

VUE.JS – Développement d'applications Web

11/09/2024 – 13/09/2024

Glodie Tshimini



m2information.fr

Glodie Tshimini

Formateur et développeur depuis 2017

- Avant-hier, développeur chez **INRAE** à Paris
- Hier, **Tech Lead** dans une agence digitale à Saint Raphaël
- Aujourd'hui, **indépendant** dans la formation et le développement d'applications Web
- Licence Analyste Concepteur Systèmes d'Information et de Décision
- Titre professionnel Concepteur Développeur Informatique
- Autodidacte

Certifié PSD I

Certifié Coderbase IT Expert Trainer

Certifié Coderbase IT Expert Trainer At POE

Email : contact@tshimini.fr

HORAIRES ET PAUSES

Lundi

- 9h00 - 12h30
- 13h30 - 17h30

Mardi

- 09h00 - 12h30
- 13h30 - 17h30

Mercredi

- 09h00 - 12h30
- 13h30 - 16h00

Les pauses

- Matin : 15 minutes
- Déjeuner : 1 heure
- Après-midi : 15 minutes



Émargement et évaluation formateur

Chaque matin et chaque après-midi, vous devrez signer les feuilles d'émargement. Les "X" et initiales ne sont pas autorisés, merci de bien vouloir émarger avec une signature.

Le dernier jour de la formation, avant 14h00, vous aurez à saisir une évaluation formateur. Cette évaluation est obligatoire, il doit être rempli consciencieusement.

Déroulement de la formation

- 40 % théorie 60% pratique
- Daily meeting
- Exercices immédiats d'application du cours
- Code refactoring et code review
- Les énoncés des exercices et corrections seront accessibles depuis le [dépôt public GitHub](#)

GitHub de la formation



<https://github.com/glo10/vuejs-11092024>

Un formateur à votre écoute



Google Forms

<https://docs.google.com/forms/d/e/1FAIpQLScZwkm4568aMJtZxBtVuKk26Q5yfpPlw1zdXiT7QaD0MgQ9Qw/viewform>

OBJECTIFS PÉDAGOGIQUES

- Mettre en œuvre le Framework Vue.js
- Utiliser Vue.js dans le cadre d'une application SPA (Single Page Application) et d'applications clientes plus conventionnelles

NIVEAU REQUIS ET PUBLIC CONCERNÉ

Niveau requis

- Très bonne connaissance (théorie et pratique)
 - HTML 5
 - CSS 3
 - JavaScript

Public concerné

- Développeurs de sites Web désirant développer des applications actuelles Front End.

PLAN DE COURS



PLAN DE COURS

I. Les fondamentaux du Framework

- A. Installations
- B. Introduction
- C. CDN Vue
- D. Vue-CLI
- E. Propriétés
- F. Directives
- G. Méthodes

II. Les composants

- A. Single File Components
- B. Cycle de vie d'un composant
- C. Echange des données entre composants
- D. Slots

III. Les notions avancées

- A. Navigation avec Vue Router
- B. Transitions et animations
- C. Store avec Vuex

INSTALLATIONS



INSTALLATIONS



1. Éditeur de code [VSCODE](#)
2. Environnement d'exécution [NODEJS](#)
3. Extensions [Vue.js devtools](#) pour vos navigateurs préférés
4. Extension [VSCODE Vue Language Features \(Volar\)](#)

I. LES FONDAMENTAUX DU FRAMEWORK

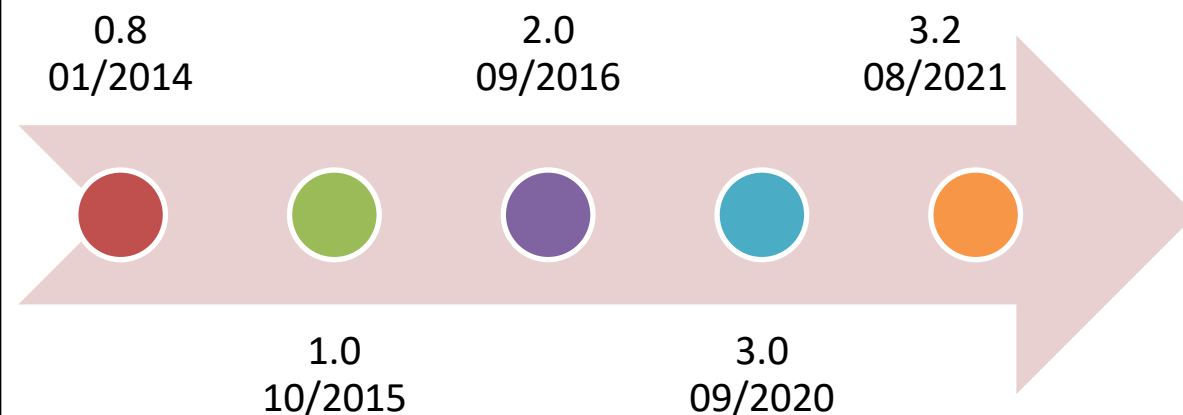


INTRODUCTION



- Créée en 2014 par [Evan You](#)
 - Ancien développeur logiciel chez Google
 - A travaillé sur [AngularJS](#) (un autre Framework JavaScript)
 - A l'initiative d'autres projets comme [Vite](#) (outil de développement front-end)
- Version actuelle **v3.4.38**
- Version 3 est écrit en [TypeScript](#) avec le typage de chaque élément (fonction, constante, etc.)

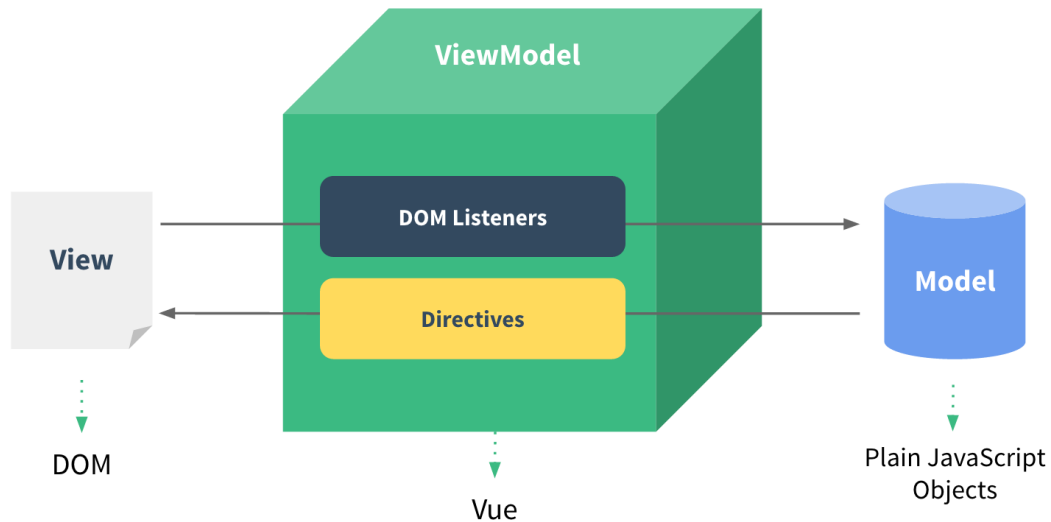
Toute l'historique des versions



Caractéristiques du Framework

- [Site officiel vuejs.org](https://vuejs.org)
 - Framework JavaScript dédié au Front-end
 - 2 APIs mise à disposition pour utiliser VUE JS
 - **Options API**
 - Existe depuis les premières versions et supportés dans la version 3
 - Rédaction orientée objet, très spécifique, enfermé dans l'écosystème Vue et très peu réutilisable
 - La logique est dispersée entre les différents concepts de Vue JS
 - **Composition API**
 - Introduite depuis la version 2.7 de Vue
 - Facilite la réutilisation du code rédigé en dehors
 - Plus naturel dans l'écriture de son code
- Open source
 - [ECMAScript 5](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ECMAScript_5)
 - Basé sur le [MVVM](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93model) (Modèle-vue-vue-Modèle)
 - Utilisation et réutilisation des composants
 - Types d'application
 - [SPA](https://en.wikipedia.org/wiki/Single-page_application) (Single Page Application)
 - [SSR](https://en.wikipedia.org/wiki/Server-side_rendering) (Server Side Rendering)
 - [Web Mobile](https://en.wikipedia.org/wiki/Web_mobile)
 - [Applications de bureau](https://en.wikipedia.org/wiki/Desktop_application)
 - [Application 3D](https://en.wikipedia.org/wiki/3D_application)
 - Web classique

[Source image vuejs.org](http://vuejs.org)



MVVM

Architecture et méthode de conception accès sur les liaisons et les événements.

- Modèle : couche de données
- Vue : couche de présentation IHM et événements
- Vue-modèle : couche de logique métier et de liaison

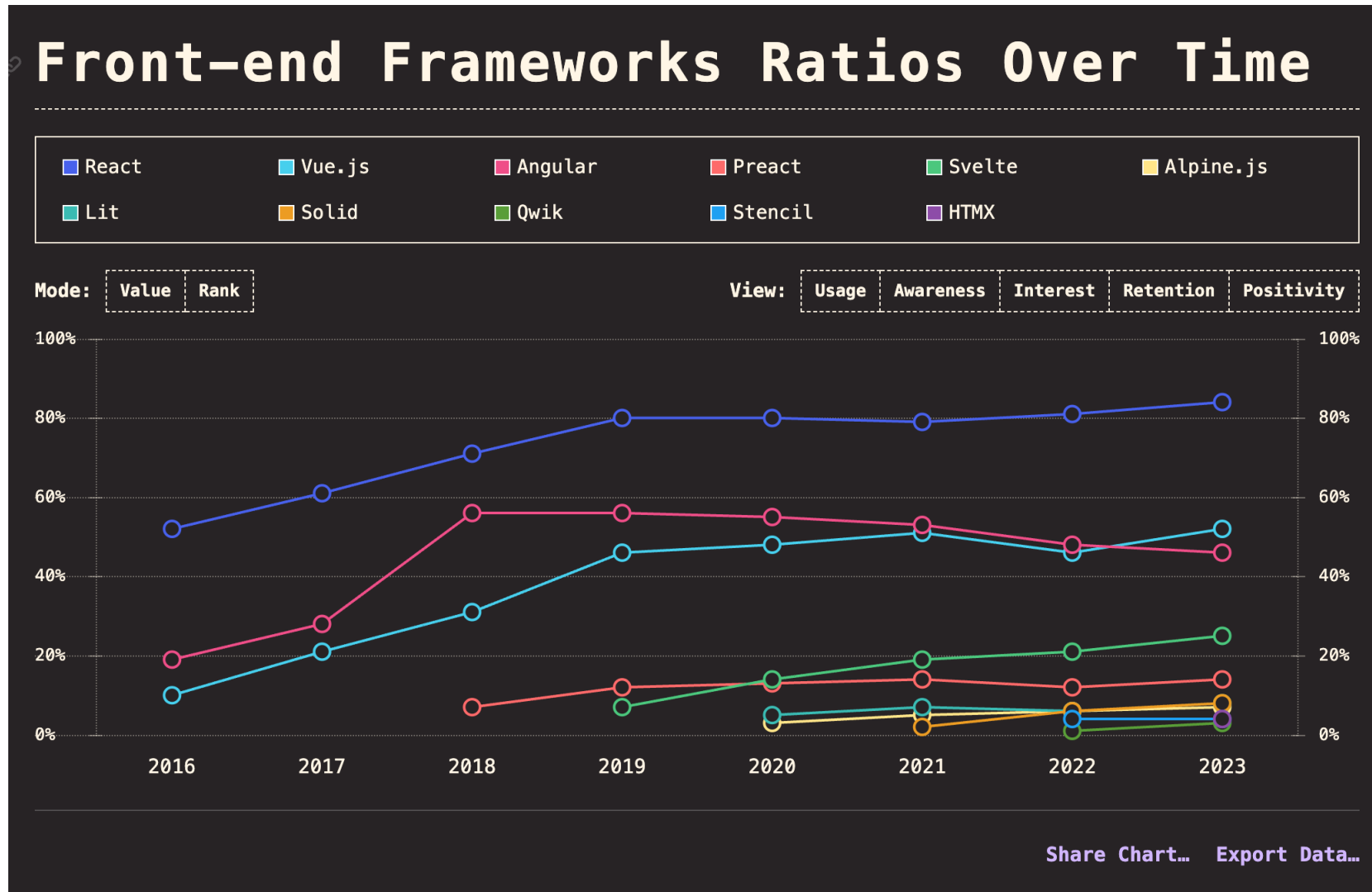
Comparaison : Vue.js vs React vs Angular

source image natek 2022






















 natek work IT with us	 React	 ANGULAR	 Vue.js
	A declarative, efficient and flexible JavaScript library for building user interfaces.	One framework. Mobile & Desktop	A progressive, incrementally adoptable JavaScript framework for building UI on the web.
	116 993 ★	59 302 ★	121 050 ★
Original author	Jordan Walke	Misko Hevery	Evan You
Developers	Facebook	Google	
Initial release	May 29, 2013	October 20, 2010	February 2014
Npm weekly downloads	3 940 035	433 361	709 943
Size	109.7 kB production 774.7 kB development	167 kB production 1.2 MB development	30.67 kB production 279 kB development
Easy to learn	Medium	Learn TypeScript	Yes
Coding speed	Normal	Slow	Fast
Documentation	✓	✓	✓
Performance	✓	✓	✓
Startup time	Quick	Longer due to its large codebase	Quick
Complete web apps	Needs to be integrated with many other tools	Can be used on standalone basis	Requires third party tools
Data binding	Uni-directional	Bi-directional	Bi-directional
Rendering	Server side	Client side	Server side
Model	Virtual DOM	MVC	Virtual DOM
Code reusability	No, only CSS	Yes	Yes, CSS and HTML
When to use	Production, custom UI apps	Production, esp. enterprise apps with Material UI	Startups, production
Big companies	Facebook, Yahoo, Netflix, atlassian, KhanAcademy	Netflix, Upwork, PayPal, TheGuardian	Facebook, Alibaba, Adobe, Grammarly, GitLab

- Courbe d'apprentissage et de progression plus simple et rapide
- Plus légère
- Communauté croissante
- Pour des petits projets

Comparaison : Vue.js vs React vs Angular ([source state of js 2023](#))



Comparaison : Vue.js vs React vs Angular : [source image pleodigital](#)

	 YouTube		 Office				
	facebook				yahoo!	 ATlassian	NETFLIX
			 Adobe		 REUTERS		

UTILISATION À PARTIR DU CDN VUE



Utilisation classique depuis un CDN

Source image Vuejs.org

```
<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
<div id="app">{{ message }}</div>
<script>
  const { createApp } = Vue

  createApp({
    data() {
      return {
        message: 'Hello Vue!'
      }
    }
  }).mount('#app')
</script>
```

html

- CDN (Content Delivery Network)
 - Serveur de relais pour servir des fichiers Web
 - Lien vers de CDN de Vue JS à inclure dans une page HTML pour avoir accès à VueJS
<https://unpkg.com/vue@3.3.4/dist/vue.global.js>
 - Le contenu du CDN peut être stocké dans un fichier local pour garantir la disponibilité
- Déstructuration (décomposition) de l'objet Vue pour utiliser sa fonction *createApp()* et un *sélecteur CSS*

EXERCICE





Exercice 1 : première application avec le CDN

0-exercices/ex1.md

VUE-CLI



[Source image vuejs.org](#)

Install:

```
npm install -g @vue/cli  
# OR  
yarn global add @vue/cli
```

sh

Create a project:

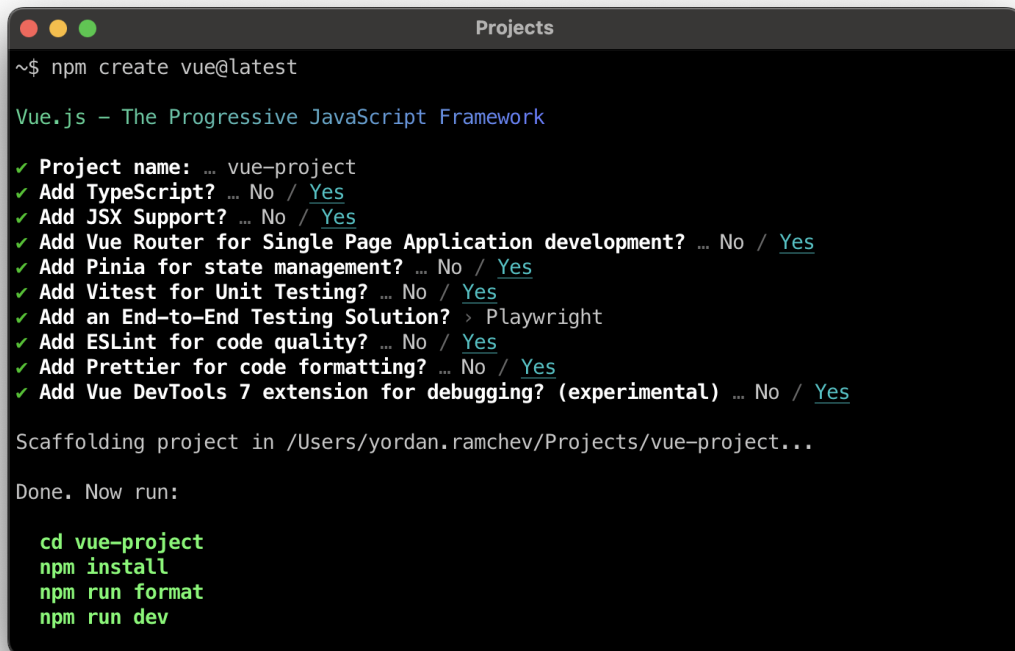
```
vue create my-project  
# OR  
vue ui
```

sh

- [Vue-cli](#) est l'interface en ligne de commande pour créer des applications vue et bien plus
- Avantages
 - **Standardisation** de la création à partir d'un template *de projet*
 - **Optimisation** du JS
 - Utilisation du [Sass](#) (préprocesseur CSS)
 - Etc.
- Vue-cli nécessite l'utilisation d'un [Bundler](#)
- Il est basé sur le bundler [Webpack](#)
- 5.0.8 version actuelle (juillet 2022)

Création du projet à partir de npm ou yarn

Source image create-vue

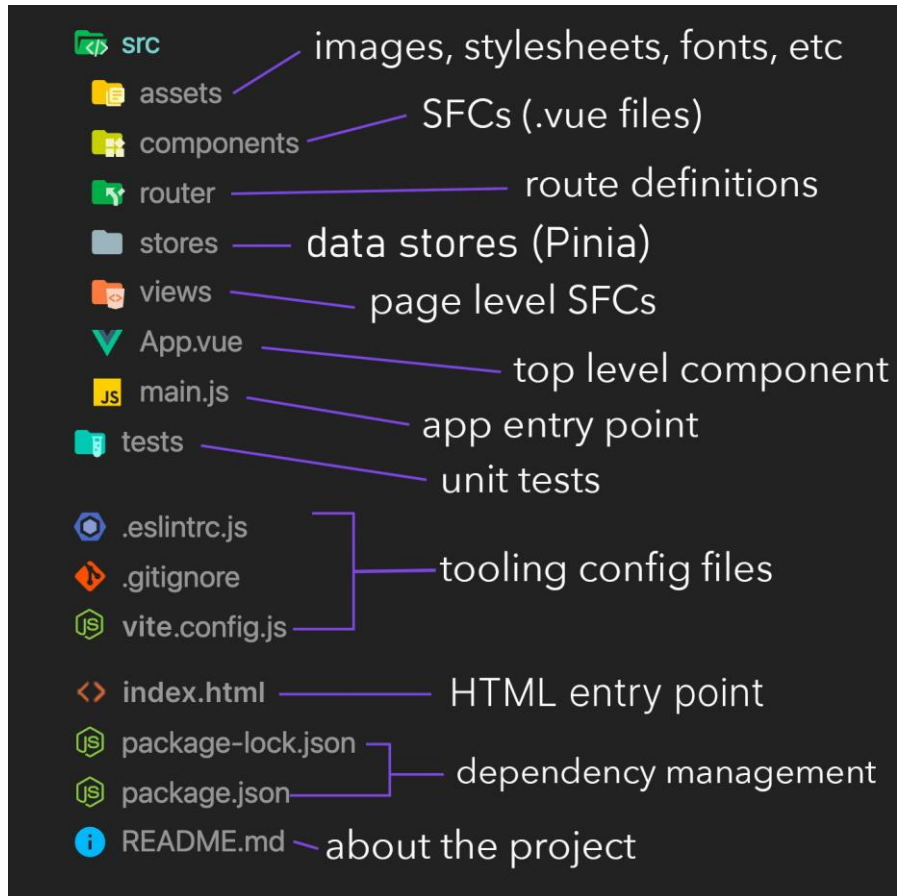


```
~$ npm create vue@latest
Vue.js - The Progressive JavaScript Framework
✓ Project name: ... vue-project
✓ Add TypeScript? ... No / Yes
✓ Add JSX Support? ... No / Yes
✓ Add Vue Router for Single Page Application development? ... No / Yes
✓ Add Pinia for state management? ... No / Yes
✓ Add Vitest for Unit Testing? ... No / Yes
✓ Add an End-to-End Testing Solution? > Playwright
✓ Add ESLint for code quality? ... No / Yes
✓ Add Prettier for code formatting? ... No / Yes
✓ Add Vue DevTools 7 extension for debugging? (experimental) ... No / Yes
Scaffolding project in /Users/yordan.ramchev/Projects/vue-project...
Done. Now run:
  cd vue-project
  npm install
  npm run format
  npm run dev
```

- *npm init vue@latest*
- Plus d'options de personnalisation
- Plus complexe
- Plus complète et avancée
- Tient compte des dernières recommandations et migrations de l'écosystème qui tourne autour de Vue tels que l'intégration de [Vite](#) et [Pinia](#)

Architecture des dossiers

Source image wordline



- **src**: dossier source contenant tous les fichiers *js*, *vue* et *assets*
- **assets**: images, css, fonts, pdf, etc
- **package.json**: dépendances du projet et scripts à exécuter
- **stores**: dédiée à l'implémentation de [Vue-x](#) (ou [Pinia](#)) qu'on verra plus tard
- **components**: tous les composants réutilisables de l'application

EXERCICE





Exercice 2 : application avec Vue-CLI et npm

0-exercices/ex2.md

LES PROPRIÉTÉS



Propriété : data

[Source image vuejs.org](https://vuejs.org)

```
export default {  
  data() {  
    return {  
      a: 1,  
      b: 2,  
      c: {  
        d: 4  
      },  
      e: 5,  
      f: 6  
    }  
  },  
}
```

- Fonction qui retourne un objet littéral contenant toutes les données rattachées au fichier vue
- Peut prendre en valeur tous les types
 - *Nombres*
 - *Chaine de caractère*
 - *Tableaux*
 - *Objets*
 - *booléens*

Propriété : computed

[Source image vuejs.org](https://source.vuejs.org)

```
export default {  
  data() {  
    return { a: 1 }  
  },  
  computed: {  
    // readonly  
    aDouble() {  
      return this.a * 2  
    },  
    // writable  
    aPlus: {  
      get() {  
        return this.a + 1  
      },  
      set(v) {  
        this.a = v - 1  
      }  
    }  
  }  
},
```

- Pour effectuer des transformations ou calculs à partir des données initialisées dans `data()`, uniquement destinées à la couche de présentation (la vue)
- Les données calculées sont **mises en cache** (optimisation)
- *computed()*, contient une ou plusieurs fonctions qui retournent obligatoirement une valeur après avoir effectué vos transformations.
- Attention, les données dans **computed** ne doivent pas altérer les données d'origines dans `data`, l'utilisation du *spread operator ...* est une solution pour faire une copie à partir des données d'origines sans modifier celles-ci
- Exemples d'utilisation : trie, filtre, etc.

Propriété : watch

[Source image vuejs.org](https://vuejs.org)

```
export default {
  data() {
    return {
      a: 1,
      b: 2,
      c: {
        d: 4
      },
      e: 5,
      f: 6
    }
  },
  watch: {
    // watching top-level property
    a(val, oldVal) {
      console.log(`new: ${val}, old: ${oldVal}`)
    },
    // string method name
    b: 'someMethod',
    // the callback will be called whenever any of the watched object properties changes
    c: {
      handler(val, oldVal) {
        console.log('c changed')
      },
      deep: true
    },
    // watching a single nested property:
    'c.d': function (val, oldVal) {
      // do something
    },
    // the callback will be called immediately after the start of the observation
    e: {
      handler(val, oldVal) {
        console.log('e changed')
      },
      immediate: true
    },
    // you can pass array of callbacks, they will be called one-by-one
    f: [
      'handle1',
      function handle2(val, oldVal) {
        console.log('handle2 triggered')
      },
      {
        handler: function handle3(val, oldVal) {
          console.log('handle3 triggered')
        }
      }
    ],
    /* ... */
  ]
}
```

- Permet d'observer les changements effectués sur les données réactives dans le but de rajouter de la logique supplémentaire.
Autrement dit, la logique présente dans un watcher sera exécuté à chaque fois que la valeur observée sera modifiée.
- Donne la possibilité d'accéder à l'ancienne valeur

Interpolation des données

[Source image vuejs.org](https://vuejs.org)

```
template
{{ number + 1 }}

{{ ok ? 'YES' : 'NO' }}

{{ message.split('').reverse().join('') }}

<div :id="`list-${id}`"></div>
```

- Affichage des valeurs d'une variable JS à l'aide de l'opérateur « doubles moustaches » ou en anglais *double binding* `{{ }}`

LES DIRECTIVES



[Source image vuejs.org](https://vuejs.org)

```
<!-- bind an attribute -->


<!-- dynamic attribute name -->
<button v-bind:[key]="value"></button>

<!-- shorthand -->


<!-- shorthand dynamic attribute name -->
<button :[key]="value"></button>

<!-- with inline string concatenation -->


<!-- class binding -->
<div :class="{ red: isRed }"></div>
<div :class="[classA, classB]"></div>
<div :class="[classA, { classB: isB, classC: isC }]"></div>

<!-- style binding -->
<div :style="{ fontSize: size + 'px' }"></div>
<div :style="[styleObjectA, styleObjectB]"></div>

<!-- binding an object of attributes -->
<div v-bind="{ id: someProp, 'other-attr': otherProp }"></div>
```

- La directive **v-bind** permet d'effectuer un *binding* d'un attribut HTML avec une variable JS
- Notation raccourcie de *v-bind* est :
- Autrement dit, les directives rendent les attributs HTML dynamiques et réactives

Directive : v-if

[Source image vuejs.org](https://vuejs.org)

```
<div v-if="type === 'A'">
  A
</div>
<div v-else-if="type === 'B'">
  B
</div>
<div v-else-if="type === 'C'">
  C
</div>
<div v-else>
  Not A/B/C
</div>
```

template

- Crée et affiche un élément et ses enfants en fonction de l'évaluation d'une condition à *true* ou *false*
- Évaluation de la condition à
 - *True* : élément ajouté et affiché au *DOM*
 - *False* : élément n'existe pas dans le *DOM*
- Autres directives conditionnelles
 - *v-else-if* (forcément précédée d'un *v-if*)
 - *v-else* (*idem* précédée d'un *v-if*)

Directive : v-show

[Source image vuejs.org](https://vuejs.org)

```
<h1 v-show="ok">Hello!</h1>
```

template

- Similaire *v-if* sauf que l'élément existe dans le *DOM* peu importe que la condition soit vraie ou fausse
- L'évaluation de la condition modifie la propriété *CSS display*
- Evaluation de la condition à
 - *True* : affiche l'élément en appliquant une valeur différente de *none* à la propriété *CSS display*
 - *False* : cache l'élément, avec un *display* à *none*

Directive : v-model

[Source image vuejs.org](https://vuejs.org)

```
<p>Message is: {{ message }}</p>  
<input v-model="message" placeholder="edit me" />
```

template

Message is: Glodie Tshimini

- Créer une **liaison bidirectionnelle** (écriture et affichage)
- Permet de lier une entrée depuis un champ du formulaire à une affichage dynamique de la donnée saisie ailleurs
- Utilisation du modificateur *lazy* pour empêcher la mise à jour de l'affichage à chaque saisie. L'affichage est alors effectuée lors de la perte du focus sur le champ
- Autres modificateurs
 - *trim* : suppression des espaces
 - *number* : changement de typage en nombre
- Il est possible de *combiner plusieurs modifieurs* à la suite, attention l'ordre aura une importance

Directive : v-for

[Source image vuejs.org](https://vuejs.org)

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

template

```
<div v-for="(item, index) in items"></div>
<div v-for="(value, key) in object"></div>
<div v-for="(value, name, index) in object"></div>
```

template

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

template

- Comme une boucle classique, il permet de répéter un élément HTML N fois
- A utiliser pour le rendu des éléments d'un tableau
- Il est obligatoire de mentionner l'attribut *:key* avec une valeur unique pour élément de la boucle afin de les dissocier

Directive : v-on

[Source image vuejs.org](https://vuejs.org)

```
<!-- method handler -->
<button v-on:click="doThis"></button>

<!-- dynamic event -->
<button v-on:[event]="doThis"></button>

<!-- inline statement -->
<button v-on:click="doThat('hello', $event)"></button>

<!-- shorthand -->
<button @click="doThis"></button>

<!-- shorthand dynamic event -->
<button @[event]="doThis"></button>

<!-- stop propagation -->
<button @click.stop="doThis"></button>

<!-- prevent default -->
<button @click.prevent="doThis"></button>

<!-- prevent default without expression -->
<form @submit.prevent></form>

<!-- chain modifiers -->
<button @click.stop.prevent="doThis"></button>

<!-- key modifier using keyAlias -->
<input @keyup.enter="onEnter" />

<!-- the click event will be triggered at most once -->
<button v-on:click.once="doThis"></button>

<!-- object syntax -->
<button v-on="{ mousedown: doThis, mouseup: doThat }"></button>
```

- Ajouter les événements directement dans l'élément comme un attribut
- *v-on:click*
- *v-on:keyup*
- Etc.
- Notation courte de *v-on* est *@*

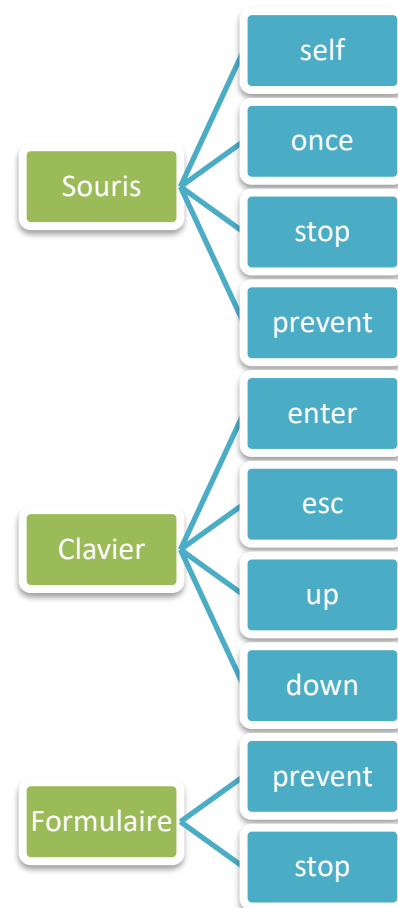
Modificateurs des événements

[Source image vuejs.org](https://sourceimage.vuejs.org)

- Modifiers:

- `.stop` - **call** `event.stopPropagation()` .
- `.prevent` - **call** `event.preventDefault()` .
- `.capture` - **add event listener in capture mode.**
- `.self` - **only trigger handler if event was dispatched from this element.**
- `.{keyAlias}` - **only trigger handler on certain keys.**
- `.once` - **trigger handler at most once.**
- `.left` - **only trigger handler for left button mouse events.**
- `.right` - **only trigger handler for right button mouse events.**
- `.middle` - **only trigger handler for middle button mouse events.**
- `.passive` - **attaches a DOM event with** `{ passive: true }` .

Quelques modificateurs événements



LES MÉTHODES



Les méthodes

```
export default {  
  data () {  
    return { ...  
  }  
},  
methods: {  
  getResults() { ...  
  },  
  displayResults(rates) { ...  
  },  
  clearVal() { ...  
  }  
}
```

- Des fonctions contenant de la logique associé à *l'instance* de *Vue*
- Une méthode peut être appelée au niveau d'une *directive*, *hook*, une autre méthode, etc.
- La notation *ES6* `fonctionName()` `{}` est recommandée
- Avec la notation fléchée, *fonctionName* : `() => {}`, le contexte d'exécution et la valeur de *this* diffère selon ce dernier

EXERCICE





Exercice 3 : les directives

0-exercices/ex3.md

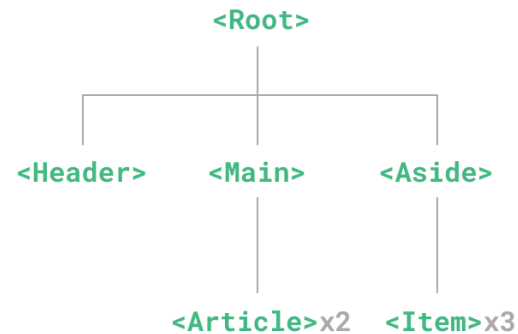
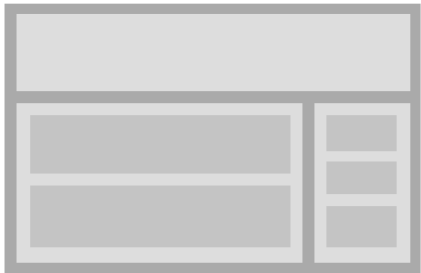
II. LES COMPOSANTS



SINGLE FILE COMPONENTS



[Source image vuejs.org](https://vuejs.org)



- Permet de séparer l'interface utilisateur (UI) en plusieurs éléments indépendants et réutilisables
- Développement **modulaire**
- Permet de faire des mises à jour partiel du DOM en rechargeant uniquement le composant dont les données ont été modifiées
- Dans Vue, un composant regroupe dans un seul fichier avec l'extension **.vue** :
 - Le HTML
 - CSS
 - JavaScript
- **Tout-en-un**, on parle de SFC (Single File Components)

Single File Components (SFC)

[Source image vuejs.org](https://vuejs.org)

```
<script>
export default {
  data() {
    return {
      count: 0
    }
  }
}
</script>

<template>
  <button @click="count++">Count is: {{ count }}</button>
</template>

<style scoped>
button {
  font-weight: bold;
}
</style>
```

Le composant est constitué

1. Balise `<template></template>`
 - Partie dédiée au *HTML* et aux composants enfants
 - Le contenu HTML du composant doit absolument être encadré par un élément racine (un *wrapper*, une balise sémantique *HTML*, une *div* par exemple)
2. Balise `<script></script>`
 - Partie dédiée à la logique *JavaScript* et implémentation de *Vue*
3. Balise `<style></style>`
 - Partie réservée au *CSS*

Intégrer un composant enfant dans un composant parent

[Source image vuejs.org](https://vuejs.org)

```
<script>
import ButtonCounter from './ButtonCounter.vue'

export default {
  components: {
    ButtonCounter
  }
}
</script>

<template>
  <h1>Here is a child component!</h1>
  <ButtonCounter />
</template>
```

1. Importez le composant enfant entre les balises `<script></script>`
 2. Ajoutez le composant enfant dans *components* toujours entre les balises `<script></script>`
 3. Utilisez dans la balise `<template></template>` du parent, le composant importé
- Le nom d'un composant doit respecter la convention de nommage *PascalCase* ou *kebab-case*. et être constitué de plusieurs mots
 - On peut utiliser dans le *template* parent, autant de composant enfant que l'on souhaite
 - Chaque composant est une nouvelle instance et possède son propre contexte d'exécution

EXERCICES



Exercice 4 : refactorisation d'une application avec vue JS

0-exercices/ex4.md



Exercice 5 : refactoring avec plusieurs composants réutilisables

0-exercices/ex5.md

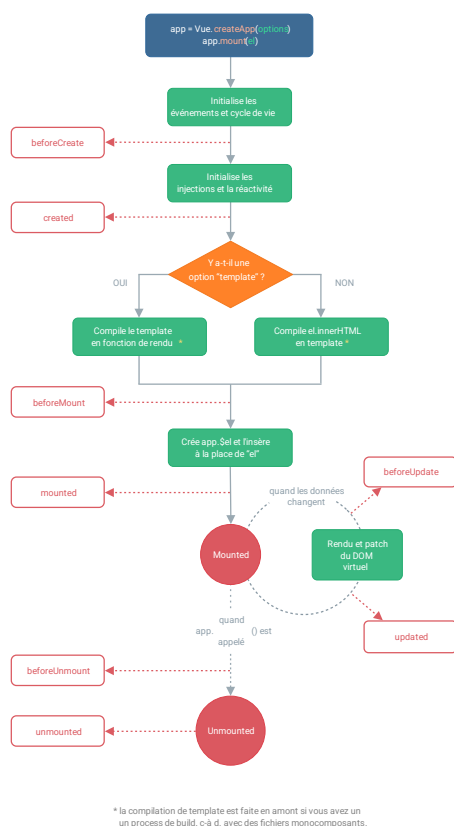
CYCLE DE VIE D'UN COMPOSANT

Inspiré du cours Vue JS de Bruno Tavernier



Cycle de vie d'un composant avec l'options API

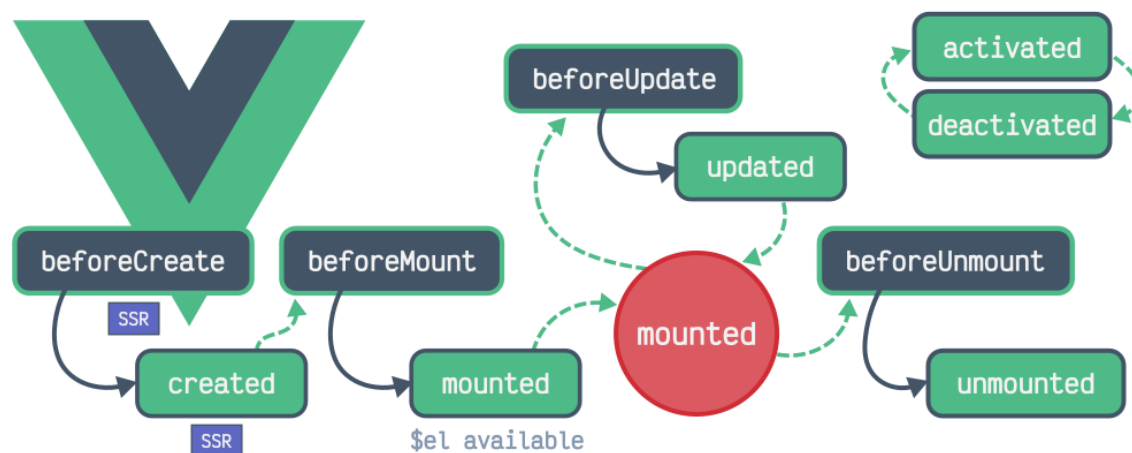
Source image worldline



- Un composant est **créé**, puis **monté** et **supprimé** en fonction des interactions de l'utilisateur avec l'application
- Dès qu'il y a une modification des données (data), Vue recharge implicitement et automatiquement la vue (le DOM) de manière asynchrone
- Toutes les modifications sont ajoutées dans une boucle d'événement
 - Les doublons sont supprimés (une même modification effectuée plusieurs fois)
- **Cycle de vie repose sur le design pattern Observer**
 1. **Plusieurs observateurs** souscrivent auprès d'un **sujet** pour une tâche
 2. Lorsque la tâche est exécutée, le **sujet** **notifie tous les abonnés**
 3. Une fois notifié, les abonnés effectuent une action

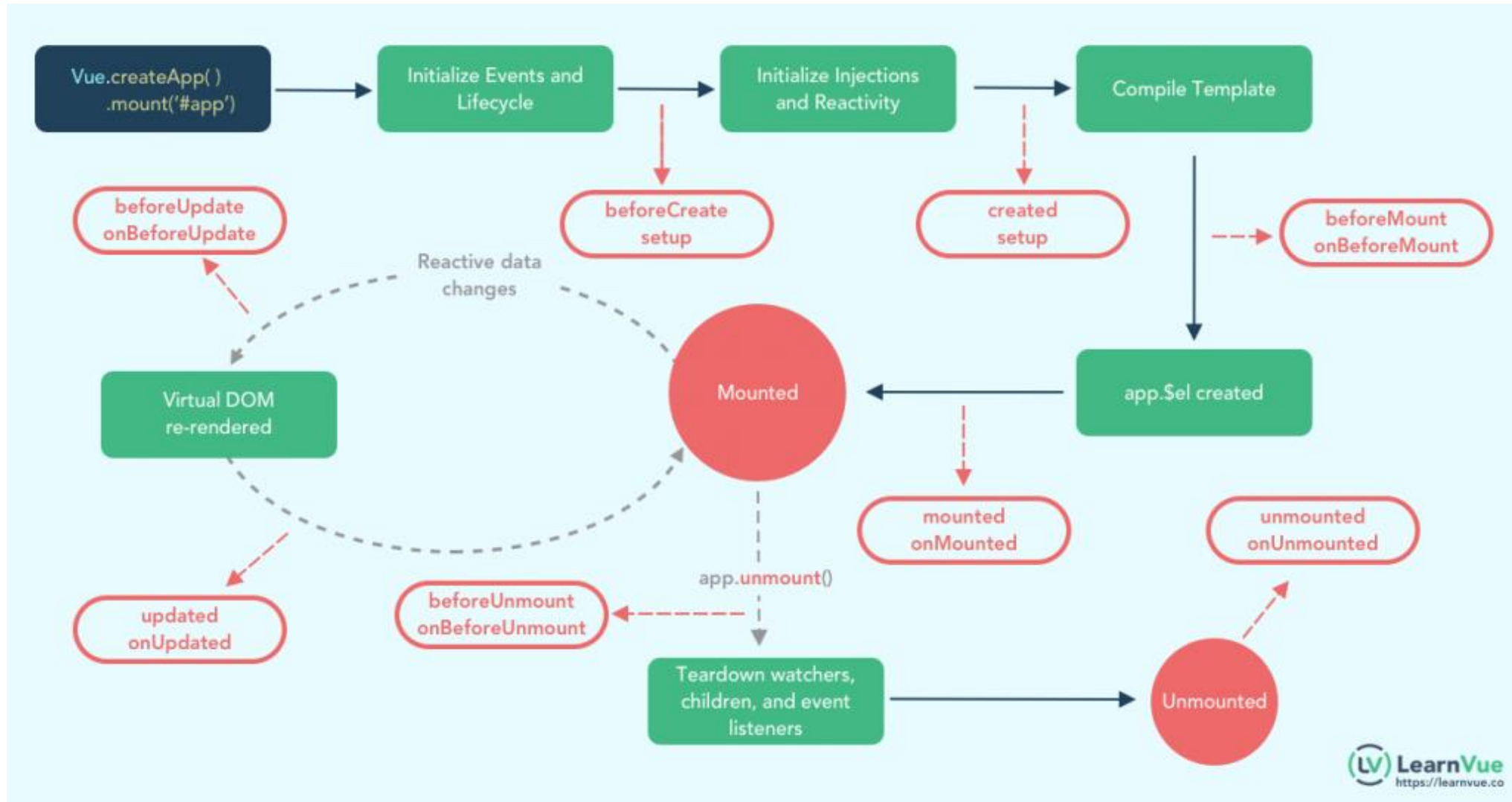
Hooks du cycle de vie d'un composant avec Options API

Source image langagejs



- Chaque *hook* du cycle de vie peut être utilisé en tant que méthode dans l'instance de *Vue* pour ajouter des comportements spécifiques (surcharge) lors du déclenchement de ce *hook*
- Les *hooks* et méthodes
 - *beforeCreate()* , *created()*
 - *beforeMount()* , *mounted()*
 - *beforeUpdate()*, *updated()*
 - *beforeUnmount()*, *unmounted()*

Cycle de vie avec la composition API ([source image learnvue](https://learnvue.co))



Caractéristiques

- Les premiers à s'exécuter dans le cycle de vie des composants vue
- Réaliser les actions avant l'ajout du composant au DOM
- S'exécute lorsque le rendu est effectué côté serveur
- Le composant n'existe pas encore dans le DOM donc l'accès à `this.$el` n'est pas possible

Les *hooks*

- *beforeCreate*
 - Initialise le composant
- *Created*
 - Permet d'accéder aux **data** et **events** actifs
 - Les appels vers des *API* peuvent s'effectuer sur ce hook

Hooks de compilation

Caractéristiques

- Permettent d'accéder au composant avant ou après son ajout au DOM
- Permettent de modifier le DOM initial
- A utiliser pour intégrer des bibliothèques hors Vue

Les hooks

- **beforeMount**
 - s'exécute juste avant le rendu du DOM initial et la compilation des fonctions du modèle
- ***Mounted***
 - accès au composant réactif, modèles et DOM rendu
 - Les appels vers des *API* peuvent également s'effectuer sur ce hook

Hooks de modification

Caractéristiques

- Se déclenche à chaque *re-render* (nouveau rendu du *DOM*) ou à chaque modification d'une propriété réactive
- A utiliser pour les
 - *Watch*
 - *Compute*
 - *Render*
- A utiliser pour le
 - Débogage (corriger des erreurs)
 - Profilage (optimiser les performances)

Les *hooks*

- **beforeUpdate**
 - juste avant la régénération du DOM après une modification d'une données ou du cycle de vie
- ***updated***
 - au moment du re-render

Hooks de destruction

Caractéristiques

- Effectuer des actions lorsque le composant est détruit
- Se déclenche lors de la destruction ou suppression du composant du DOM

Les hooks

- *beforeUnmount*
 - Avant le démontage du composant du DOM
 - A utiliser pour nettoyer les événements ou retirer les abonnements
 - Le composant est encore fonctionnel
- *Unmounted*
 - Lors de la destruction totale du composant
 - A utiliser pour du nettoyage au dernier moment
 - A utiliser pour signaler au back-end la destruction du composant



Propriétés globales accessibles depuis un composant avec l'Options API

- *\$attrs* : attributs de la portée parente, se manipule avec v-bind
- *\$parent* : instance parente de l'instance courante
- *\$ref* : attribut d'un élément du DOM, équivalent à [document.querySelector\(\)](#) par exemple)
- *\$refs* : liste des éléments du DOM ayant un attribut ref
- *\$root* : instance racine ou instance courante si celle-ci est la racine
- *\$slots* : liste des slots nommées et non nommés
- Etc.
- Attention, **lisez la documentation** avant d'utiliser ces propriétés, certaines sont ou seront **dépréciées** prochainement.

L'ÉCHANGE DES DONNÉES ENTRE LES COMPOSANTS



Echanger des informations du parent à l'enfant avec les props

[Source image vuejs.org](https://sourceimage.vuejs.org)

```
<!-- BlogPost.vue -->
<script>
export default {
  props: ['title']
}
</script>

<template>
  <h4>{{ title }}</h4>
</template>
```

```
export default {
  // ...
  data() {
    return {
      posts: [
        { id: 1, title: 'My journey with Vue' },
        { id: 2, title: 'Blogging with Vue' },
        { id: 3, title: 'Why Vue is so fun' }
      ]
    }
  }
}
```

```
<BlogPost
  v-for="post in posts"
  :key="post.id"
  :title="post.title"
/>
```

- La transmission des données d'un composant parent à un composant enfant, s'effectue grâce aux *props*
- *Props* = propriétés passées à un composant enfant
- Le composant enfant définit les propriétés qu'il peut recevoir dans un objet *props*
- Lecture seule des *props*, l'enfant ne doit pas modifier les informations provenant du parent. Pour modifier une *props*, il faut créer une variable locale qui copie une *props*

Toujours du parent à l'enfant avec les options des props

[Source image vuejs.org](https://vuejs.org)

```
export default {  
  props: {  
    // type check  
    height: Number,  
    // type check plus other validations  
    age: {  
      type: Number,  
      default: 0,  
      required: true,  
      validator: (value) => {  
        return value >= 0  
      }  
    }  
  }  
}
```

- On peut donner plus de détails en ajoutant aux **props** des options non obligatoires
 - **Type** : domaine de définition de la propriété (*String, Number, Boolean, Object, etc.*)
 - **Required**: valeur obligatoire ou non (valeurs possibles *true* ou *false*)
 - **Default** : valeur par défaut
 - **Validator** : fonction qui permet de vérifier que la donnée est valide en retournant **true** ou **false**

Echanger des informations de l'enfant au parent avec les événements personnalisés

- La transmission des données d'un composant enfant à un composant parent, s'effectue grâce à l'émission d'un **événement personnalisé** de l'enfant au parent avec la donnée en paramètre
 - Depuis le corps d'une méthode dédiée *this.\$emit('my-event', param)*
 - Directement au niveau de la balise `<template></template>` du composant enfant avec la notation *\$emit('my-event', param)* depuis un événement
- Le composant parent intercepte l'événement et récupère un objet en paramètre souvent nommé payload *@my-event='myFunction'*

Emission et réception d'un événement personnalisé

[Source image vuejs.org](https://source.vuejs.org)

```
<script>
import BlogPost from './BlogPost.vue'
export default {
  components: {BlogPost},
  data() {
    return {
      posts: [
        { id: 1, title: 'My journey with Vue' },
        { id: 2, title: 'Blogging with Vue' },
        { id: 3, title: 'Why Vue is so fun' }
      ],
      postFontSize: 1
    }
  }
}
</script>
<template>
  <div :style="{ fontSize: postFontSize + 'em' }">
    <BlogPost
      v-for="post in posts"
      :key="post.id"
      :title="post.title"
      @enlarge-text="postFontSize += 0.1"
    ></BlogPost>
  </div>
</template>
```

```
<script>
export default {
  props: ['title'],
  emits: ['enlarge-text']
}
</script>

<template>
  <div class="blog-post">
    <h4>{{ title }}</h4>
    <button @click="$emit('enlarge-text')">Enlarge text</button>
  </div>
</template>
```

EXERCICE





Exercice 6 : échanges des informations entre les composants

0-exercices/ex6.md

LES SLOTS



[Source image vuejs.org](https://source.vuejs.org)

Docs ▾ API Playground Ecosystem ▾ About ▾ Sponsor

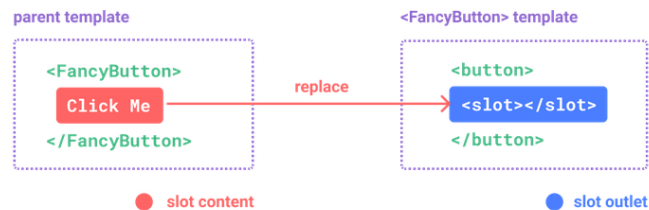
```
<FancyButton>
  Click me! <!-- slot content -->
</FancyButton>
```

template

The template of `<FancyButton>` looks like this:

```
<button class="fancy-btn">
  <slot></slot> <!-- slot outlet -->
</button>
```

template



And the final rendered DOM:

```
<button class="fancy-btn">Click me!</button>
```

html

- Lorsqu'un parent doit transmettre plusieurs informations à un composant enfant, il est préférable de passer par des *slots* qui sont moins complexes et plus faciles à maintenir
- La balise `<slot>` indique l'emplacement des données qui seront reçues par le composant enfant lors de la transmission des données dynamiquement par le parent
- Idéale lorsqu'on doit transmettre aux composants enfants du *HTML*

Slots : utilisation de l'attribut name

- Lorsqu'il y a plusieurs *slot* dans un composant enfant, l'attribut *name* permet de leur donner un nom unique pour les distinguer. Un *slot* sans attribut *name* a par défaut la valeur "*default*"
- # est la notation courte de v-slot

```
1
<div class="container">
  <header>
    <slot name="header"></slot>
  </header>
  <main>
    <slot></slot>
  </main>
  <footer>
    <slot name="footer"></slot>
  </footer>
</div>
```

```
2
<BaseLayout>
  <template v-slot:header>
    <!-- content for the header slot -->
  </template>
</BaseLayout>
```

```
3
<BaseLayout>
  <template #header>
    <h1>Here might be a page title</h1>
  </template>

  <!-- implicit default slot -->
  <p>A paragraph for the main content.</p>
  <p>And another one.</p>

  <template #footer>
    <p>Here's some contact info</p>
  </template>
</BaseLayout>
```

```
4
<div class="container">
  <header>
    <h1>Here might be a page title</h1>
  </header>
  <main>
    <p>A paragraph for the main content.</p>
    <p>And another one.</p>
  </main>
  <footer>
    <p>Here's some contact info</p>
  </footer>
</div>
```

- 1 : Composant Enfant
- 2 et 3 : Composant Parent
- 4 Rendu final

EXERCICE

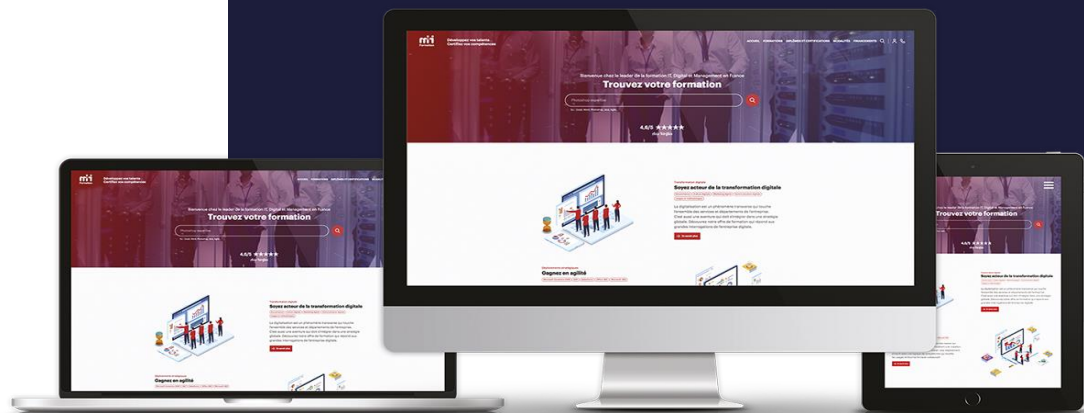




Exercice 7 : mise en place des slots

0-exercices/ex7.md

III. LES NOTIONS AVANCÉES



LA NAVIGATION AVEC VUE ROUTER



Ajout du routeur dans l'application Vue

```
import {createApp} from 'vue'
import {createRouter, createWebHistory} from 'vue-router'
import Home from '@/views/Home'
import About from '@/views/About'
import App from './App.vue'

const router = createRouter({
  history: createWebHistory(),
  routes: [
    {
      path: '/',
      name: 'Home',
      component: Home
    },
    {
      path: '/about',
      name: 'About',
      component: About
    }
  ]
})

createApp(App)
  .use(router)
  .mount('#app')
```

- Installation

1. vue add router

- Permet de mettre en place les routes d'une application

2. Ou via *npm i vue-router@4*

3. Ou via le [CDN](#)

4. Modifiez *main.js* pour mettre en place les routes de votre application

5. Avec npm (vite), lors de la création d'un projet, il est possible de dire oui à l'ajout de vue-router, dans ce cas, il y aura juste à s'occuper de modifier le fichier *src/routes/index.js* pour ajouter les routes

Définition des routes de l'application

- On définit toutes les routes de l'application et généralement dans le fichier `router/index.js`
- Les paramètres d'une route
 - ***path***: chemin
 - ***name***: nom de la route
 - ***component***: composant associé à la route
- La propriété ***linkActiveClass*** permet d'associer une *classe CSS* à la route courante
 - Si on définit pas cette *classe*, par défaut vue gère automatiquement le style de la route active avec la *classe* `router-link-active`

Les modes de navigation

1. HTML5

- Similaire à la navigation classique du web page par page appliqué à un *SPA (Single Page Application)*.
- Recommandé pour le rendu côté client

2. Hash

- Avec les ancres `#`
- Ne nécessite pas de configuration côté serveur car c'est dans un contexte de SPA

3. Memory

- Pour le rendu côté serveur ou les tests des composants

Les routes dynamiques

[Source image vuejs.org](https://vuejs.org)

```
const User = {  
  template: '<div>User</div>',  
}  
  
// these are passed to `createRouter`  
const routes = [  
  // dynamic segments start with a colon  
  { path: '/users/:id', component: User },  
]  
  
const User = {  
  template: '<div>User {{ $route.params.id }}</div>',  
}
```

- Il suffit de préfixer les noms des paramètres dynamiques par **:** dans la valeur de path
- Les valeurs des paramètres sont accessibles à partir de `this.$route.params` pour l'Options API.

S'agissant de la composition API , il faut utiliser [les fonctions useRouter\(\) et useRoute\(\)](#)

Les liens et les vues

```
<template>
  <header>
    <div class="wrapper">
      <nav>
        <RouterLink to="/">Home</RouterLink>
        <RouterLink to="/about">About</RouterLink>
      </nav>
    </div>
  </header>

  <RouterView />
</template>
```

- La balise ***router-link*** (globale, pas besoin d'import)
 - Pour la navigation interne (entre les pages de l'application), remplacez les balises `<a>` par les balises `<router-link>` pour éviter de recharger la page
- La balise ***router-view*** (globale, pas besoin d'import)
 - Emplacement dans le composant parent ou root du contenu des pages à afficher
- Nommez ces balises en ***PascalCase*** ou ***kebab-case***
- Gardez la balise `<a>` pour les liens externes à l'application

Navigation dans le programme avec l'Options API

- Grâce à la fonction *this.\$router.go(nb)*, on peut faire de retour vers les pages précédentes
 - Par exemple, *this.\$router.go(-1)* permet de retourner à la page précédente
 - Ou *this.\$router.back()*
- Avec la fonction *this.\$router.push({name: 'Login'})*, dans le programme, on peut rediriger l'utilisateur vers une autre page.
- La méthode *this.\$router.replace* a le même comportement que la méthode push, cependant, *replace* ne rajoute pas la route dans la pile de navigation. Elle se contente de remplacer uniquement la route courante dans la pile.

Navigation guards

```
import {createApp} from 'vue'
import {createRouter, createWebHistory} from 'vue-router'
import Home from '@views/Home'
import About from '@views/About'
import App from './App.vue'
import pages from '@data/pages.json'
const router = createRouter({
  history: createWebHistory(),
  beforeEnter: (to, from, next) => {
    const isExist = pages.find(page => page.slug === to.params.slug)
    if(!isExist) {
      return {
        name: 'NotFound404'
      }
    }
  },
  routes: [
    {
      path: '/',
      name: 'Home',
      component: Home
    },
    {
      path: '/about',
      name: 'About',
      component: About
    }
  ]
})
createApp(App)
  .use(router)
  .mount('#app')
```

- Middleware pour effectuer des vérifications au niveau de la navigation
- Plusieurs fonctions
 - `beforeEnter()`
 - `beforeEach()`
 - [Documentation de tous les hooks](#)
- Types de vérification courantes
 - Page existe
 - Authentification
 - Redirection vers une autre page
 - Etc.

EXERCICE





Exercice 8 : routing

0-exercices/ex8.md

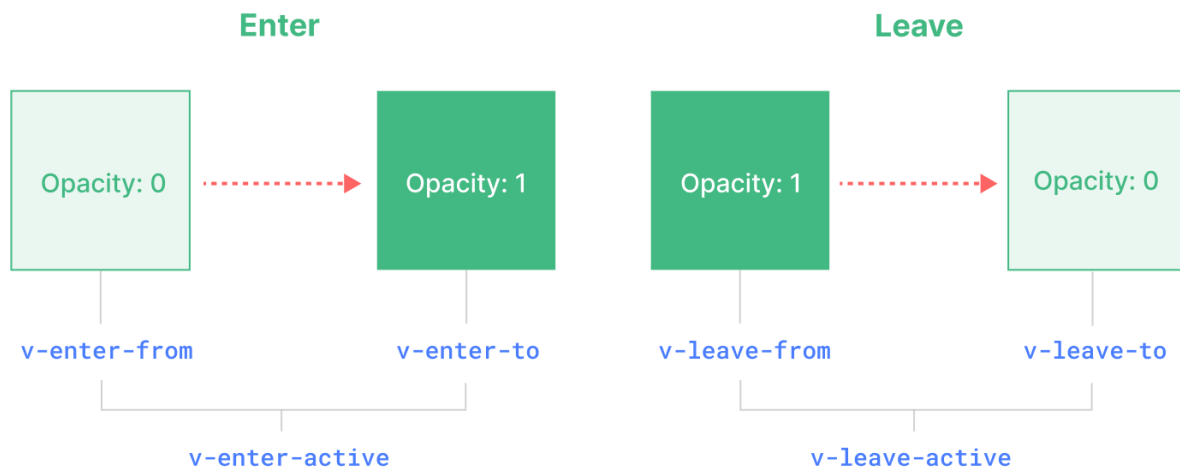
LES TRANSITIONS

Inspiré du cours Vue JS de Bruno Tavernier



Les 6 classes associées aux transitions entrée/sortie

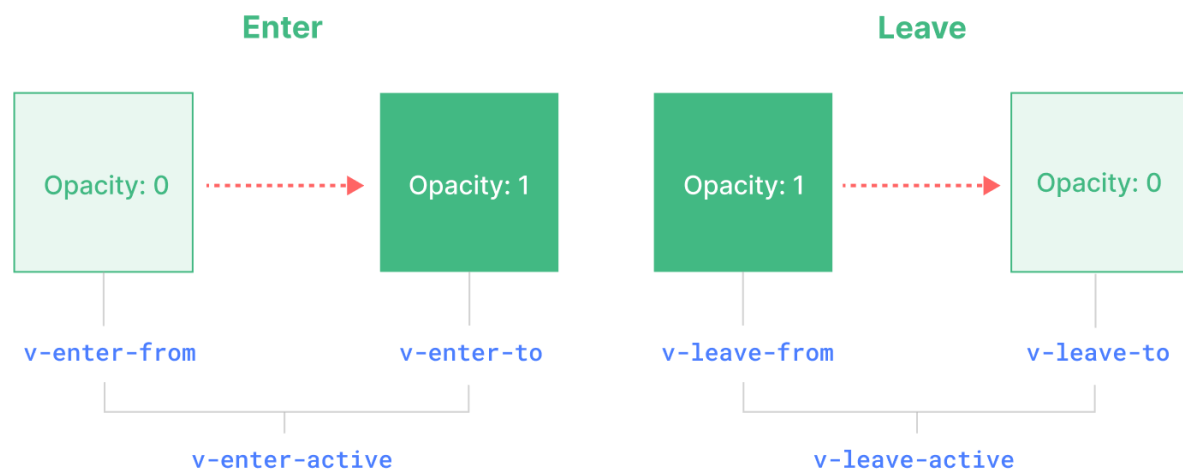
[Source image vuejs.org](https://vuejs.org)



1. ***v-enter-from***: état de départ pour entrer. Ajouté avant l'insertion de l'élément, supprimé après l'insertion de l'élément.
2. ***v-enter-active***: état actif pour entrer. Appliqué pendant toute la phase d'entrée. Ajouté avant l'insertion de l'élément, supprimé à la fin de la transition/animation. Cette classe peut être utilisée pour définir la durée, le retard et la courbe d'accélération de la transition entrante.
3. ***v-enter-to***: Etat de fin pour entrer. Ajout après l'insertion de l'élément (en même temps *v-enter-from* est supprimé), supprimé lorsque la transition/animation se termine.

Les 6 classes associées aux transitions entrée/sortie

[Source image vuejs.org](https://vuejs.org)



4. ***v-leave-from*** : état de départ de la suppression. Ajouté immédiatement lorsqu'une transition de départ est déclenchée, supprimé après une image.
5. ***v-leave-active*** : état actif pour la suppression. Appliqué pendant toute la phase de départ. Ajouté immédiatement lorsqu'une transition quitter est déclenchée, supprimé lorsque la transition/animation se termine. Cette classe peut être utilisée pour définir la durée, le retard et la courbe d'accélération de la transition de départ.
6. ***v-leave-to*** : état de fin de la suppression. Ajout d'une image après le déclenchement d'une transition de départ (en même temps *v-leave-from* est supprimée), supprimée lorsque la transition/animation se termine.

[Source image vuejs.org](https://sourceimage.vuejs.org)

```
<script>
export default {
  data() {return {show: true}}
}
</script>
<template>
  <button @click="show = !show">Toggle</button>
  <Transition name="bounce">
    <p v-if="show" style="margin-top: 20px; text-align: center;">
      Hello here is some bouncy text!
    </p>
  </Transition>
</template>
<style>
.bounce-enter-active {
  animation: bounce-in 0.5s;
}
.bounce-leave-active {
  animation: bounce-in 0.5s reverse;
}
@keyframes bounce-in {
  0% {transform: scale(0);}
  50% {transform: scale(1.25);}
  100% {transform: scale(1);}
}
</style>
```

- Englober avec la balise `<transition></transition>` les éléments qui peuvent subir un changement d'état
 - Modification de la valeur d'une propriété CSS à une autre
 - Directives `v-show/v-if`
 - La balise `router-view`
- La balise transition possède deux *props*
 - ***name***
 - ***mode***: liste fermée du comportement dans le temps de la transition(gestion des éléments entrants et sortants du DOM)
- Lorsqu'une transition est nommée (possède l'attribut *name*), le *prefixer v* doit être remplacé par la valeur donnée à l'attribut *name*
 - Exemple : fade-enter-active au lieu de v-enter-active

EXERCICE





Exercice 9 : transitions et animations

0-exercices/ex9.md

LE STORE AVEC VUEX



[Source image vuejs.org](https://vuejs.org)

Direct Download / CDN

<https://unpkg.com/vuex@4>

Include `vuex` after Vue and it will install itself automatically:

```
<script src="https://unpkg.com/vuex@4"></script>
```

html

NPM

```
npm install vuex@next --save
```

sh

Yarn

```
yarn add vuex@next --save
```

sh

Mise en place dans le projet

```
import { createApp } from 'vue'
import { createStore } from 'vuex'

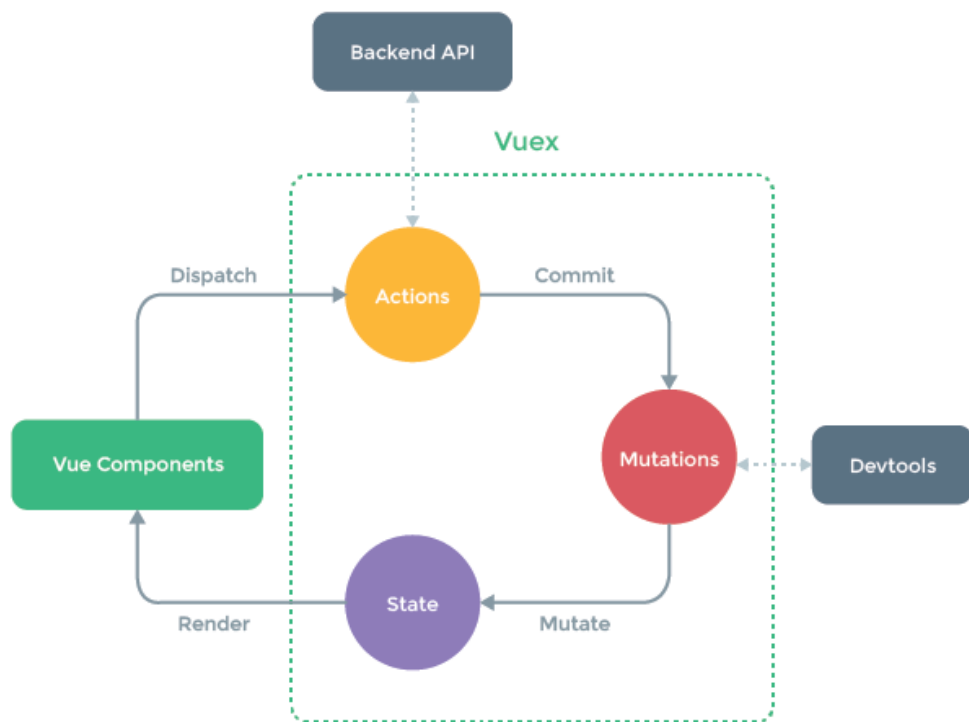
// Create a new store instance.
const store = createStore({
  state () {
    return {
      count: 0
    }
  },
  mutations: {
    increment (state) {
      state.count++
    }
  }
})

const app = createApp({ /* your root component */ })

// Install the store instance as a plugin
app.use(store)
```

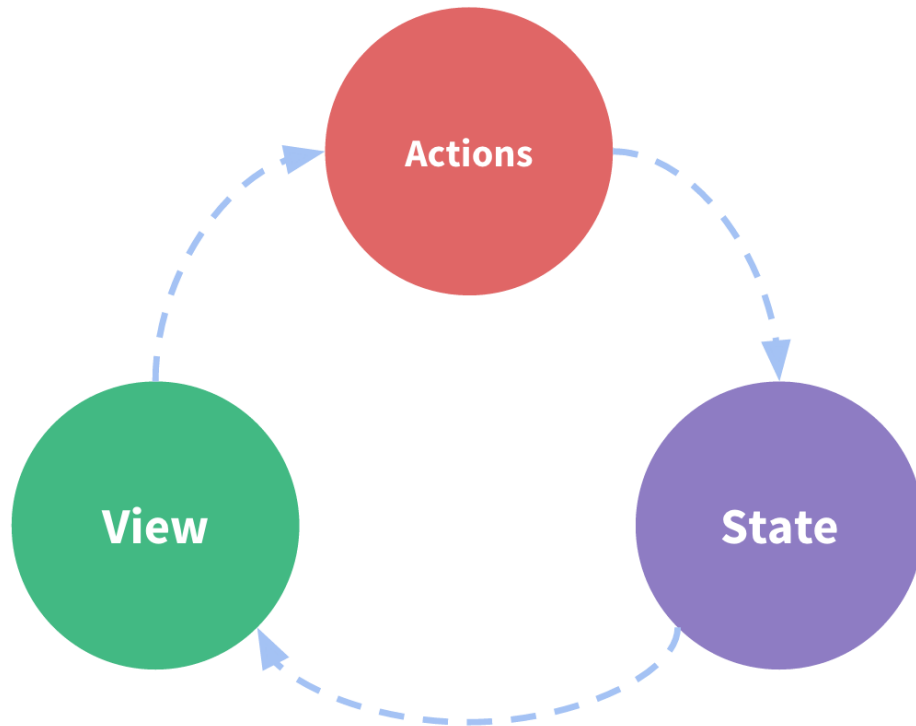

Librairie gestionnaire d'état Vuex

[Source image vuejs.org](https://vuejs.org)



- *State management pattern (gestionnaire d'état)*
- Zone de stockage des données centralisées pour toute l'application
- [Flux Design Pattern](#)
- A utiliser pour des applications ayant un grand nombre de composants qui partagent des données communes.
- Avec [Vuex](#)
 - Les composants se mettent automatiquement à jour lorsque l'état d'une donnée utilisée par le composant est modifié depuis le store.
 - Il y a une traçabilité (historique) des données
- [Pinia](#)
 - *Devenue la librairie officielle* de l'implémentation d'un *store* dans une application Vue
 - Fonctionnement similaire à *Vuex*

[Source image vuejs.org](https://vuejs.org)



- « Zone de stockage commune »
- Composé de
 1. **State**: stockage des données communes (équivalent à data dans un contexte global)
 2. **Mutation**: seul moyen pour muter (modifier) le State
 3. **Action**: réaction aux interactions utilisateurs pouvant déclencher une mutation
 4. **Getters**: retourne les données transformées du state (équivalent de *computed*)

[Source image vuejs.org](https://vuejs.org)

```
const Counter = {
  template: `<div>{{ count }}</div>`,
  computed: {
    count () {
      return this.$store.state.count
    }
  }
}
```

```
// in full builds helpers are exposed as Vuex.mapState
import { mapState } from 'vuex'

export default {
  // ...
  computed: mapState({
    // arrow functions can make the code very succinct!
    count: state => state.count,

    // passing the string value 'count' is same as `state => state.count`
    countAlias: 'count',

    // to access local state with `this`, a normal function must be used
    countPlusLocalState (state) {
      return state.count + this.localCount
    }
  })
}
```

```
computed: mapState([
  // map this.count to store.state.count
  'count'
])
```

- *Objet* unique contenant toutes les données communes partagées par plusieurs composants
- 2 manière d'accéder aux propriétés (données) du *store*.
Par exemple, pour avoir la donnée *user* depuis le *state* commun
 1. *this.\$store.state.user*
 2. Ou faire appel à la méthode *mapState*(*['user']*) de *Vuex*
Cette dernière méthode est recommandée et facile à implémenter pour l'Options API
- Pour la composition API, il faut utiliser la méthode [useStore\(\)](#)

[Source image vuejs.org](https://vuejs.org)

```
const store = createStore({
  state: {
    todos: [
      { id: 1, text: '...', done: true },
      { id: 2, text: '...', done: false }
    ]
  },
  getters: {
    doneTodos (state) {
      return state.todos.filter(todo => todo.done)
    }
  }
})
```

```
import { mapGetters } from 'vuex'

export default {
  // ...
  computed: {
    // mix the getters into computed with object spread operator
    ...mapGetters([
      'doneTodosCount',
      'anotherGetter',
      // ...
    ])
  }
}
```

If you want to map a getter to a different name, use an object:

```
...mapGetters({
  // map `this.doneCount` to `this.$store.getters.doneTodosCount`
  doneCount: 'doneTodosCount'
})
```

- Permet
 - Retourner directement les données du state
 - Faire des transformations et retourner les données calculées du state
- Obligatoirement en paramètre le *state* et éventuellement (optionnel) les autres getters
- Retourne obligatoirement une valeur
- Exemples de transformation
 - Trier les éléments d'un tableau
 - Filtrer les éléments d'un tableau
 - Etc.

[Source image vuejs.org](https://vuejs.org)

```
const store = createStore({  
  state: {  
    count: 1  
  },  
  mutations: {  
    increment (state) {  
      // mutate state  
      state.count++  
    }  
  }  
})
```

```
// ...  
mutations: {  
  increment (state, payload) {  
    state.count += payload.amount  
  }  
}
```

- Seul endroit autorisé pour modifier le state faisant suite à un commit depuis une action
- Responsabilité uniquement dédiée à la modification du state
- Pas d'opérations complexes ou appels vers des APIs externes à cet endroit
- Une mutation est une fonction qui prend 2 paramètres
 1. Le premier et obligatoire est le *state*
 2. Le deuxième et optionnel est le *payload* (objet)
- La modification du *state* est *synchrone*

[Source image vuejs.org](https://sourceimage.vuejs.org)

```
const store = createStore({  
  state: {  
    count: 0  
  },  
  mutations: {  
    increment (state) {  
      state.count++  
    }  
  },  
  actions: {  
    increment (context) {  
      context.commit('increment')  
    }  
  }  
})
```

```
actions: {  
  increment ({ commit }) {  
    commit('increment')  
  }  
}
```

Dispatching Actions

Actions are triggered with the `store.dispatch` method:

```
store.dispatch('increment')
```

- Similaires aux méthodes
- L'endroit dédié à la logique et aux appels vers des API externes
- Une action prend en paramètre :
 - En premier, obligatoirement, une méthode *commit*
 - En second, un objet *payload*
- Une action déclenche au moins une mutation grâce à la méthode *commit('nameOfMutation')*
- Ne jamais modifier le *state* depuis une *action*
- L'appel d'une action par la méthode *dispatch()* est asynchrone

Appel d'une action depuis un composant

```
import { mapActions } from 'vuex'

export default {
  // ...
  methods: {
    ...mapActions([
      'increment', // map `this.increment()` to `this.$store.dispatch('increment')`

      // `mapActions` also supports payloads:
      'incrementBy' // map `this.incrementBy(amount)` to `this.$store.dispatch('incrementBy', amount)`
    ]),
    ...mapActions({
      add: 'increment' // map `this.add()` to `this.$store.dispatch('increment')`
    })
  }
}
```

EXERCICE





Exercice 10 : mise en place d'un data store

0-exercices/ex10.md

VueX plugins



Principe du plugin qui utilise Vuex

[Source image solomon eseme
logrocket.com](https://sourceimage.solomon.eseme.rocket.com)

```
import axios from "axios";
const url = // Your Slack Webhook here;
export default (store) => {
  store.subscribe((mutation) => {
    if (mutation.type === "STORE_ERRORS") {
      // Alert Slack here
      errorSlackAlert(mutation.payload);
    }
  });
  store.subscribeAction((action) => {
    // Alert Slack here
    vuexActionSlackAlert(action.type);
  });
};
```

- Une fonction qui prend en paramètre uniquement l'objet *store*
- Cette fonction est déclenché lorsque certaines mutations ou actions sont appelées par *Vuex*
- 2 fonctions de l'objet Store
 - *Store.subscribe(mutation)*: s'abonner à une mutation, la propriété *mutation.type* permet d'avoir le nom de la mutation
 - *Store.subscribeAction(action)*

Approfondir Vue JS

- [Composition API](#)
- Les tests unitaires avec [Vitest](#)
- [TypeScript](#)
- Framework [Nuxt.js](#) basé sur Vue JS
- [Formation M2I fonctionnalités avancées](#)


Émargements


Pensez à vérifier les feuilles d'émargement

Chao Manu

LEVI STRAUSS Claude

HULOT Nicolas

A large rectangular box containing a handwritten signature in black ink that reads 'Levi'. The signature is written in a cursive style.

A small blue icon of an eraser with a pencil, located in the bottom right corner of the signature box.

ANNULER

ENREGISTRER

Évaluations formateur

N'oubliez pas les évaluations formateur avant de partir.





MERCI

Glodie Tshimini

contact@tshimini.fr



m2iinformation.fr