

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 Build Tools, 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.

2- Une application faite de composants

Création de composants Vue.js

Quand une application devient trop importante, il est important de la diviser en **plusieurs composants** qui s'appellent de façon hiérarchique. Les composants peuvent être définis de façon **globale ou locale**. Un composant se compose de la même façon qu'un ViewModel, c'est-à-dire qu'il peut avoir un Model avec data(), des methods, computed, un template, et aussi l'appel à d'autres composants via components.

Composant global :

Ces composants sont enregistrés globalement pour l'application. Cela signifie qu'ils peuvent être utilisés dans le template de n'importe quelle instance de composant dans cette application.

```
<body>
  <div id="app">
    <hello-world></hello-world>
  </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({});

    app.component('hello-world', {
      template: '<h2>Hello World !</h2>'
    })
    app.mount('#app');
  </script>
</body>
```

*Nous venons de créer globalement le composant nommé « **hello-world** ».*

*A noter qu'il faut s'assurer d'avoir bien ajouté son **app.component()** avant le **app.mount()** puisqu'il faut enregistrer le composant avant de monter l'application Vue.*

*Une fois le composant créé, il peut être utilisé comme une balise HTML en réutilisant son nom, ici donc <**hello-world**>*

Création de composants Vue.js

Composant local :

Les problèmes des composants définis de façon globale c'est qu'ils sont toujours inclus dans l'application, même s'ils ne sont pas utilisés, ce qui augmente inutilement les sections JavaScript à télécharger par l'utilisateur. Pour résoudre ce problème, on utilise donc les composants localement la plupart du temps. Nous devons ajouter la propriété `components` à chaque composant Vue pour définir quels composants pourront être utilisés en tant qu'enfant.

```
<body>
  <div id="app">
    <hello-world></hello-world>
  </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 HelloWorld = {
      template: '<h2>Hello World !</h2>'
    }
    const app = Vue.createApp({
      components: {
        'hello-world': HelloWorld
      }
    })
    app.mount('#app');
  </script>
</body>
```

Nous venons de créer de façon locale le composant nommé « `hello-world` ».

Comparé à la façon globale, nous créons une variable `const HelloWorld` (un composant doit toujours commencer par une majuscule) avec le `template` à l'intérieur.

Nous ajoutons ensuite le composant dans le ViewModel via la propriété `components` avec en objet, le nom du composant en paramètre et la variable `const` en valeur.



2- Une application faite de composants

Exercice numéro 1

01

Mettre en place une application Vue.js 3 et créer un composant Compteur permettant de présenter un bouton dont sa valeur sera le nombre de clics et qui s'incrémentera de 1 à chaque clic.

[Créer une version globale et locale de ce composant.](#)

2- Une application faite de composants

Utilisation des props pour les composants

Les props sont des attributs personnalisés que vous pouvez enregistrer sur un composant. Il y a transmission de données aux composants enfants depuis leur parent.

Exemple :

```
<body>
  <div id="app">
    <hello prenom="Jérémie"></hello>
  </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 Hello = {
      template: '<h2>{{ prenom }}</h2>',
      props: ['prenom']
    }
    const app = Vue.createApp({
      components: {
        'hello': Hello
      }
    });

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

*L'exemple est sur un composant local à qui on a ajouté un attribut **prenom**. Ce prénom ici contient une valeur en dure qui est **Jérémie** mais aurait très bien pu contenir une valeur dynamique provenant des datas du Model du parent.*

*Dans le template du composant, nous avons directement utilisé le nom donné en propriété, ici **prenom**. Pour qu'il soit reconnu comme propriété utilisable par le composant enfant, il faut ajouter la propriété **props** qui est un tableau de valeur représentant les données envoyées depuis le parent. On a donc ajouté ici **prenom** pour pouvoir l'utiliser.*



2- Une application faite de composants

Exercice numéro 2

02

Mettre en place une application Vue.js 3 et créer un composant local appelé TexteMaj.

Le but étant de :

- Envoyer une props texte à l'enfant provenant d'une data du Model du parent
- Ajouter une méthode computed dans le composant TexteMaj permettant de récupérer ce texte et de l'afficher en majuscule.

2- Une application faite de composants

Écoute événements sur composant enfant

Dans le contraire de la **props** qui permettait au parent d'envoyer des données au composant enfant, le composant enfant peut émettre un événement sur lui-même en appelant la méthode intégrée **\$emit**, en passant le nom de l'événement.

Exemple :

```
<div id="app">
  <p>Total des incrémentation : {{ total }}</p>
  <compteur @incrementation="plusUnTotal()"></compteur>
</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 Compteur = {
    data() {
      return {
        nbClics: 0
      }
    },
    methods: {
      ajouterClic() {
        this.nbClics++;
        this.$emit('incrementation');
      }
    },
    template: '&lt;button @click="ajouterClic()"&gt;{{ nbClics }}&lt;/button&gt;'
  }
  const app = Vue.createApp({
    data() {
      return {
        total: 0
      }
    },
    methods: {
      plusUnTotal() {
        this.total++;
      }
    },
    components: {
      'compteur': Compteur
    }
  });
</pre>
```

L'exemple ici était de réutiliser le compteur de clic dans le composant enfant et d'envoyer son incrémentation au parent pour qu'il puisse faire un calcul du total.

*Pour réaliser cette écoute, nous avons ajouté un **this.\$emit** en lui donnant un nom. Le nom donné a été **incrementation**, et c'est celui-ci qui a été ajouté dans l'attribut du composant via la propriété **@name**, donc **@incrementation**. À chaque action, donc **\$emit**, on a un déclenchement de cette propriété et on s'en sert pour appeler la méthode **plusUnTotal()** qui incrémente la propriété **total** du Model.*

2- Une application faite de composants

Écoute événements sur composant enfant

Il est également possible d'ajouter une valeur précise dans un événement `$emit` pour le transmettre au parent.

Exemple :

```
<body>
  <div id="app">
    <p :style="{'fontSize': `${texteSize}em`}">Texte à agrandir</p>
    <increase @increase-texte="texteSize += $event"></increase>
  </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 Increase = {
      methods: {
        agrandirTexte() {
          this.$emit('increase-texte', 0.2);
        }
      },
      template: '<button @click="agrandirTexte()">Agrandir le texte</button>'
    }
    const app = Vue.createApp({
      data() {
        return {
          texteSize: 1
        }
      },
      components: {
        'increase': Increase
      }
    });

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

Le but était d'utiliser le composant `increase` pour qu'au clic la méthode `agrandirTexte()` envoie l'événement `increase-texte` avec cette fois-ci une valeur précise.

Pour récupérer cette valeur, on prend le nom de l'événement sur le composant `@increase-texte` et on attribut la valeur que l'on souhaite en reprenant la valeur envoyée grâce à `$event`. Ici on augmente de `0.2` donc à chaque clic sur le bouton.

2- Une application faite de composants

Utilisation des slots dans les composants

Il est souvent utile de pouvoir transmettre du contenu à un composant grâce à l'élément personnalisé `<slot>` de Vue. Il existe des slots simples, nommés, et scopés.

Exemple slots simples :

```
<body>
  <div id="app">
    <utilisation-slot></utilisation-slot>
    <utilisation-slot>Un texte de remplacement</utilisation-slot>
  </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 UtilisationSlot = {
      template: '<div><slot>Texte par défaut</slot></div>'
    }
    const app = Vue.createApp({
      components: {
        'utilisation-slot': UtilisationSlot
      }
    });

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

*Nous avons créé un composant simple nommé **utilisation-slot** pour tester les différents types de slot.*

*Dans le template du composant, nous devons utiliser **<slot>** pour utiliser un slot simple avec entre ses balises un texte par défaut. Cela permettra de dire au composant d'afficher un texte s'il n'y a pas de texte présent à l'intérieur.*

2- Une application faite de composants

Utilisation des slots dans les composants

Exemple slots nommés :

```
<div>
  <div id="app">
    <utilisation-slot>
      <template v-slot:header-texte>
        <p>Ceci est le texte du haut dans le composant</p>
      </template>
      <template v-slot:default>
        <p>Ceci est le texte de base</p>
      </template>
      <template v-slot:footer-texte>
        <p>Ceci est le texte du bas dans le composant</p>
      </template>
    </utilisation-slot>
  </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 UtilisationSlot = {
      template: `<div>
        <header><slot name="header-texte"></slot></header>
        <p><slot></slot></p>
        <footer><slot name="footer-texte"></slot></footer>
      </div>`
    }
    const app = Vue.createApp({
      components: {
        'utilisation-slot': UtilisationSlot
      }
    });

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

Nous avons créé un composant simple nommé **utilisation-slot** pour tester les différents types de slot.

Entre les balises HTML du composant, nous devons utiliser la balise **<template>** avec un **v-slot:name** pour définir le slot nommé à utiliser. Et dans le **template** View Model du composant, on ajoute la balise **<slot>** en lui ajoutant un attribut **name=** qui sera égal au **v-slot** défini. Le **v-slot:default** équivaut au **<slot>** sans **name=**

L'avantage des slots nommés sera pour définir les différents emplacements dans un composant avec des données à lui envoyer dynamiquement.

Il y a une syntaxe raccourcie pour **v-slot:** qui est #

2- Une application faite de composants

Utilisation des slots dans les composants

Exemple slots scopés, avec une portée :

```
<body>
  <div id="app">
    <utilisation-slot>
      <template v-slot:default="slotProps">
        <span>Item {{ slotProps.index + 1 }}<br/>
          est : {{ slotProps.item }}</span>
      </template>
    </utilisation-slot>
  </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 UtilisationSlot = {
      data() {
        return {
          items: ['Ordinateur', 'Sacoche', 'Ecran']
        }
      },
      template: `<ul>
        <li v-for="(item, index) in items">
          <slot :item="item" :index="index"></slot>
        </li>
      </ul>`
    }
    const app = Vue.createApp({
      components: {
        'utilisation-slot': UtilisationSlot
      }
    })
    app.mount('#app');
  </script>
</body>
```

Nous avons créé un composant simple nommé **utilisation-slot** pour tester les différents types de slot.

Parfois, il est utile que le contenu du slot ait accès aux données disponibles uniquement dans le composant enfant. Ici pour l'exemple c'est un tableau d'item qui sera donné dans le slot et qui sera uniquement disponible pour récupérer. Les attributs liés à un élément **<slot>** sont appelés props de slot, d'où le fait que l'on utilise **v-slot:default="slotProps"** et où **slotProps** est utilisée ensuite pour récupérer les données fournies par le composant enfant. On aurait pu donner n'importe quelle autre identifiant que **slotProps** bien entendu.

Il y a une syntaxe raccourcie pour **v-slot:** qui est #

Exercice numéro 3

03

Mettre en place une application Vue.js 3 et créer un composant local appelé CardIdentity.

Le but étant de :

- Créer un slot permettant d'avoir une entête de la carte nommée header et qui permettra de récupérer le nom et prénom de la personne
- Ajouter un slot permettant d'avoir un footer et qui sera nommé footer, et qui permettra de récupérer le téléphone et le mail de la personne
- Ajouter un style css simple seulement pour entourer la carte et ajouter une séparation entre le header et le footer
- Les données dans le composant CardIdentity seront un tableau d'objet avec deux exemples, et avec en informations nom, prénom, téléphone, et mail.

2- Une application faite de composants

Utilisation du setup officiel de Vue.js

Après avoir compris la philosophie du framework Vue.js grâce au CDN et à l'affichage dans un fichier HTML simple, nous allons à présent passer au setup officiel de Vue.js, qui est une méthode beaucoup mieux structurée, digne de ce framework et utilisée pour de plus gros projet d'application web. Une approche composant avec chaque composant Vue.js défini dans un fichier séparé. Il sera intéressant d'utiliser Babel pour réaliser le tout en ES6. Le module bundler Webpack sera utilisé pour organiser les différents fichiers utilisés, organisé en module. C'est parti pour créer la première application avec le setup officiel.

Installation du projet Vue.js via le setup officiel :

Si vous n'avez pas encore installé sur votre machine Vue CLI, il est temps de le faire directement via le terminal en exécutant : `> npm init vue@latest`

Il suffira ensuite d'aller dans le dossier au nom du projet choisi et de faire :

```
> npm install
```

Puis lancer son projet Vue.js via :

```
> npm run dev
```

Quand l'application sera prête à être déployée, il suffira de faire :

```
> npm run build
```



2- Une application faite de composants

Architecture de l'application

Une fois le projet installé et monté, faisons un tour de l'architecture mise en place et des fichiers importants à retenir.

Il y a notamment :

- Le fichier main.js dans le dossier src sera le point d'entrée de l'application avec l'instanciation de l'application Vue.js
- Le fichier index.html dans le dossier public est présent pour afficher l'application (il ressemble à ce que l'on a pu faire jusqu'à présent).
- Le fichier App.vue dans le dossier src est là pour servir de premier template global pour l'application Vue.js
- Et il y a un dossier components dans le dossier src pour les composants écrits en .vue que nous utiliserons pour la création de différents composants

2- Une application faite de composants

Architecture de l'application

Un fichier .vue se compose de cette façon :

```
<template>
  <div>

  </div>
</template>

<script>
export default {

}

</script>

<style>

</style>
```

*Dans un fichier .vue il y a en premier le **<template>** qui nous servira de base pour englober notre code View.*

*Il y a ensuite le **<script>** avec un `export default {}` pour exporter tout le ViewModel de notre composant vers son parent.*

*Il y a ensuite le **<style>** qui englobera tout le style css du composant. Il pourra être scoped **<style scoped>** pour ne garder le css uniquement valable pour ce composant.*

*Avec l'extension **Vetur** installée, pour créer cette base plus rapidement il suffira de faire **<vue***

*Noter qu'un fichier composant **.vue** commencera toujours par une lettre majuscule : **Name.vue***



2- Une application faite de composants

Architecture de l'application

Pour importer un fichier composant dans son parent, il faut :

```
<script>
import HelloWorld from './components/HelloWorld.vue'

export default {
  name: 'App',
  components: {
    HelloWorld
  }
}
</script>
```

*Il suffit donc de faire un **import NameComposant from './components/name.vue'***

*Et ensuite, d'utiliser la propriété **components** pour l'ajouter avec le même **NameComposant**.*

*À noter que nous ne sommes plus obligés de faire un '**name-composant**': **NameComposant** mais simplement le **NameComposant***

Bonnes pratiques : Pour éviter toute erreur de compilation, bien penser à avoir un **nom composé** pour les composants

2- Une application faite de composants

Architecture de l'application

Reprendons l'exercice 1 du compteur pour exemple :

```
> ▼ App.vue > ...
1   <template>
2   |   <MonCompteur />
3   </template>
4
5   <script>
6   import MonCompteur from './components/MonCompteur.vue'
7
8   export default {
9       name: 'App',
0       components: {
1           MonCompteur
2       },
3   }
4   </script>
5
6   <style>
7   </style>
```

```
> components > ▼ MonCompteur.vue > ...
1   <template>
2   |   <p>Mon premier bouton composant avec Vue CLI :</p>
3   |   <button @click="nbClics++">{{ nbClics }}</button>
4   </template>
5
6   <script>
7   export default {
8       name: "MonCompteur",
9       data() {
0           return {
1               nbClics: 0,
2           }
3       },
4   };
5   </script>
6
7   <style scoped>
8   </style>
```

*La création est mieux organisée, et permet d'être mieux visible pour le développeur. Le composant **MonCompteur.vue** créé, nous pouvons l'utiliser dans notre layout **App.vue** en faisant un import.*

À noter qu'un fichier Composant doit toujours avoir un name (mettre le même nom que le nom du fichier du compostant).

Exercice numéro 4

04

Mettre en place une application Vue.js 3 avec le setup officiel.

Le but étant de :

- Présenter une liste des inscrits grâce au composant ListInscrits. Il faudra avoir 4 exemples d'inscrits dans un tableau d'objet avec pour les inscrits, prénom nom et âge.
- Les données du tableau seront dans App.vue et c'est le composant Inscrits qui les affichera

Utilisation de Axios

Lors de la création d'une application web, l'utilisation des données provenant d'une API peut se faire de plusieurs manières, mais l'approche la plus populaire est d'utiliser Axios, un client HTTP basé sur les Promesses.

Installation de Axios :

Il faudra installer Axios au global, mais aussi vue-axios. L'explication détaillée est ici

<https://www.npmjs.com/package/vue-axios>

La marche à suivre : `> npm install axios vue-axios`

```
> JS main.js > ...
  import { createApp } from "vue";
  import App from "./App.vue";
  import axios from 'axios'
  import VueAxios from 'vue-axios'

  const app = createApp(App);

  app.use(VueAxios, axios)

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

```
created() {
  this.axios.get('https://randomuser.me/api/?results=10')
    .then((response) => {
    }).catch((err) => {
      console.log(err);
    });
},
```

Une fois l'installation du package, il suffit d'aller dans le `main.js` de l'application pour importer `axios` et `VueAxios` et l'utiliser directement pour son instance de Vue en faisant un `app.use(VueAxios, axios)`

Lors de son utilisation dans les différents composants où un appel API est nécessaire, nous utilisons `axios` dans le cycle de vie `created()` du composant, et nous utilisons `axios` via `this.axios`. Toutes les méthodes de CRUD seront disponibles.

Par exemple, un appel `get`, cela sera `this.axios.get().then().catch()`, un appel post, `this.axios.post().then().catch()`, etc.

Exercice numéro 5

05

Mettre en place une application Vue.js 3 avec le setup officiel.

Le but étant de :

- Présenter une liste des inscrits qui sera récupérée dynamiquement par L'API du site randomuser.me via <https://randomuser.me/api/?results=10> et qui sera utilisée pour récupérer les données des utilisateurs via axios
- Les données récupérées du tableau seront dans App.vue et c'est le composant nommé ListUsers qui les affichera



2- Une application faite de composants

Exercice projet fil rouge



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

Créer le projet fil rouge via le setup officiel de Vue.js :

- 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 ou modification de film dans la liste, pouvoir sélectionner une catégorie via un listing déjà imaginé de catégories disponibles
- Pouvoir ajouter une nouvelle catégorie dans le tableau des catégories de film disponibles + en fonction de l'avancé, pouvoir lister les catégories existantes pour les modifier ou supprimer