Titre professionnel Concepteur Développeur d'Applications

— Le framework Vue JS

Formateur : Hugo VIROT

SOMMAIRE

Objectifs

- Découvrir Vue JS
- Récupérer et afficher des données provenant d'une API
- Comprendre le système de composants
- Utiliser le state avec Pinia

1. Présentation

- **Vue JS a été créé en 2013 par le chinois Evan You.** Développeur travaillant chez Google sur des projets en Angular (framework JS nettement plus complexe), il décide de reprendre les fonctionnalités d'Angular qui lui plaisent, tout en les incluant dans un nouveau framework bien plus léger et facile à maîtriser. C'est ainsi qu'a été créé VueJS, qui est depuis devenu de plus en plus populaire au fil des années.

- **Vue JS** (prononcez "view") **est comme son nom l'indique centré sur la vue restituée à l'utilisateur.** Il est principalement dédié à la création d'interfaces utilisateur et de **SPA (Single Page Applications** : applis web/mobile composées d'une seule page HTML et dont l'affichage est changé dynamiquement via JavaScript).
- **C'est un framework exclusivement front-end : il ne peut pas communiquer directement avec une base de données** (comme Laravel par exemple). Cependant, il permet de **gérer des données externes provenant d'une API (Application Programming Interface,** ou Interface de programmation applicative).
- **Dans le cadre de notre projet multicouche, nous allons le combiner à notre API Laravel** (il sera intégré dans le même dossier).
- La dernière version de Vue JS en date est **Vue JS 3.** C'est celle que nous allons étudier.

2. Le modèle MVVM

➢ **Vue JS fonctionne sur le modèle MVVM (Model-Vue-Vue-Model).** Voyons son fonctionnement.

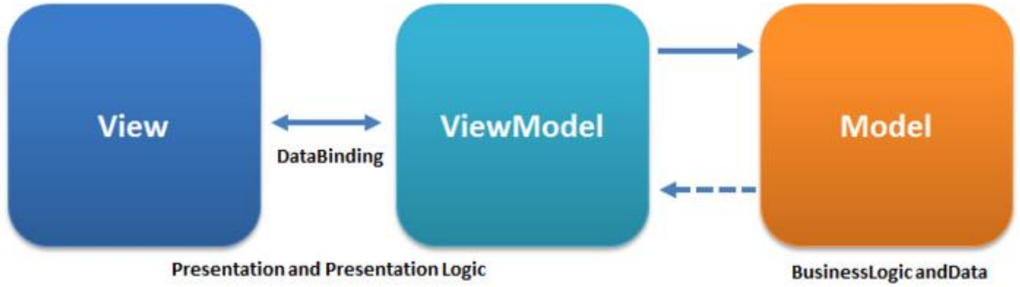


Schéma du modèle MVVM (source : Wikipédia)

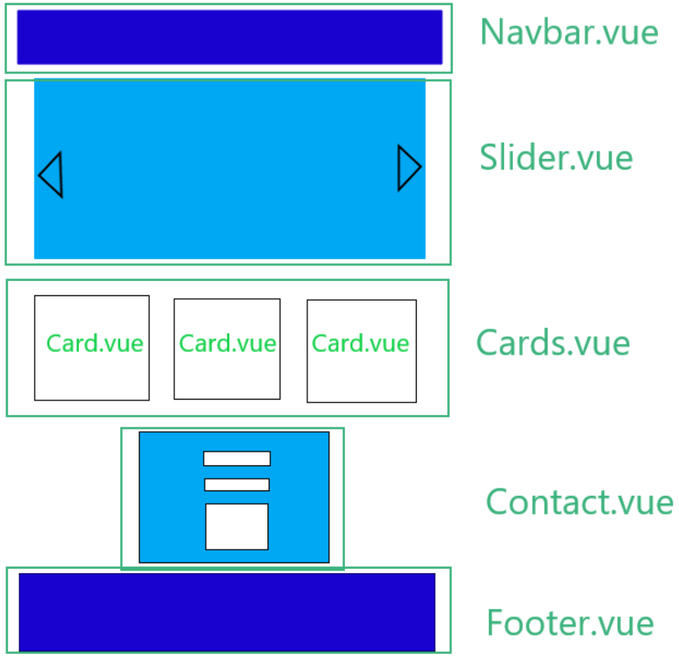
- **Le Model récupère des données via des appels API.** Son rôle est ainsi similaire au Modèle de Laravel (la base de données en moins).
- Ici, pas de Controller : **c'est le ViewModel qui joue ce rôle. Il utilise les données du Model pour les envoyer dans la View** (rendu visuel d'un composant, situé entre les balises <template></template>), qui les affiche. **Le ViewModel fait donc l'intermédiaire entre le Model et la View.**
- **Les données sont liées entre la View et le ViewModel : on parle de Data Binding. Chaque modification des données dans la View se répercute dans le ViewModel.**

➢ **Avec Vue, les 3 parties Model, ViewModel et View se trouvent dans le même fichier** (contrairement à Laravel, qui sépare strictement les composantes du modèle MVC en plusieurs fichiers distincts : les modèles, les vues et les contrôleurs)**.**

3. Les composants

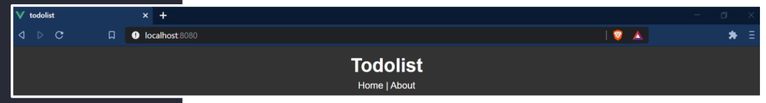
A. Présentation

- Comme de nombreux frameworks JavaScript, **Vue JS utilise des composants** pour composer (justement !) les pages qu'il affiche.
- **On ne conçoit plus le site sous forme de pages entières comme en HTML/CSS ou en PHP, mais sous forme de petits composants réutilisables, qui affichent chacun une petite partie de la page.**



Exemple de découpage d'une page basique en composants

```vue
<template>
  <header class="header">
    <h1>TodoList</h1>
    <div id="nav">
      <router-link to="/">Home</router-link> |
      <router-link to="/about">About</router-link>
    </div>
  </header>
</template>

<script>
export default {
  name: 'Header'
}
</script>

<style scoped>
.header {
  background: #333;
  color: #fff;
  text-align: center;
  padding: 10px;
}
.header a {
  color: #fff;
  padding-right: 5px;
  text-decoration: none;
}
</style>
```



Rendu du Header dans le navigateur (nécessite également un fichier App.vue pour cela)

Exemple de composant : un header très simple avec son propre style : **Header.vue**

(source : *https://medium.com/weekly-webtips/introduction-vue-js-with-a-simple-project-665bad37e0b*)

⇨ Comme on peut le voir sur le schéma, **les composants sont des fichiers en .vue** (extension des fichiers Vue JS).

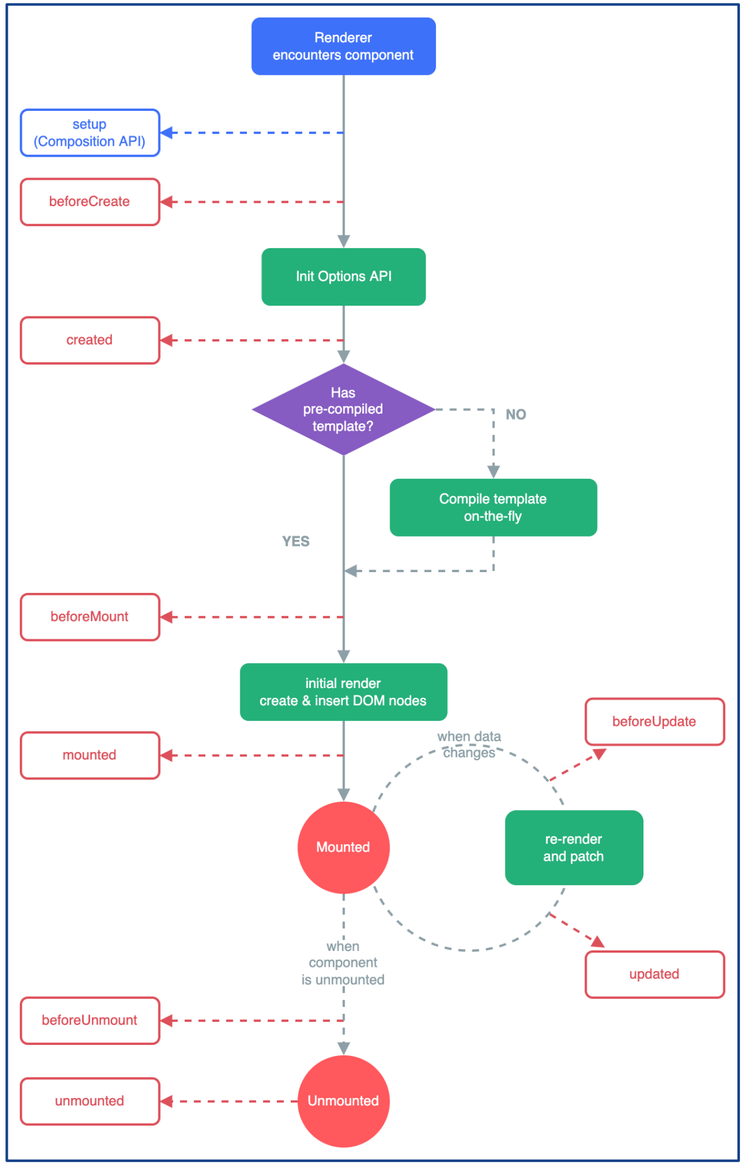


Schéma du cycle de vie d'un composant Vue JS

B. Le cycle de vie

➢ Voici le schéma détaillé **du cycle de vie d'un composant** (source : doc officielle).

➢ **Vue JS créé le composant lorsqu'il a besoin d'être affiché, et le supprime lorsqu'il n'est plus nécessaire** (exemple : changement d'affichage).

➢ **On peut déclencher des actions à certains moments précis du cycle de vie**, en utilisant des fonctions appelées **Hooks** (rectangles rouges sur le schéma).

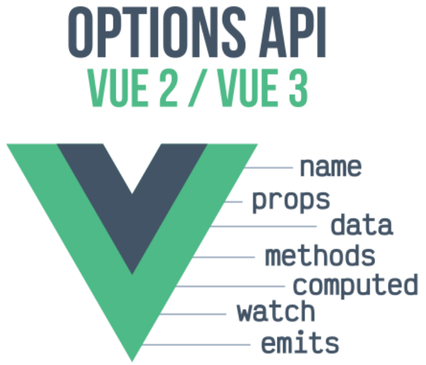
➢ **Les Hooks les plus utilisés sont les suivants :**

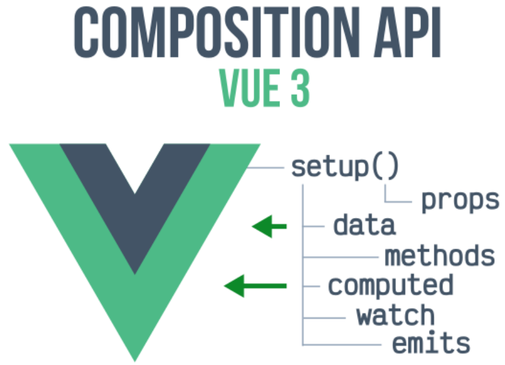
✓ **created :** création du composant (moment idéal pour récupérer des données via des appels API)

✓ **mounted :** une fois que le composant est en place (moment idéal pour interagir avec des éléments du template, car ceux-ci sont affichés)

➢ Pour plus d'informations, consultez cette page :

https://vuejs.org/guide/essentials/lifecycle.html#lifecycle-diagram






```vue
<template>
    <h1>{{ counter }}, {{ doubleCounter }} </h1>
    <button @click="incrCounter">Click Me</button>
    <button @click="increaseByTwo">Click Me For 2</button>
</template>
<script>
export default {
    data() {
        return {
            counter: 0,
            doubleCounter: 0
        }
    },
    methods: {
        incrCounter: function() {
            this.counter += 1;
        },
        increaseByTwo: function() {
            this.doubleCounter += 2;
        }
    }
}
</script>
```

```vue
<template>
    <h1>{{ counter }}, {{ doubleCounter }} </h1>
    <button @click="incrCounter">Click Me</button>
    <button @click="increaseByTwo">Click Me For 2</button>
</template>

<script setup>
    import { ref } from 'vue'

    let counter = ref(0);
    const incrCounter = function() {
        counter.value += 1;
    }

    let doubleCounter = ref(0);
    const increaseByTwo = function() {
        doubleCounter.value += 2;
    }
</script>
```

Comparaison avec un exemple simple (compteur qui augmente de 1 ou de 2)

C. Composition API VS Options API

- Avant fin 2020 (date de sortie de Vue JS 3), la syntaxe des composants Vue JS était basée sur **l'Options API.** Principe : le composant est codé avec **une syntaxe « verbeuse » qui détaille ses caractéristiques** : data, props, méthodes…

- Depuis la version 3, les composants de Vue s'écrivent selon les règles de la **Composition API**. Le système, assez différent de l'options API, permet des **composants au code plus court et plus lisible,** ressemblant davantage à du JavaScript *vanilla* (de base).

- La syntaxe Options API est encore utilisable, mais **c'est la Composition API qui est recommandée** par le site officiel. **Nous utiliserons donc cette dernière**.

- Pour en savoir plus, consultez cette page de la documentation officielle : *https://vuejs.org/guide/extras/composition-api-faq.html#composition-api-faq*

```vue
<script setup>
import { ref } from "vue";
const count = ref(0);

console.log(count);

const addToCount = () => count.value++;
const subtractFromCount = () => count.value--;

</script>

<template>
  <main>
    <div>
      <h4>
        The current count is ...
      </h4>
      <h1>{{ count }}</h1>
      <button @click="subtractFromCount">-</button>
      <button @click="addToCount">+</button>
    </div>
  </main>
</template>

<style scoped>
main {
  height: 100vh;
  width: 100vw;
  justify-content: center;
  align-items: center;
}
</style>

```

D. Structure du composant

- Un composant Vue JS comporte **3 parties** :
  - **Script** : la partie logique (100% JS). Permet d'initialiser le composant, d'importer les éléments nécessaires (composants, extensions, syntaxes Vue) et de déclarer les variables. **Important : ne pas oublier le mot clé setup, qui permet d'utiliser la Composition API.**
  - **Template** : la partie visuelle du composant. Elle comporte des balises html, des variables issues des données du composant, ainsi que des **directives** Vue (voir page suivante)
  - **Style** (facultatif) : le style du composant. On peut ajouter l'attribut **scoped** pour le limiter au composant lui-même.
- Chaque partie est englobée dans des balises qui portent son nom (voir ci-contre).
- Il n'y a pas d'ordre imposé. On les place en général dans l'ordre ci-dessus.

4. Concepts et syntaxes à connaître

A. Afficher des données

➢ Utilisez la syntaxe **{{ variable }}** pour **afficher une variable dans un template**, entre des balises html

```
<span>Message: {{ msg }}</span>
```

NB : la variable doit être déclarée dans la partie script (voir plus loin).

B. Les directives

Vue utilise une syntaxe particulière, avec des **éléments spéciaux intégrés directement au html** et appelés **directives**. Exemples :

V-for pour les **boucles for**

```
<div v-for="item in items">
  {{ item.text }}
</div>
```

V-if et **v-else** pour les **conditions**

```
<p v-if="seen">Maintenant vous me voyez</p>
```

V-bind:variable ou son raccourci **:variable** : pour **injecter une variable en tant que valeur d'un attribut** (les doubles accolades ne fonctionnent pas)

```
<div v-bind:id="dynamicId"></div>
```

(source des captures : doc officielle : *https://vuejs.org/guide/essentials/template-syntax.html*. Rendez-vous sur cette page pour en savoir plus)

C. La réactivité (ref)

- On l'utilise pour **déclarer des variables.** Exemple : compteur initialisé à zéro :

```
const count = ref(0);
```

- La variable est créée **enveloppée d'un objet doté d'une propriété value**, donc la valeur est le **paramètre passé à ref**.

NB : Pour **changer la valeur** d'une variable déclarée avec **ref**, **on doit cibler sa value** comme ici

```
console.log(count) // { value: 0 }
console.log(count.value) // 0

count.value++
console.log(count.value) // 1
```

- Les variables créées avec **ref** sont dites **réactives** :

1. Dès la création du composant, Vue les **surveille** attentivement
2. en cas de **changement de leur valeur**, le système de réactivité de Vue JS le **détecte**
3. Le composant est alors **mis à jour automatiquement** en fonction de ce changement

- Avant d'utiliser **ref**, on l'importe comme ceci :

```
import { ref } from "vue";
```

Important : ne pas oublier le mot clé setup, qui permet d'utiliser la Composition API. Sans celui-ci, la réactivité ne fonctionnera pas.

- Autres exemples :

```
<script setup>
  import { ref } from "vue"
  const showModal = ref(false)
  const newNote = ref("")
  const errorMessage = ref("")
  const notes = ref([])
```

Déclaration de variables de différents types

```
notes.value.push({
  id: Math.floor(Math.random() * 100),
  text: newNote.value,
  date: new Date(),
  backgroundColor: getRandomColor()
})
showModal.value = false
```

Modification de la valeur de notes et showModal

⇨ Plus d'informations : *https://vuejs.org/guide/essentials/reactivity-fundamentals.html*

D. Les props (propriétés)

- Un des principes de base de Vue JS est de **découper le code en petits composants réutilisables** (voir **3. Les composants**).

  => Dans cette optique, il est fréquent **d'utiliser un composant dit « enfant » dans un composant dit « parent ».**

- Le composant parent va très souvent **passer des variables au composant enfant**. Ces variables sont appelées **props** (pour *properties*, propriétés) dans le composant enfant.

- Voici le code à mettre en place :

Composant parent

1. Import de ref
2. Import du composant enfant
3. Déclaration de la variable, contenant une chaîne de caractères
4. Instanciation du composant enfant + passage de la variable en tant que props

Parent.vue

```
src > components > Parent.vue > ...
1   <script setup>
2   import { ref } from "vue";
3   import Enfant from "./Enfant.vue"
4
5   const variable = ref("hello world")
6   </script>
7
8   <template>
9   <Enfant :texte="variable" />
10  </template>
```

Composant enfant

1. Déclaration de la variable attendue en props (via defineProps)
2. Affichage de cette variable dans le template

Enfant.vue

```
src > components > Enfant.vue > ...
1   <script setup>
2   defineProps(['texte'])
3
4   </script>
5
6   <template>
7       <p>{{ texte }}</p>
8   </template>
```

Résultat : hello world

- On peut faire passer **plusieurs props** du parent à l'enfant :

Parent.vue

src > components > Parent.vue > ...

```vue
<script setup>
import { ref } from "vue";
import Enfant from "./Enfant.vue"

const message = ref("hello world")
const chiffre = ref(1)
const vrai = ref(true)

</script>

<template>
<Enfant :texte="message" :nombre="chiffre" :booleen="vrai" />
</template>
```

Enfant.vue

src > components > Enfant.vue > ...

```vue
<script setup>
defineProps(['texte', 'nombre', 'booleen'])
</script>

<template>
    <p>{{ texte }}</p>
    <p>{{ nombre }}</p>
    <p>{{ booleen }}</p>
</template>
```

⇨ Résultat :

```
hello world
1
true
```

5. Installation

➢ Deux possibilités :

- **installation de Vue de façon isolée**
- **ajout dans un projet existant** (multicouche) : voir cours ***Créer un projet multicouche Laravel VueJS*** pour intégrer Vue JS dans votre projet Laravel

➢ Pour lancer un projet Vue isolé :

1. Vérifiez d'abord que **Node JS** est bien **installé et à jour** :

```
C:\Users\hugov>node -v
v21.5.0
```

⇨ Vous devez avoir la **version 20.12** au minimum. Si ce n'est pas le cas, téléchargez la version la plus récente *ici*

2. Même chose pour **NPM** :

```
C:\Users\hugov>npm -v
10.5.2
```

⇨ Si vous n'avez pas la **version 10.5.2** au minimum**,** mettez-le à jour ainsi : `npm install -g npm@latest`

3. lancez ensuite la commande suivante : `npm create vue@latest`

Entrez le nom de votre projet, puis **ajoutez les options suivantes : Vue Router, Pinia et Vue Devtools** (répondez Non à toutes les autres)

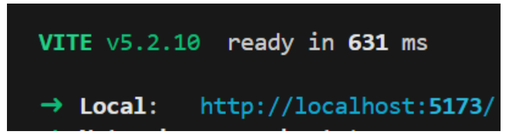
```
Vue.js - The Progressive JavaScript Framework

√ Nom du projet : ... test
√ Ajouter TypeScript ? ... Non / Oui
√ Ajouter le support de JSX ? ... Non / Oui
√ Ajouter Vue Router pour le développement d'applications _single page_ ? ... Non / Oui
√ Ajouter Pinia pour la gestion de l'état ? ... Non / Oui
√ Ajouter Vitest pour les tests unitaires ? ... Non / Oui
√ Ajouter une solution de test de bout en bout (e2e) ? » Non
√ Ajouter ESLint pour la qualité du code ? ... Non / Oui
√ Ajouter l'extension Vue DevTools 7 pour le débogage ? (expérimental) ... Non / Oui
```

4. Ouvrez votre projet dans VS Code.

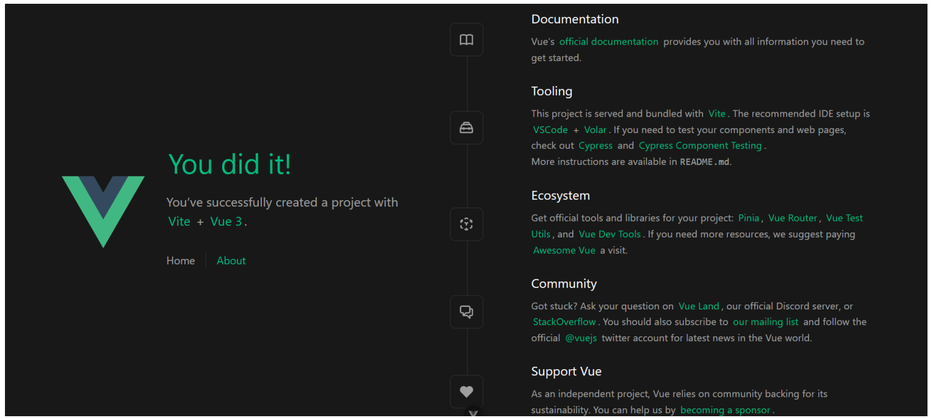
5. Lancez la commande `npm install` pour installer les dépendances.

6. Lancez le serveur avec `npm run dev` .

7. Faites un CTRL + clic gauche sur l'url affichée, pour ouvrir le projet avec votre navigateur.



=> Vous devez obtenir cet écran :

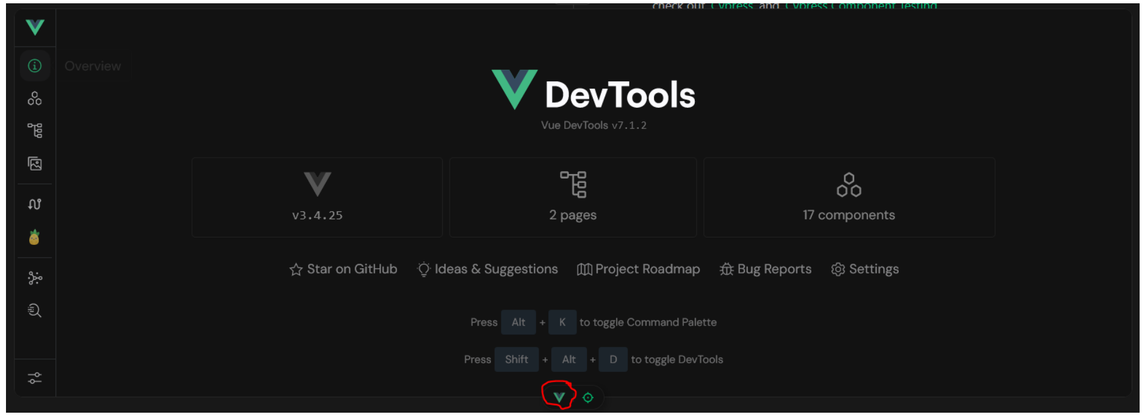


8. En cliquant sur le « V » en bas de l'écran, vous pouvez afficher les **outils de développement** (DevTools).

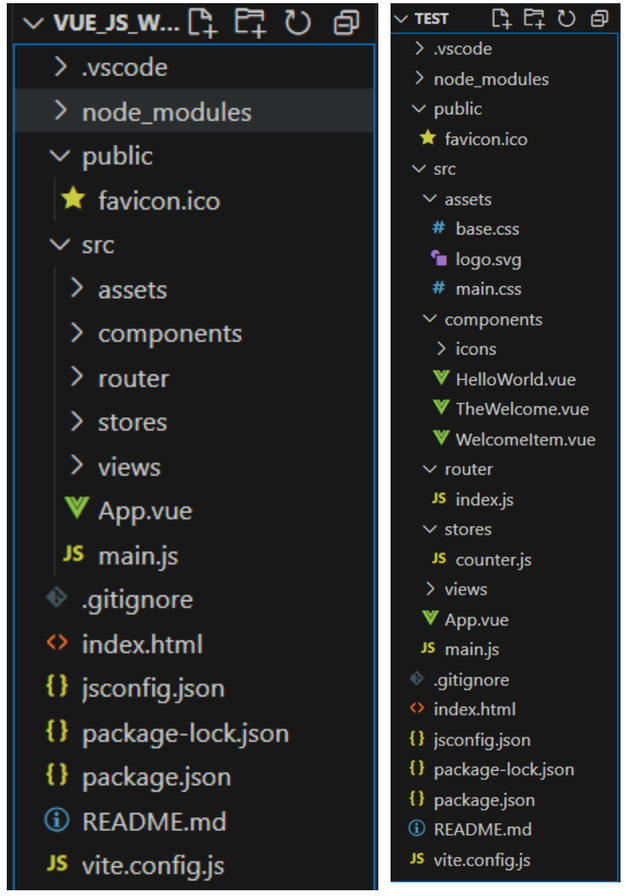
   ⇨ Ils permettent de visualiser :

   - L'arbre des composants
   - Les variables utilisées dans chaque composant
   - Les routes

Pensez à les utiliser, ils sont très pratiques !



6. Arborescence



Arborescence du template de base dépliée et repliée (tout est affiché sauf node_modules)

Voici les fichiers principaux :

- **App.vue** : situé dans le dossier **src**, **c'est le composant principal.** Il va contenir la **structure de base du site**, avec un Header, un Footer, et une partie centrale qui change en fonction de la route. Il sert également très souvent à afficher l'accueil.

- **index.html** : situé dans le dossier **public**, **c'est la page html de base.** Il contient une **div qui comporte l'id app : l'application va s'afficher à cet endroit.**

- **main.js** : **fichier "racine" de l'application. C'est lui qui instancie l'application, et l'injecte dans la div comportant l'id "app" de l'index.html.**

- **HelloWorld.vue** : les **composants** comme celui-ci se trouvent dans le dossier **src/components**.

Remarques

- Vue JS est une **librairie** (comme React par exemple). **Il n'impose donc pas de structure de fichiers**. Cependant, **il est conseillé de suivre la structure de base** sans trop la modifier.
- **Stockez les images dans le dossier *src/assets*** pour un projet Vue JS indépendant. Elles seront stockées dans le dossier *public/images* de Laravel dans le cadre d'un projet multicouche.
- Comme avec Laravel, un projet VueJS contient un **package.json** (court) et un **package-lock.json** (détaillé) avec la **liste des dépendances JavaScript**.

7. Le routage avec Vue Router

➢ Principe et installation

➢ Nous utiliserons le routeur officiel de Vue JS : **Vue Router**.

➢ **Si vous avez répondu « oui » lors de l'installation, Vue Router est déjà présent dans votre projet**. Sinon, il vous faudra installer le package npm :

```
npm i vue-router
```

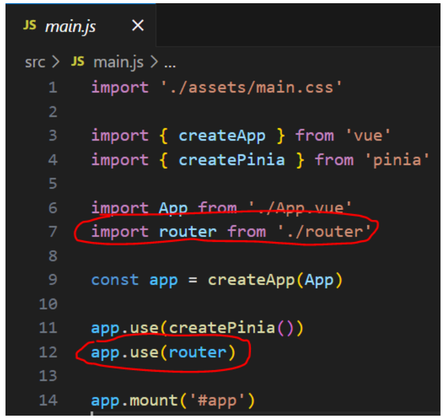
➢ Très utile, **il permet de simuler une navigation sur plusieurs pages web**, avec des **"pseudo-URL"** dans la barre d'adresse et des **changements d'affichage** dans le composant App.

En pratique, on reste dans l'index.html et dans le composant App

⇨ Cela permet de créer une **SPA** (**Single-Page Application**) proposant plusieurs affichages différents. Voir les liens utiles pour plus d'infos.

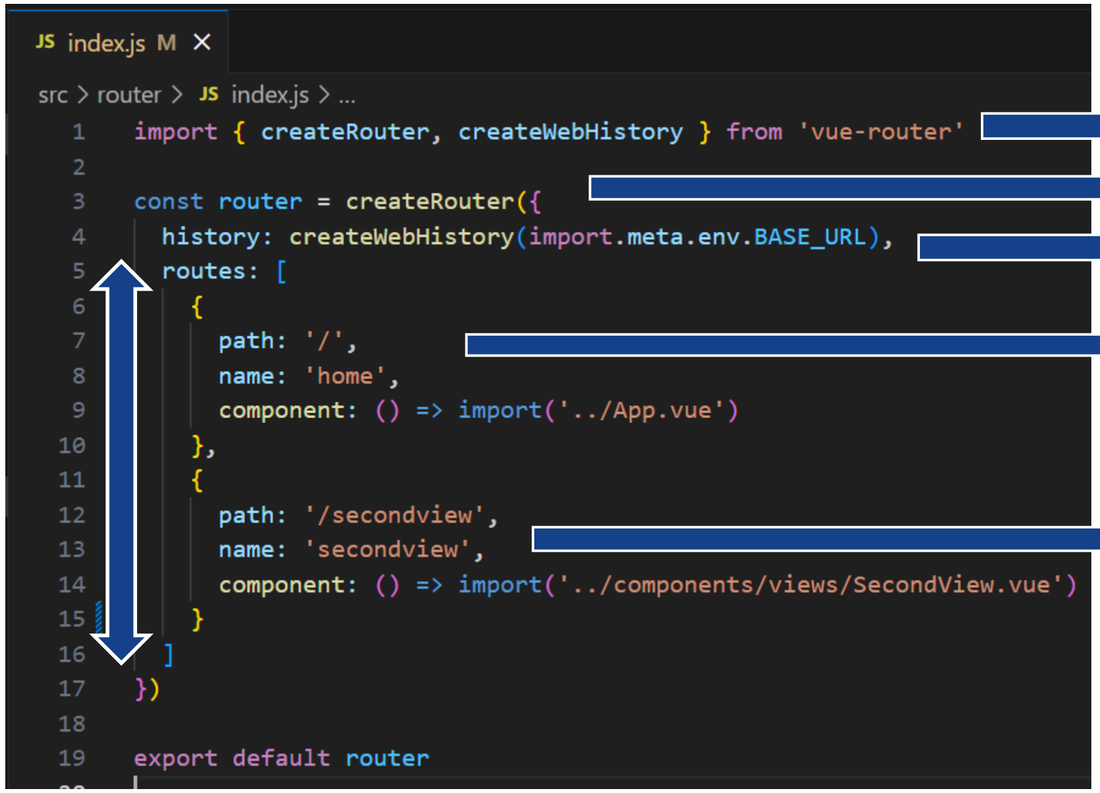
Voici le lien de la doc officielle de Vue router: *https://router.vuejs.org/*

➢ Dans main.js, le code suivant permet **d'intégrer Vue router à votre application**. Il est ajouté automatiquement lorsque l'on choisit Vue Router au lancement du projet (à ajouter manuellement sinon).



➢ Initialiser le router et déclarer des routes

➢ Tout cela se passe dans le fichier **index.js** placé dans le dossier router (les deux sont créés automatiquement en choisissant Vue Router lors du lancement du projet, à créer manuellement sinon). Voici sa structure :



Import des fonctions nécessaires

Création du router

Activation du mode **webHistory** pour que le *path* (chemin) de chaque page soit visible dans la barre de navigation)

Création de la **première route** (accueil / composant *App*) : path + nom de la route + composant associé

Création d'une **seconde route** (vers un autre composant, *SecondView*)

Flèche verticale : **tableau des routes**

✓ **Pour ajouter une route, placez un nouvel objet dans le tableau des routes, en renseignant au moins le path et le composant associé.**

✓ Le name est facultatif, mais permet notamment de passer des paramètres plus facilement (voir page suivante).

- Pour **faire passer un paramètre** via une route, on utilise la syntaxe suivante :

```
{
  path: '/user/:id',
  name: 'profile',
  component: () => import('../components/views/Profile.vue')
}
```

- On y accède ensuite ainsi dans le composant concerné :

```
<template>
  <div>
    <!-- The current route is accessible as $route in the template -->
    User {{ $route.params.id }}
  </div>
</template>
```

- On peut également faire passer **plusieurs paramètres** :

```
{
  path: '/user/:id/posts/:postid',
  name: 'Post',
  component: () => import('../components/views/Post.vue')
}
```

Créer des liens dans les pages

- Pour cela, on utilise la syntaxe **router-link**:
  - L'attribut ***to*** est l'équivalent du href d'une balise a. On y indique le **path** de la route ciblée.
  - On peut également y indiquer le **name**, comme ceci :

```
<router-link to="/">Accueil</router-link>
```

```
<router-link :to="{name: 'home'}"> Home page </router-link>
```

- Pour faire passer un ou plusieurs **paramètre(s)** quand c'est nécessaire:
  - **Privilégiez le name** dans ce cas-là (plus facile à écrire)

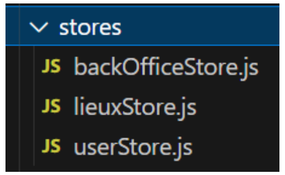
```
<router-link class="link" :to="`/MovieDetails/${movie.id}`">
```

```
<router-link :to="{ name: 'profile', params: { username: 'erina' } }">
  User profile
</router-link>
```

8. Le système de store avec Pinia

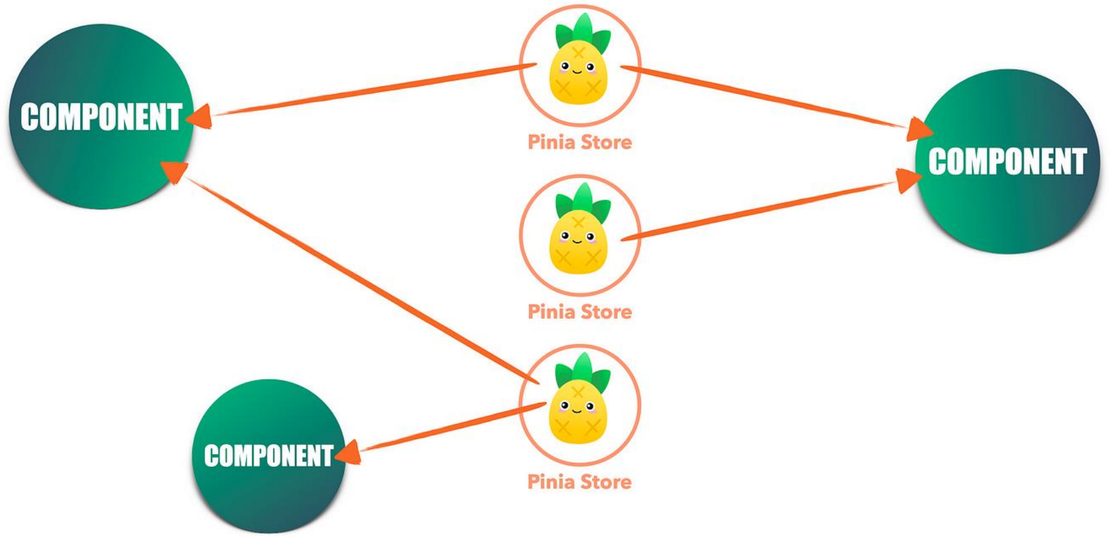
Doc officielle : *https://pinia.vuejs.org/*

- Dans Vue JS (et dans d'autres frameworks front-end comme React), il est possible de **mettre en place un système de store.** Ce store permet de **stocker des variables globales, accessibles dans tous les composants**.
- **On l'implémente grâce à la librairie Pinia** (qui remplace Vue X, plus complexe et maintenant moins utilisée).
- Pinia fonctionne ainsi :
  - On crée un **store** (un fichier) **par type de données** utilisées dans l'application : *userStore, articlesStore*, etc.
    - ⇨ Il y aura donc plusieurs stores différents dans votre projet phare.



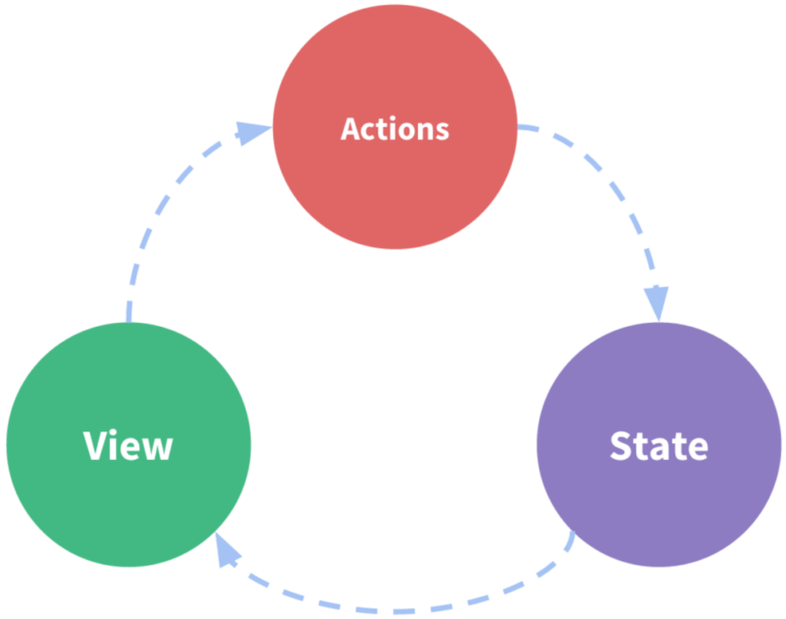
  - Chaque store contient un **State** : il s'agit d'une fonction renvoyant un objet qui **contient les variables globales.** Dans un composant, on va **accéder à un (ou plusieurs) store(s)**, pour afficher des variables de son state dans le template, et éventuellement les modifier.
  - Le store contient également des **getters**, qui permettent **d'effectuer des traitements spéciaux** sur les variables du state (exemple : filtrage d'un tableau) et de récupérer le résultat.

    NB : inutile de créer un getter pour accéder à chaque variable : Pinia permet de le faire sans passer par un getter.
  - Enfin on y trouve des **actions**, qui servent à **définir / modifier les variables du state.**



Le schéma suivant permet de bien comprendre l'interaction avec le store dans un composant:

- **View** : c'est le **rendu visuel** de votre page (situé entre les balises **template** de chaque composant). On y **affiche les données issues du state d'un store** (elles peuvent être modifiées), et on y **déclenche des actions** (voir ci-dessous).
- **Action** : **déclenchée par l'utilisateur** (clic sur un bouton, ajout d'un élément à une liste, lancement d'un tri, etc), **l'action modifie une variable du state** d'un store.
- **State :** le state concerné **voit donc une de ses variables changer. Ces changements vont se répercuter sur la vue**.



Fonctionnement des interactions d'un composant VueJS avec le state de Pinia

Exemple d'utilisation de Pinia avec un compteur

- On **installe pinia** dans le projet : `npm i pinia` (seulement si l'option n'a pas été choisie à l'installation)
- On **l'ajoute au projet** dans le fichier main.js (idem)

main.js

```
src > main.js > ...
1  import { createApp } from 'vue'
2  import App from './App.vue'
3
4  import { createPinia } from 'pinia'
5  const pinia = createPinia()
6
7  createApp(App).use(pinia).mount('#app')
```

- On **initialise le store en créant le fichier counterStore.js** (dans le dossier src ou un dossier stores si il y en a plusieurs) : avec un **state** et des **actions. Deux syntaxes sont possibles : « options store »** et **« setup store »** (même résultat dans les deux cas, choisissez celle que vous préférez)

counterStore.js

```
counterStore.js > useCounterStore > actions > decrement
1   import { defineStore } from "pinia";
2
3   export const useCounterStore = defineStore('counter', {
4       state: () => ({ count: 0 }),
5       actions: {
6           increment() {
7               this.count++
8           },
9           decrement() {
10              this.count--
11          }
12      }
13  })
14
```

options store

counter.js

```
src > stores > counter.js > ...
1   import { ref } from 'vue'
2   import { defineStore } from 'pinia'
3
4   export const useCounterStore = defineStore('counter', () => {
5     const count = ref(0)
6
7     function increment() {
8       count.value++
9     }
10
11    function decrement() {
12      count.value--
13    }
14
15    return { count, increment, decrement }
16  })
```

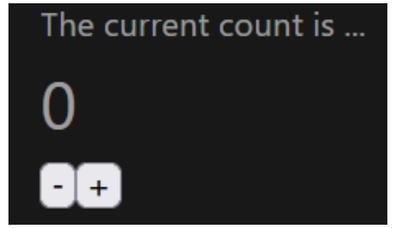
setup store

- Dans le **template** d'un composant (= partie vue / view du modèle MVVM), **on affiche le compteur** : la variable count, un bouton + et un bouton -

App.vue

src > App.vue > {} style scoped > main

```vue
<script setup>

import { useCounterStore } from "../counterStore"
import { storeToRefs } from 'pinia'
const store = useCounterStore()
const { count } = storeToRefs(store)

console.log(count);

</script>

<template>
  <main>
    <div>
      <h4>
        The current count is ...
      </h4>
      <h1>{{ count }}</h1>
      <button @click="store.increment()">+</button>
      <button @click="store.decrement()">-</button>
    </div>
  </main>
</template>
```



NB : storeToRefs permet d'accéder aux variables du state de façon réactive : un changement au niveau de la variable « count » se répercute instantanément sur le template du composant.

- **on clique sur le bouton + qui ajoute 1 au compteur => cela déclenche l'action increment du counterStore => la variable count passe à 1 dans le state du counterStore => l'affichage passe également à 1.**
- Même principe avec le bouton - (le compteur baissera de 1).

Liens utiles

Description	Adresse
Doc officielle	https://vuejs.org/guide/introduction.html
Le modèle MVVM	https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel
Présentation générale et exemples	https://www.numendo.com/blog/framework/vue-js-framework-javascript/
Créer des routes avec VueRouter	https://router.vuejs.org/
Autre site sur VueRouter	https://flaviocopes.com/vue-router/
Tour d'horizon de Vue 3	https://madewithvuejs.com/blog/vue-3-roundup
Composition API VS Options API	https://fjolt.com/article/vue-composition-api-vs-options-api