercice numéro 6

06

Reprendre l'exercice 5 et :

Ajouter un bouton sur chaque élément de la liste créée en fonction des résultats entrés pour pouvoir supprimer une réponse

1- Une application web simple avec Vue.js 3

Les boucles v-for dans Vue.js

Exemple pour parcourir un tableau d'objet avec **v-for** :

```
<body>
  <div id="app">
    <ul>
      <li v-for="personne in personnes">{{ personne.prenom }} {{ personne.nom }}</li>
    </ul>
  </div>

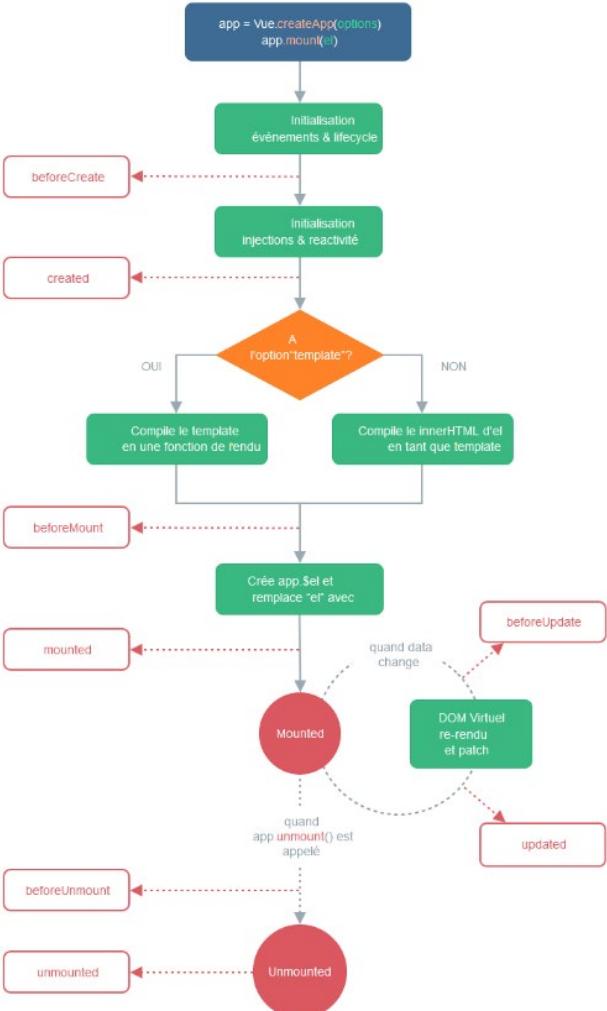
  <script src="https://unpkg.com/vue@next"></script>

  <script>
    const app = Vue.createApp({
      data() {
        return {
          personnes: [
            {
              prenom: 'Jérémie',
              nom: 'Dumaye'
            },
            {
              prenom: 'Evan',
              nom: 'You'
            }
          ]
        };
      };
      app.mount('#app');
    </script>
</body>
```

*Une fois les données parcouru avec **v-for**, on peut appeler les données de chaque objet grâce à **data.name***

1- Une application web simple avec Vue.js 3

Les différents hooks cycle de vie Vue.js



Un aperçu de tout le diagramme de cycle de vie à gauche, et leur utilisation en bas en exemple. Il n'est pas nécessaire de tout apprendre, mais en fonction des différents cas, cela pourra être une référence utile.

```
const app = Vue.createApp({  
  data() {  
    return {  
      message: 'Hello World !'  
    }  
  },  
  beforeCreate: function () {  
    console.log("Avant ma création")  
  },  
  created: function () {  
    console.log("Une fois créé")  
  },  
  beforeMount: function () {  
    console.log("Avant d'être monté")  
  },  
  mounted: function () {  
    console.log("Une fois monté")  
  },
```

Exercice projet fil rouge

PR

Projet fil rouge : Mises en pratique avec un annuaire de films :

Objectifs du jour pour le projet fil rouge :

- Pouvoir lister les éléments d'un catalogue de films disponibles indiquant pour l'instant le nom du film, son année de sortie, son réalisateur, et sa catégorie
- Pouvoir ajouter un autre film dans la page, et supprimer ou éditer les éléments de la liste
- Lors d'un ajout de film dans la liste, pouvoir sélectionner une catégorie via un listing déjà imaginé de catégories disponibles