

VUE.JS

LES FONDAMENTAUX



2024

By Boris SAUVAGE

Sommaire 1/3

- Rappels sur Javascript
- Histoire, Equipe, Financements et Popularité
- Installation

Atelier de mise en pratique (Premier programme)

- CLI Create-Vue
- Le linter
- Le composant
- Le CSS et sa portée
- Le cycle de vie du composant

Atelier de mise en pratique (Mise en place fil rouge)

Sommaire 2/3

- Les templates
- Les directives

Atelier de mise en pratique (Afficher une liste de catégories)

- Propriété calculées
- Références réactives

Atelier de mise en pratique (Afficher tutoriels et commentaires)

- Composants et communication

Atelier de mise en pratique (hiérarchie de composants)

- Le routeur

Atelier de mise en pratique (navigation)

Sommaire 3/3

- Le protocole HTTP et API Rest
- Les services

Atelier de mise en pratique (Créer des services)

- Les formulaires

Atelier ensemble (formulaire et appel réseau)

Atelier de mise en pratique (Créer un formulaire d'ajout)

- La validation des formulaires

Atelier de mise en pratique (Valider le formulaire d'ajout)

- Les tests
- Les composables

VUE.JS

Rappels Javascript



Javascript ESNext

- ES6 (ECMAScript 2015) est une importante mise à jour du langage JavaScript
- Il apporte de nouvelles fonctionnalités et améliorations de syntaxe
 - pour rendre le code plus lisible, concis et expressif
- Les mots-clés **let, const**
- Les classes
- Les fonctions fléchées
- Les Littéraux de gabarits (template strings)
- Déstructuration
- Les promesses avec **async et await (ES8)**
- Les modules

Les mots-clés **let**, **const** 1/2

- Il est possible de définir une variable avec la portée locale à un bloc
- Le mot clef **let** permet de déclarer une variable visible (uniquement dans le bloc courant)

let maVariable = 1;
- Quelle est la différence entre **let** et **var** ?
 - La portée (le scope) de la variable
 - Avec **let**, la variable est locale, elle est donc accessible uniquement dans le bloc où elle a été déclarée
 - Avec **var**, la déclaration de la variable est remontée au début du code et sa portée est globale

Les mots-clés `let`, `const` 2/2

- Permet de figer la référence
- Une erreur sera produite si on tente de modifier `maConstante` ci-dessous :

```
const maConstante = 42;
```

- Par convention on peut l'écrire en majuscules, cela permet de les repérer rapidement

```
const FIRSTNAME = 'Boris';
```

Les classes

- JavaScript permet d'utiliser les classes
 - syntaxe plus simple pour des objets

```
class Cat {  
  
    constructor(name, age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    meow() {  
        console.log("Miaouu");  
    }  
}  
  
let myCat = new Cat("Réglinette", 2);  
myCat.meow();      // Miaouu
```

Les fonctions fléchées 1/2

- Il existe de nouvelles manières d'écrire les fonctions
- La nouvelle syntaxe se rapproche de la syntaxe du C#, Java 8 ou coffeescript

```
// ES5 (2009)
```

```
var addition = function (x) {  
    return x + 1;  
}
```

```
// ES6 (2015)
```

```
let addition = (x) => {  
    return x + 1;  
}
```

// Avec un seul argument, pas besoin de parenthèses

```
let addition = x => {  
    return x + 1;  
}
```

// Avec une seule instruction, pas besoin d'accolades et du return

```
let addition = x => x + 1;
```

Les fonctions fléchées 2/2

- Les fonctions fléchées utilisent un `this` lexical
 - la fonction n'aura pas son `this`
 - elle utilisera le `this` du scope parent
- Les fonctions fléchées ne sont pas totalement identiques aux fonctions classiques
 - Elles ne changent pas les valeurs `this`, arguments et `super`
 - Ces valeurs correspondent à celles de l'objet englobant (scope parent)

```
const objet = {};
objet.m1 = function () {
    return this
};
objet.m2 = () => this;
window.console.log(objet.m1()); // affiche l'objet

window.console.log(objet.m2()); // affiche window
```

Les Littéraux de gabarits (Template strings)

- Les chaînes de caractères peuvent être contenues :
 - entre des apostrophes : 'Example'
 - entre des guillemets doubles : "Example"
- Avec ES2016, les chaînes de caractères peuvent être contenues
 - entre des **backquotes** ``
- Plus besoin de concaténer avec des + les chaînes de caractères

```
// Avant
var hello = function (name) {
    return "Your welcome " + name;
}
hello("Jean");
```

```
// Avec ES6
let hello = (name) => {
    return `Your welcome ${name}`;
}
hello("Jean");
```

Déstructuration (Object Destructuring) 1/2

- Javascript permet de décomposer certaines propriétés d'un objet

```
let person = {  
    firstName: 'John',  
    lastName: 'Doe'  
};
```

```
let firstName = person.firstName;  
let lastName = person.lastName;
```



```
let { firstName: fname, lastName: lname } = person;  
console.log(fname);  
console.log(lname);
```

Déstructuration (Object Destructuring) 2/2

- Javascript permet de décomposer certaines propriétés d'un objet
 - Fonctionne aussi avec des paramètres de fonctions

```
let person = {  
  firstName: 'John',  
  lastName: 'Doe'  
};
```

```
sayHello(person);
```

```
const sayHello = (person) => {  
  console.log(`Hey ${person.firstName} ${person.lastName}`);  
}
```



```
const sayHello = ( {firstName, lastName} ) => {  
  console.log(`Hey ${firstName} ${lastName}`);  
}
```

Les promesses avec **async** et **await** (ES8)

- **async** et **await** sont des fonctionnalités introduites dans ES8 (ECMAScript 2017)
- Permettent de simplifier et rendre plus lisible le traitement des promesses
 - Elles permettent d'écrire du code asynchrone de manière synchrone, en utilisant des promesses

```
async function fetchData() {  
    try {  
        const data = await fetchDataFromAPI(); // Await permet d'attendre la résolution de la promesse  
        console.log(data);  
    } catch (error) {  
        console.error(error);  
    }  
}  
  
fetchData();
```

Les modules ES6 – Introduction 1/5

- Fournissent un moyen standard de modéliser/d'organiser le code en plusieurs fichiers
 - moyen efficace de partager du code entre différentes parties d'une application
- Les modules ES6 permettent d'organiser le code de manière modulaire et réutilisable
- À partir d'un module, il est possible d'exporter :
 - une seule valeur, fonction ou classe
 - plusieurs valeurs, fonctions ou classes
- Il est possible de renommer les éléments importés lors de l'importation
 - Utile pour éviter les conflits de noms

Les modules ES6 – Export par default 2/5

(**export default**)

- Permet d'exporter une seule valeur, fonction ou classe par défaut
- Il n'est pas nécessaire d'utiliser des accolades lors de l'importation

```
//person.js

const person = {
  name: 'John',
  age: 30
};

export default person;
```

```
//main.js

import person from './person.js';

console.log(person.name); // Affiche 'John'
console.log(person.age); // Affiche 30
```

Les modules ES6 – Export nommé 3/5

(export)

- Permet d'exporter plusieurs valeurs, fonctions ou classes à partir d'un module
- Les éléments exportés sont spécifiés entre accolades lors de l'importation

```
// math.js
export const add = (a, b) => a + b;
export const subtract = (a, b) => a - b;
export const multiply = (a, b) => a * b;
```

```
// app.js
import { add, subtract, multiply} from './math.js';

console.log(add(5, 3)); // Affiche 8
console.log(subtract(10, 4)); // Affiche 6
console.log(multiply(2, 6)); // Affiche 12
```

Les modules ES6 – Import avec renommage 4/5

(import)

- Permet de renommer les éléments importés lors de leur importation
- Utile pour éviter les conflits de noms ou pour utiliser des noms plus significatifs

```
// math.js
export const add = (a, b) => a + b;
export const subtract = (a, b) => a - b;
```

```
// app.js
import { add as addition, subtract as substraction } from './math.js';

console.log(addition(5, 3));      // Affiche 8
console.log(substraction(10, 4)); // Affiche 6
```

Les modules ES6 – MJS 5/5

- Avec les modules ECMAScript (**ESM**), les fichiers peuvent utiliser l'extension **.mjs**
 - Les fichiers **.mjs** sont utilisés pour indiquer explicitement que le fichier est un module
- Ces fichiers sont traités comme des modules ECMAScript
 - l'exportation et l'importation se font de la même manière que dans les fichiers **.js**

```
// Dans math.mjs
export const add = (a, b) => a + b;
export const subtract = (a, b) => a - b;
```

```
// Dans main.js
import { add, subtract } from './math.mjs';

console.log(add(5, 3));    // Affiche 8
console.log(subtract(10, 4)); // Affiche 6
```

VUE.JS

Introduction



Introduction

- Vue (prononcé /vju:/, comme **view**)
- Framework JavaScript permettant de créer des interfaces utilisateur
- S'appuie sur les standards HTML, CSS et JavaScript
- Fournit un modèle de programmation déclaratif basé sur des composants
- Vue 3 est la version par défaut depuis février 2022
 - La version 2 ne doit plus être utilisée depuis décembre 2023
- Vue.js possède un site officiel
 - <https://vuejs.org/>

VUE.JS

Histoire et acteurs

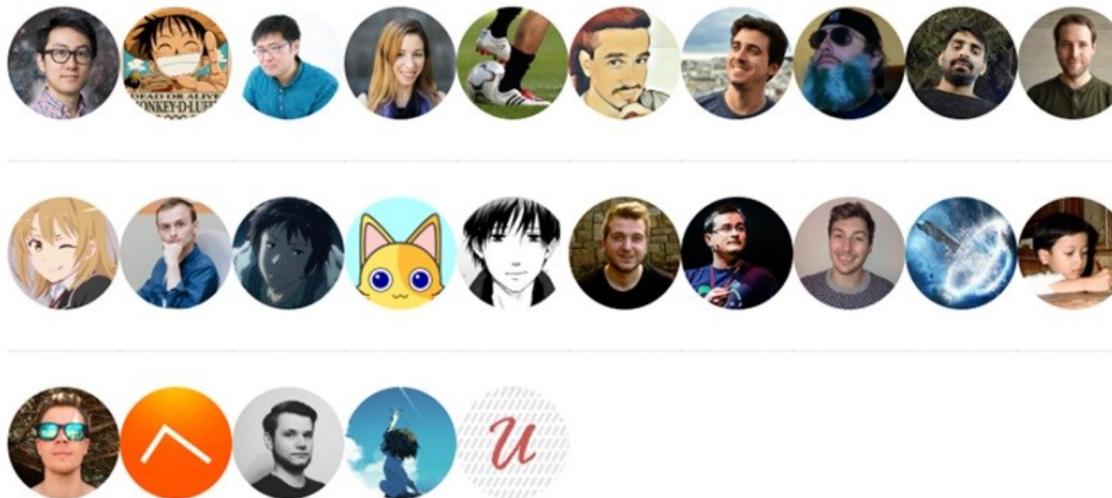


Histoire

- Vue a été créé par **Evan You**, un ancien employé de Google
- Chez Google, Il découvre les Framework AngularJS et Meteor
 - Il découvre alors des mécanismes de réactivité
- En 2013, il publie sur Github une première version de Vue.js
- En 2024, Vue se retrouve sur le même podium que React et Angular
 - Le projet Github présente un nombre important d'étoiles

Équipe

- Vue dispose d'une équipe internationale d'une trentaine de personnes
- La présentation de l'équipe est disponible sur le site officiel
 - <https://vuejs.org/about/team.html>

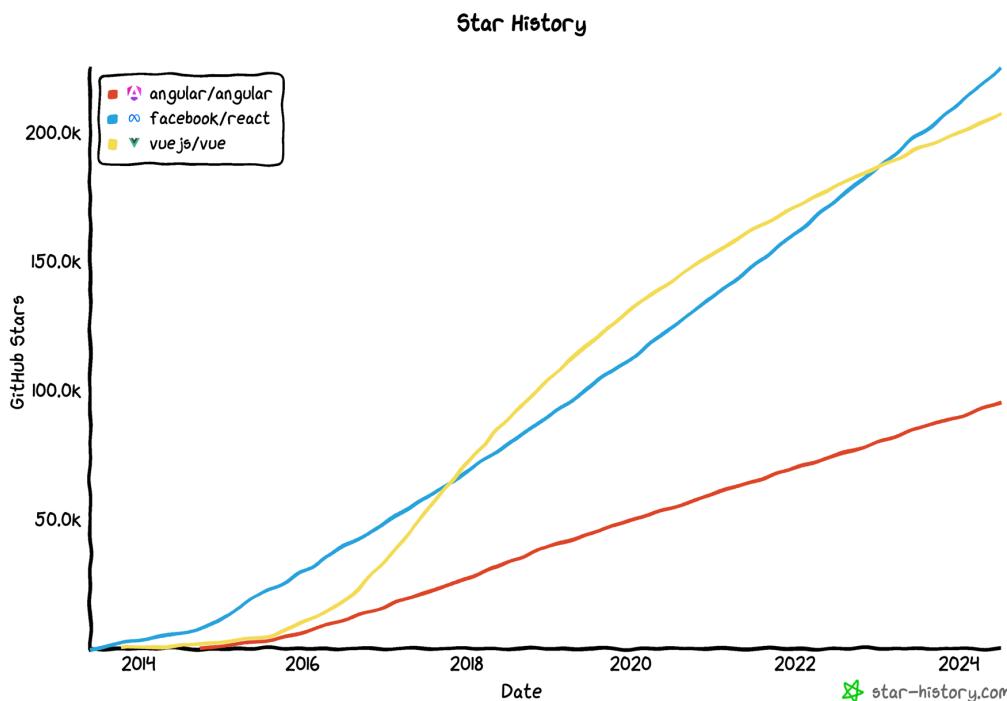


Financement et Business model

- Vue.js est un projet indépendant
 - À la différence d'Angular qui est porté par Google, ou encore React (Facebook)
- Les financements limités proviennent de donations (Patreon) ou de Sponsoring
- Grâce au succès du projet, les contributeurs peuvent travailler à temps plein sur Vue
- Les sources de donations sont assez variées pour préserver l'indépendance de Vue

Popularité

- Vue.js est l'un des projets avec le plus de stars sur Github
 - <https://github.com/vuejs/vue>



VUE.JS

Installation



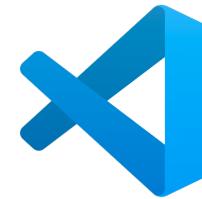
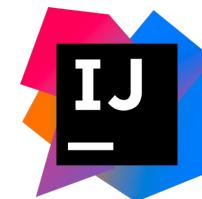
Installation : Node.js 1/2

- Télécharger et installer Node.js :
 - <https://nodejs.org/en/download>
- Choisir la version LTS (Long Term Support)
 - Version recommandée pour la stabilité et la sécurité
- Vérifier que l'installation est correcte :
 - Node.js

```
$ node --version  
v20.11.0
```

Installation : IDE 2/2

- Choisir et installer un IDE
- IntelliJ IDEA Ultimate
 - version d'essai de 30 jours
 - <https://www.jetbrains.com/idea/download/?section=mac>
- Visual Studio Code
 - gratuit et open-source
 - <https://code.visualstudio.com/download>



VUE.JS

Premier programme



Premier programme 1/2

- Dans un dossier `0_demo`, exécuter la commande :

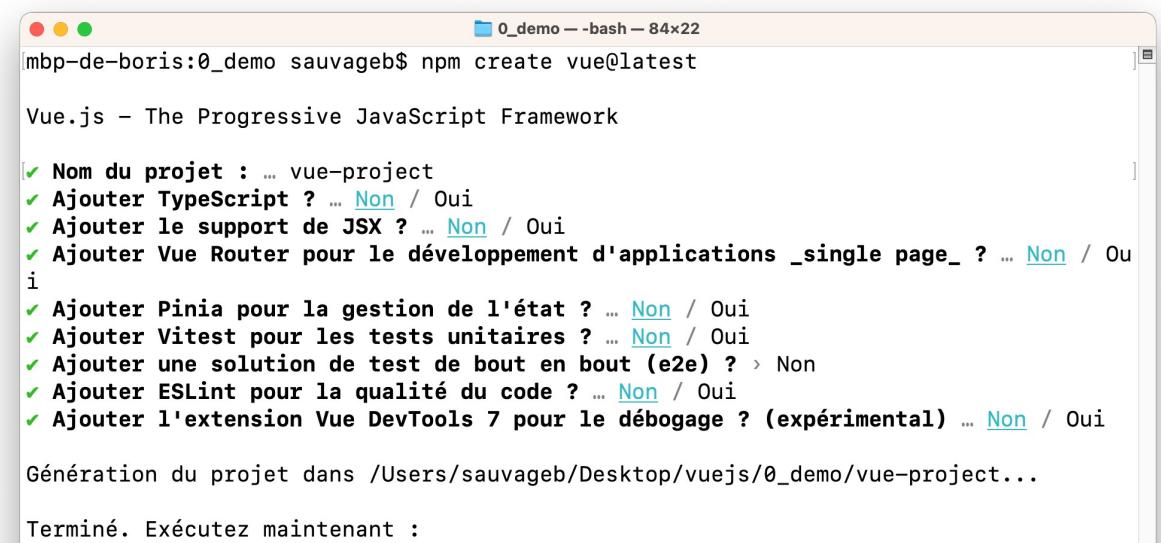
```
$ npm create vue@latest
```

- Cette commande installe et exécute `create-vue`

- Un outil de génération de projet
- <https://github.com/vuejs/create-vue>

- Répondez aux questions du terminal

- Non à toutes



```
mbp-de-boris:0_demo sauvageb$ npm create vue@latest

Vue.js – The Progressive JavaScript Framework

✓ Nom du projet : ... vue-project
✓ 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

Génération du projet dans /Users/sauvageb/Desktop/vuejs/0_demo/vue-project...

Terminé. Exécutez maintenant :
```

Premier programme 2/2

- Ouvrez votre projet avec l'IDE
- Installez les dépendances (à la racine du dossier du projet)

```
$ npm install
```

- Démarrez le projet

```
$ npm run dev
```



```
Génération du projet dans /Users/sauvageb/Desktop/vuejs/0_demo/vue-project...
```

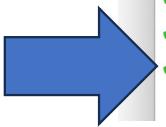
```
Terminé. Exécutez maintenant :
```

```
cd vue-project  
npm install  
npm run dev
```

```
mbp-de-boris:vue-project sauvageb$ npm run dev  
  
> vue-project@0.0.0 dev  
> vite  
  
Port 5173 is in use, trying another one...  
  
VITE v5.3.3 ready in 829 ms  
  
→ Local: http://localhost:5174/  
→ Network: use --host to expose  
→ press h + enter to show help
```

Debogguer avec Vue.js

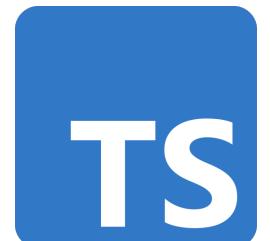
- Pour débogguer le code vue.js, il est possible d'utiliser l'extension **Vuejs-devtools**
 - <https://devtools.vuejs.org/guide/installation.html>
- Il est alors nécessaire d'installer Vuejs-devtools durant la création du projet



```
✓ 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
```

Vue.js avec Typescript

- TypeScript est un langage open source maintenu et développé par Microsoft
 - Il est apprécié et utilisé par de nombreux développeurs de logiciels à travers le monde
- Il ajoute de nouvelles fonctionnalités au langage (comme le typage)
 - Il rend le code sécurisé et robuste en évitant de nombreux bugs avec une phase de compilation
 - Il propose des interfaces, des annotations, des types,
- Le TypeScript est transcompilé en Javascript Vanilla
- Développer en Typescript avec le Framework Vue.Js est possible



VUE.JS

Atelier 1 : Premier programme



VUE.JS

Vue : introduction



Introduction à Vue : Create Vue

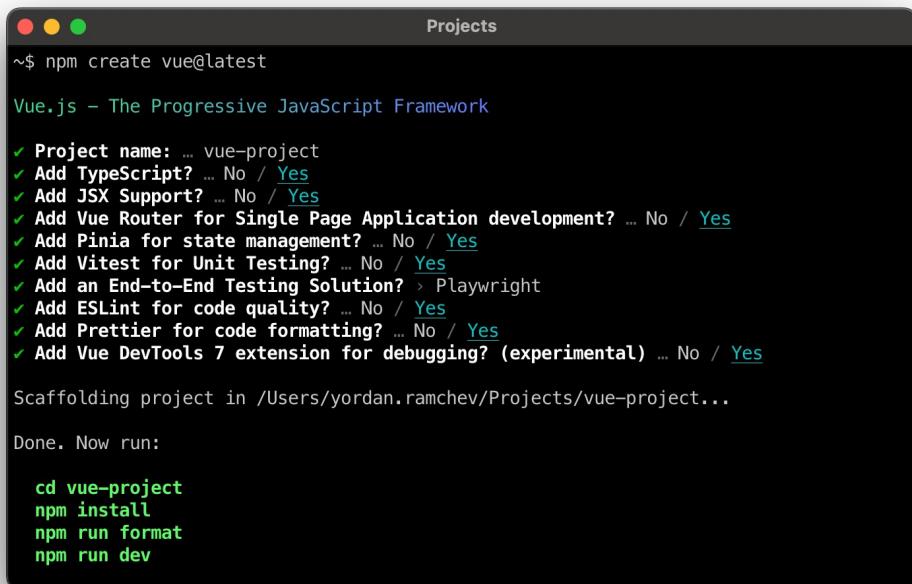
Il y a deux manières de mettre en place une application Vue :

- En utilisant la version via CDN

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

- En utilisant la commande **create-vue (recommandé)**

```
$ npm create vue@latest
```



The screenshot shows a terminal window titled "Projects" with the following text:

```
~$ npm create vue@latest
Vue.js – The Progressive JavaScript Framework

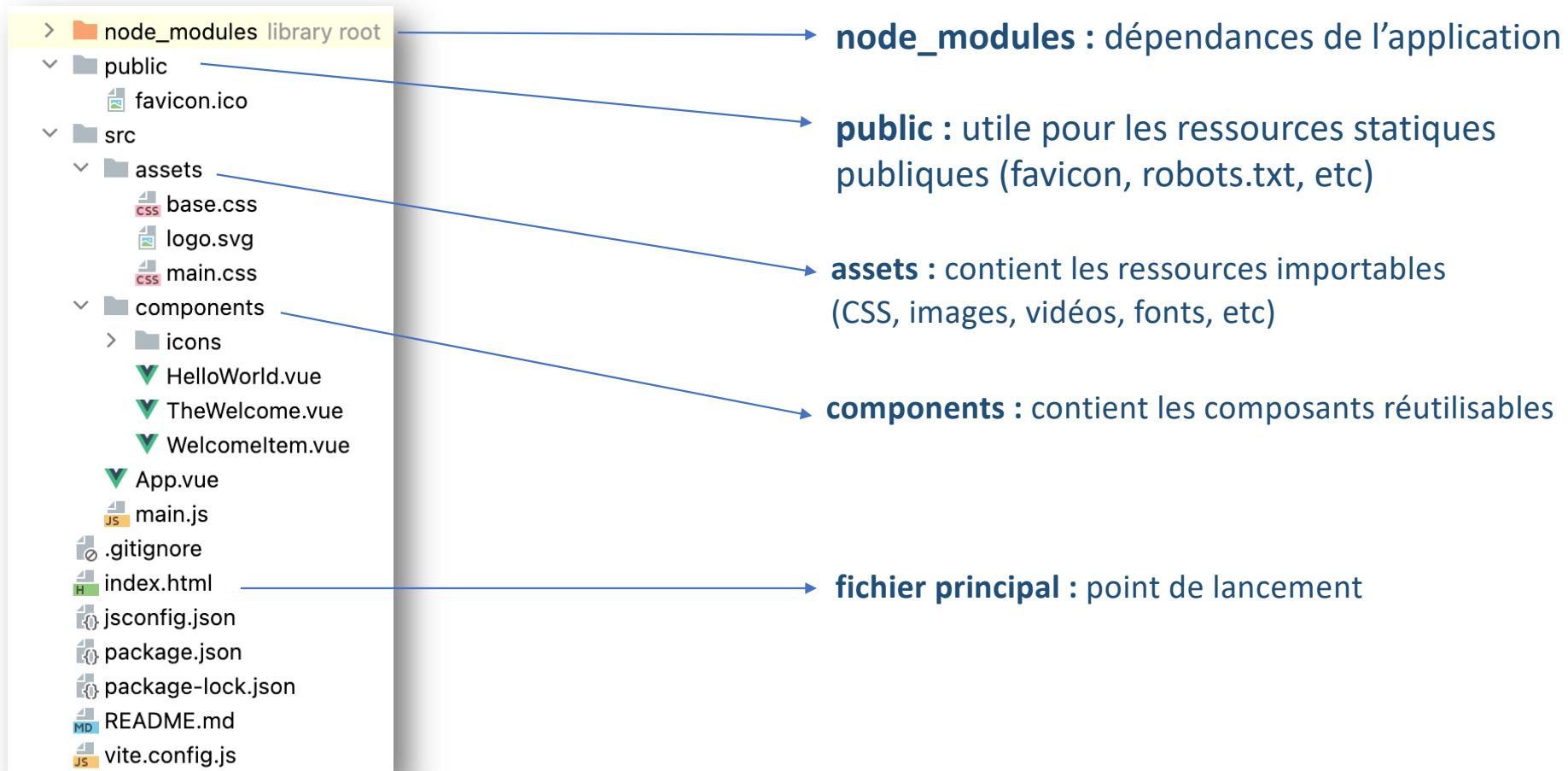
✓ Project name: ... vue-project
✓ Add TypeScript? ... No / Yes
✓ Add JSX Support? ... No / Yes
✓ Add Vue Router for Single Page Application development? ... No / Yes
✓ Add Pinia for state management? ... No / Yes
✓ Add Vitest for Unit Testing? ... No / Yes
✓ Add an End-to-End Testing Solution? > Playwright
✓ 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:

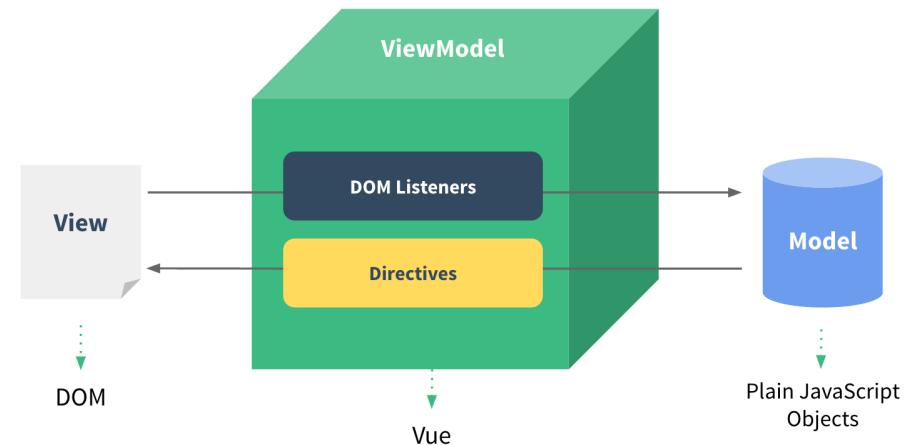
  cd vue-project
  npm install
  npm run format
  npm run dev
```

Architecture d'un projet généré



Architecture globale

- Vue.js est framework orienté sur le design pattern MVVM
 - Vue.js relie la **View** avec le **Model** via des liaisons de données bidirectionnelles
- Les **View** définissent ce que l'utilisateur voit
- Les **Model** définissent les données Javascript
 - La manipulation du DOM et le formatage des sorties sont gérés par les Directives et les Filtres



VUE.JS

Le linter



ESLint

- Installable au démarrage, il est utilisable sur tout le projet

```
$ npm run lint
```

- ESLint est un outil d'analyse de code statique
 - permet d'identifier les erreurs dans le Javascript

```
/usr/local/opt/node@18/bin/npm run lint

> tester@0.0.0 lint
> eslint . --ext .vue,.js,.jsx,.cjs,.mjs --fix --ignore-path .gitignore

/Users/sauvageb/Desktop/vuejs/tester/src/helpers/helpers.js
  2:9  error  'a' is assigned a value but never used  no-unused-vars

✖ 1 problem (1 error, 0 warnings)
```



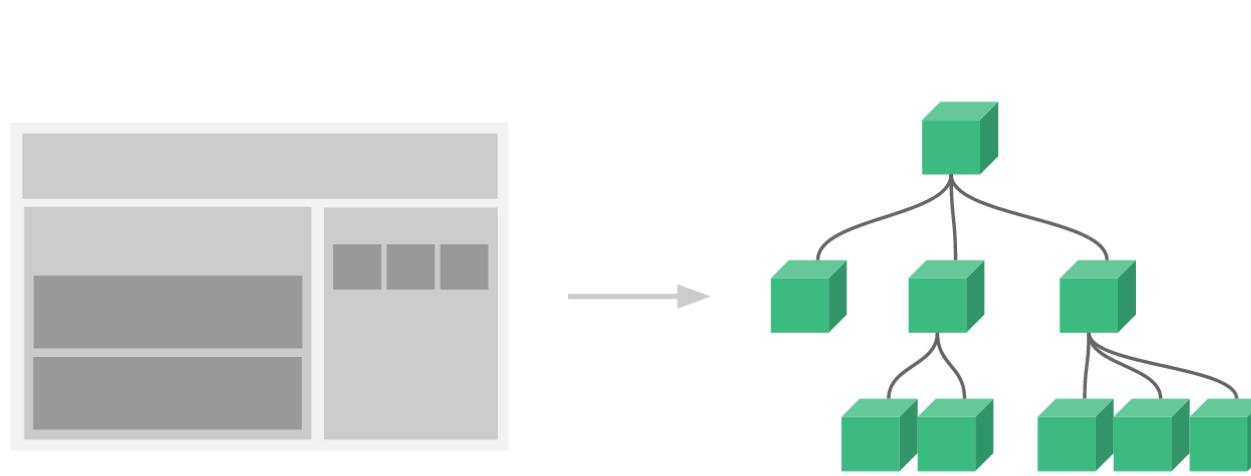
VUE.JS

Le composant



Le composant 1/3

- Partie réutilisable et autonome de l'interface utilisateur d'une application
 - Les composants structurent et modularisent le code en morceaux indépendants et réutilisables
- Un composant Vue.js est une instance de Vue qui a son propre état, ses propres comportements et son propre rendu



Le composant : exemple 2/3

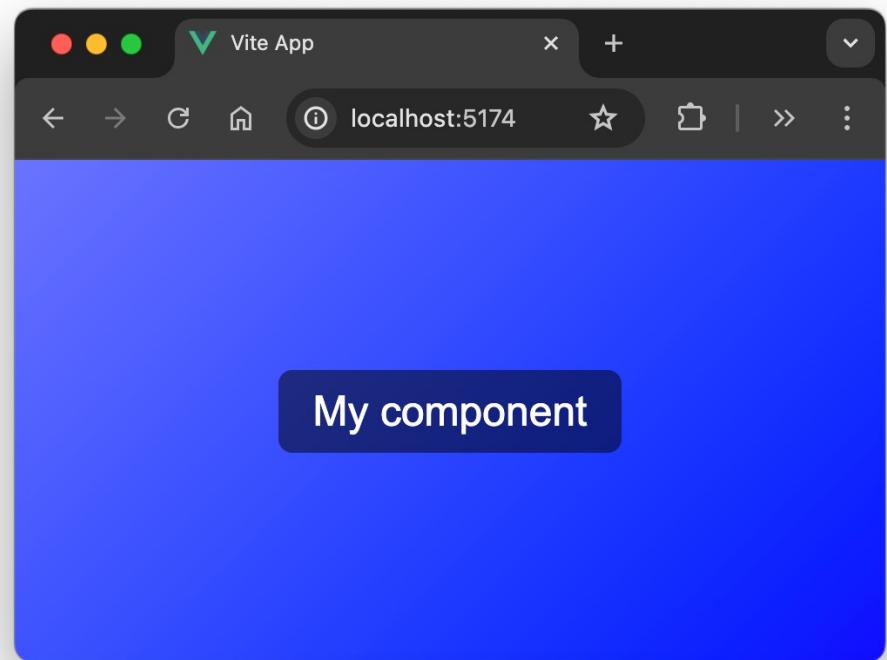
```
<script setup></script>

<template>
  <div class="example-component">
    <p>My component</p>
  </div>
</template>

<style scoped>
.example-component {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  background: linear-gradient(135deg, #6b73ff 0%, #000dff 100%);
}

.example-component p {
  font-size: 1.5rem;
  color: white;
  font-family: 'Arial', sans-serif;
  padding: 10px 20px;
  background: rgba(0, 0, 0, 0.5);
  border-radius: 8px;
}
</style>
```

MyComponent.vue



```
src
  assets
  components
  main.js
  MyComponent.vue
  .gitignore
  index.html
```

Le composant : utilisation 3/3

- Une fois créé, le composant est utilisable dans le template d'autres composants
 - Nécessite d'être importé
- Le @ dans le composant représente src
 - Il est défini dans vite.config.js

```
// https://vitejs.dev/config/
export default defineConfig({
  plugins: [
    vue(),
  ],
  resolve: {
    alias: { '@': fileURLToPath(new URL('./src', import.meta.url)) }
  }
})
```

```
<script setup>

import MyComponent from "@/MyComponent.vue";

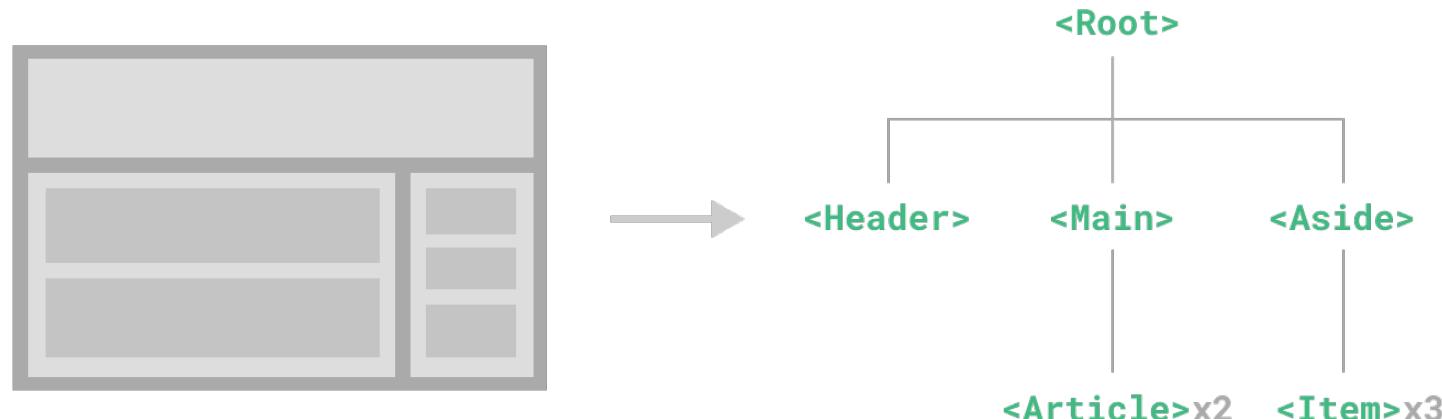
</script>

<template>
  <MyComponent/>
</template>

<style scoped>
</style>
```

Arbre de composants

- Organisation similaire à la manière dont nous imbriquons les éléments HTML natifs
- Vue implémente son propre modèle de composants
 - Encapsuler du contenu et de la logique personnalisés dans chaque composant

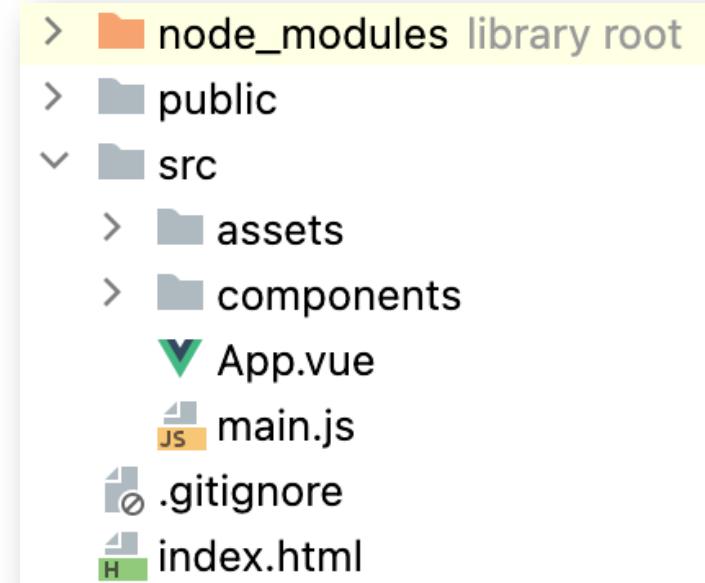


Composant principal 1/2

- Par défaut, votre application possède un composant principal
 - Nous l'utiliserons pour afficher nos composants

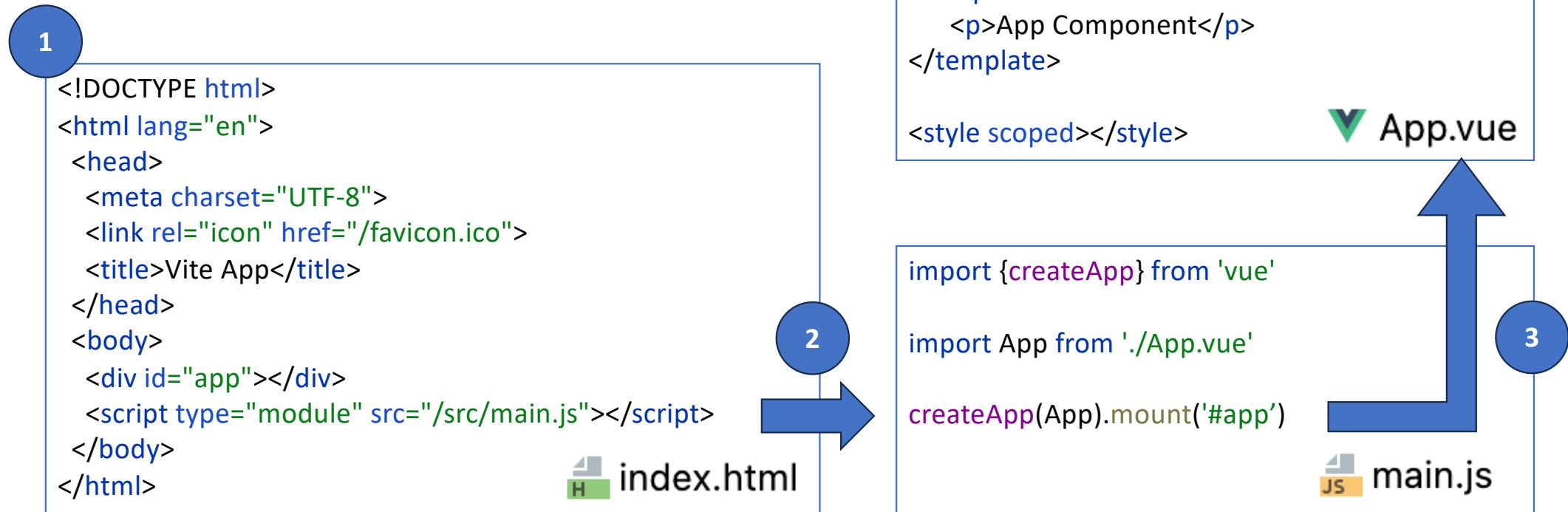
- Les étapes du chargement sont le suivantes :

- 1 Index.html est chargé par le navigateur
- 2 Index.html charge main.js
- 3 main.js charge App.vue



Composant principal 2/2

- Dans le code, voici le déroulement des étapes :



VUE.JS

CSS et portée



Le CSS par composant

- Chaque composant peut inclure du CSS s'appliquant sur le DOM de ce dernier
- Ces styles font partie du système de modularisation Javascript
 - Ils sont inclus automatiquement lors du chargement du composant
- Ces styles se situent dans une balise **style scoped**
 - Ils s'appliqueront donc uniquement au composant où se trouve le style

```
<style scoped>

.example-component {
  display: flex;
  background: linear-gradient(135deg, #6b73ff 0%, #000dff 100%);
}

</style>
```

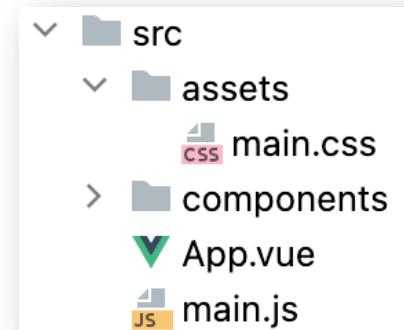
Le CSS global à l'application

- Si vous souhaitez appliquer un CSS global à l'application, vous pouvez utiliser une feuille de stylee globale, importée dans main.js

```
import {createApp} from 'vue'  
import App from './App.vue'  
  
// Global CSS  
import './assets/main.css'  
  
createApp(App).mount('#app')
```



main.js



```
#app {  
  max-width: 1280px;  
}
```



main.css

Framework CSS

- Vue.js supporte l'utilisation de nombreux Framework CSS :

- Vuetifyjs (<https://vuetifyjs.com>)
- Element Plus (<https://element-plus.org>)
- PrimeNG (<https://primevue.org>)
- Tailwind (<https://v2.tailwindcss.com/docs/guides/vue-3-vite>)
- BootstrapVue (<https://bootstrap-vue.org>)
- ...

VUE.JS

Composant et Cycle de vie

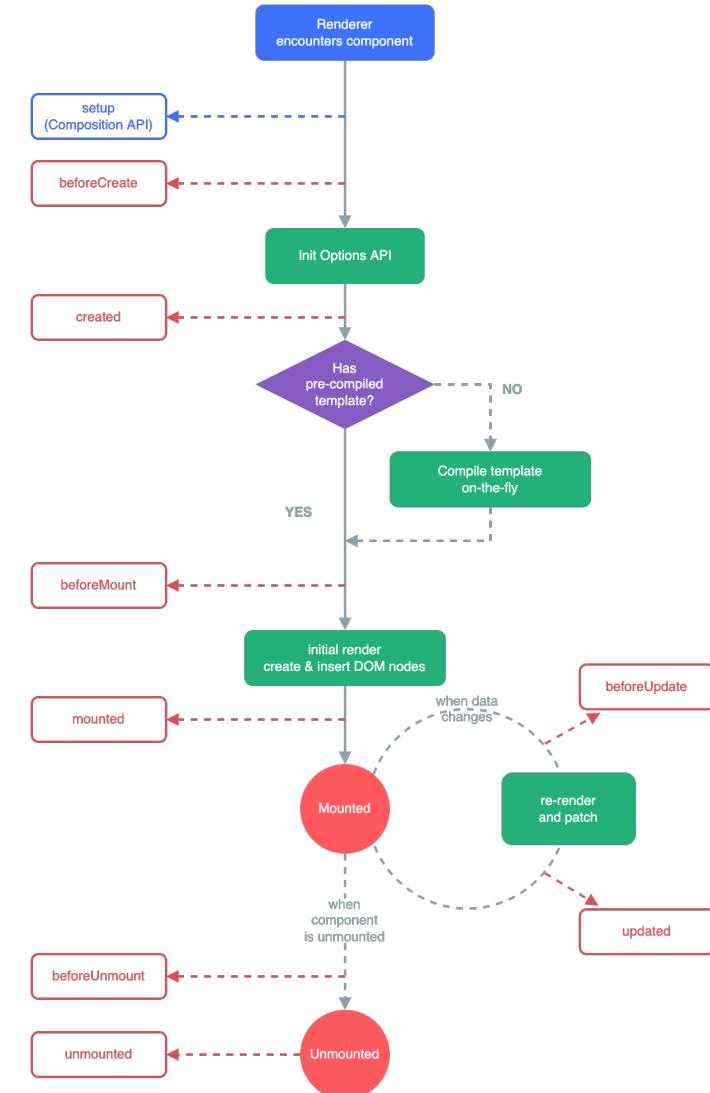


Cycle de vie du composant 1/3

- Chaque instance d'un composant Vue passe par une série d'étapes d'initialisation
- Par exemple, l'observation des données, la compilation du template, le montage de l'instance sur le DOM, la mise à jour lorsque les données changent
- Il existe des hooks
 - fonctions du cycle de vie
 - Permettent d'ajouter du code à des étapes spécifiques
 - **onMounted, onUpdated, onUnmounted...**
- <https://fr.vuejs.org/guide/essentials/lifecycle>

Cycle de vie du composant 2/3

- Diagramme du cycle de vie d'une instance



Cycle de vie du composant 3/3

- Voici la liste des hooks du cycle de vie :

- onMounted()
- onUpdated()
- onUnmounted()
- onBeforeMount()
- onBeforeUpdate()
- onBeforeUnmount()
- onErrorCaptured()
- onRenderTracked()
- onRenderTriggered()
- onActivated()
- onDeactivated()
- onServerPrefetch()
- <https://fr.vuejs.org/api/composition-api-lifecycle.html>

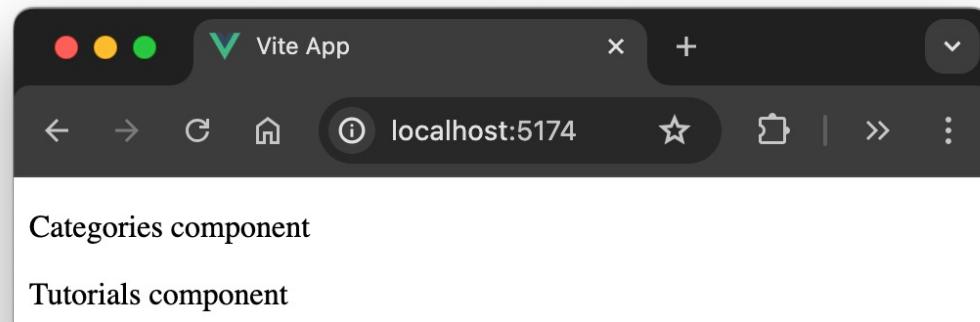
VUE.JS

Atelier 2 : Mettre en place l'application fil rouge



Atelier : Mettre en place l'application fil rouge

- Créez une nouvelle application nommée **tutorials**
- Cette application est une application de gestion de tutoriels
 - **Tutorials** sera le fil rouge de la formation
- Créez 2 composants nommés **tutorials** et **categories** dans le répertoire **composants**
- Affichez les



VUE.JS

Les templates



Les templates

- Dans chacun de vos composant, il y a une partie HTML
- Vous n'avez pas besoin de redéfinir `<html>`, `<body>`
 - Ces balises sont déjà incluses dans `index.html`

```
<template>  
  
  <p>My sentence</p>  
  
</template>
```

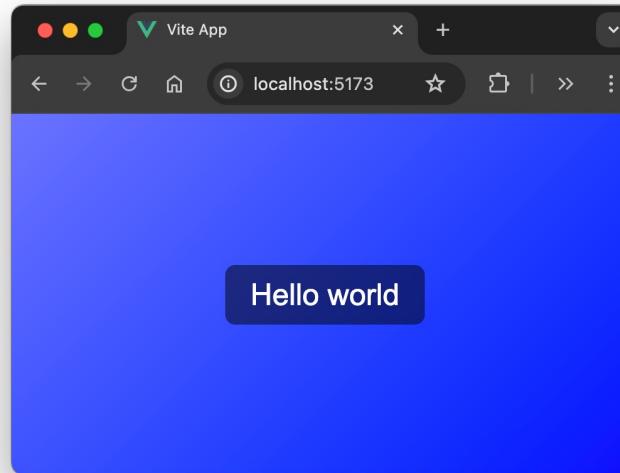
Interpolation

- Utilisation dans l'HTML d'une variable déclarée dans le script
- Les templates sont du HTML syntaxiquement valide
 - Peut être analysé par des navigateurs
 - Peut être des analyseurs HTML conformes aux spécifications
- Les doubles moustaches interprètent les données comme du texte brut

```
<script setup>

const message = 'Hello world';
</script>

<template>
  <p>{{ message }}</p>
</template>
```



VUE.JS

Les directives



Les directives Vue

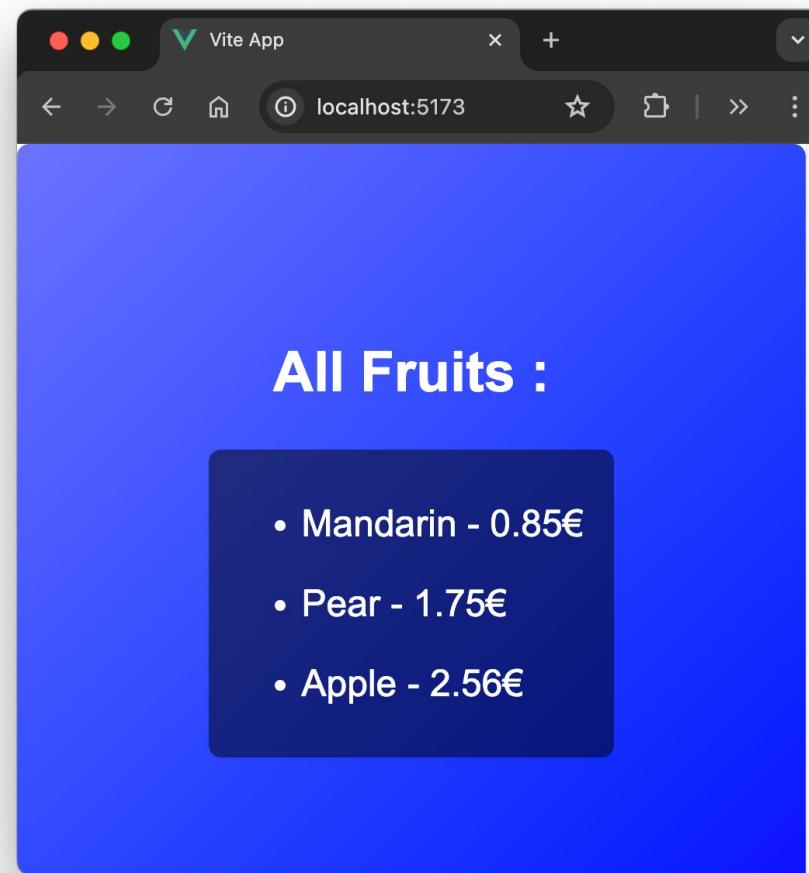
- Attributs HTML spéciaux avec le préfixe v-
 - donnent à la balise HTML des fonctionnalités supplémentaires
- Se connectent à l'instance Vue pour créer des interfaces dynamiques et réactives
 - La création de pages réactives est simplifiée avec les directives de Vue

Il existe plusieurs types de directives :

- Les directives structurelles : modifient le DOM en ajoutant/modifiant un élément
- Les directives d'attributs : modifient l'apparence
- Les directives de contenu
- Les directives personnalisées

Données d'exemple pour les directives

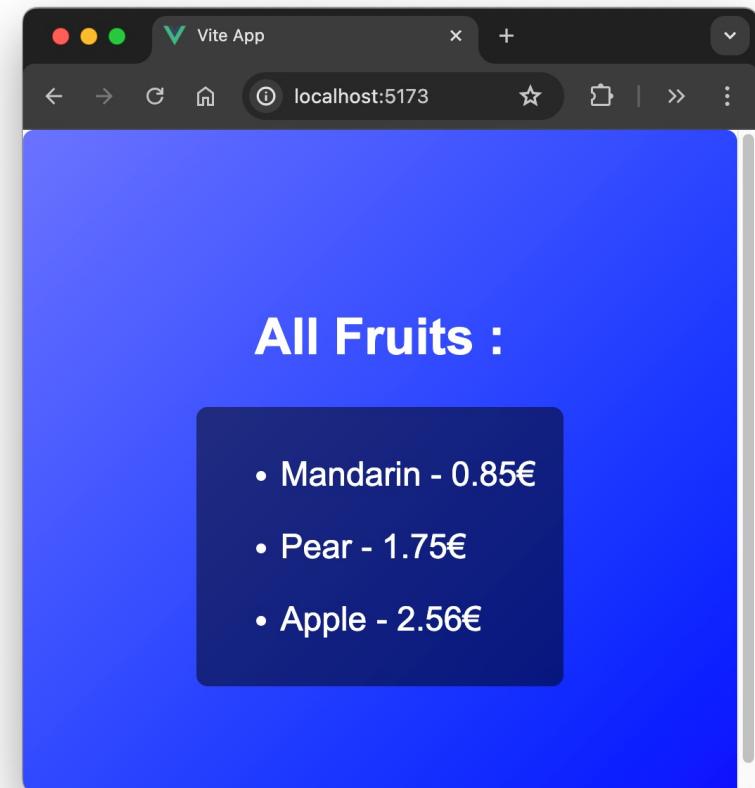
```
<script setup>
const fruits = [
  {
    name: "Mandarin",
    price: 0.85,
    origin: "SPAIN"
  },
  {
    name: "Pear",
    price: 1.75,
    origin: "BELGIUM"
  },
  {
    name: "Apple",
    price: 2.56,
    origin: "FRANCE"
  }
];
</script>
```



v-for

- Rend l'élément ou le bloc d'un template plusieurs fois en fonction des données sources

```
<template>
  <div class="container">
    <h2>All Fruits :</h2>
    <div>
      <ul v-for="fruit in fruits" :key="fruit.name">
        <li>{{ fruit.name }} - {{ fruit.price }}€</li>
      </ul>
    </div>
  </div>
</template>
```

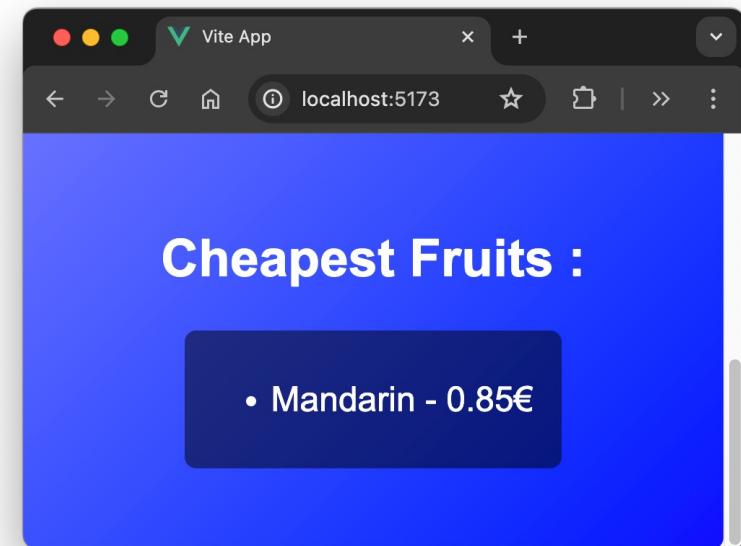


- <https://fr.vuejs.org/api/built-in-directives.html#v-for>

v-if v-else v-if-else

- Utilisée pour restituer conditionnellement un bloc
 - Le bloc ne sera rendu que si l'expression de la directive retourne une valeur évaluée à vrai

```
<template>
  <div class="container">
    <h2>Cheapest Fruits :</h2>
    <div>
      <ul v-for="fruit in fruits" :key="fruit.name">
        <li v-if="fruit.price < 1">{{ fruit.name }} - {{ fruit.price }}€</li>
      </ul>
    </div>
  </div>
</template>
```



- <https://fr.vuejs.org/guide/essentials/conditional.html#v-if>

v-show

- Une autre option pour afficher conditionnellement un élément est la directive v-show
 - v-show bascule uniquement la propriété CSS display de l'élément
- <https://fr.vuejs.org/guide/essentials/conditional.html#v-show>

```
...<table>
<thead>
<tr>
<th>name</th>
<th>price (€)</th>
<th>origin</th>
<th>flag</th>
</tr>
</thead>
<tbody v-for="fruit in fruits" :key="fruit.name">
<tr>
<td>{{ fruit.name }}</td>
<td>{{ fruit.price }}</td>
<td>{{ fruit.origin }}</td>
<td>



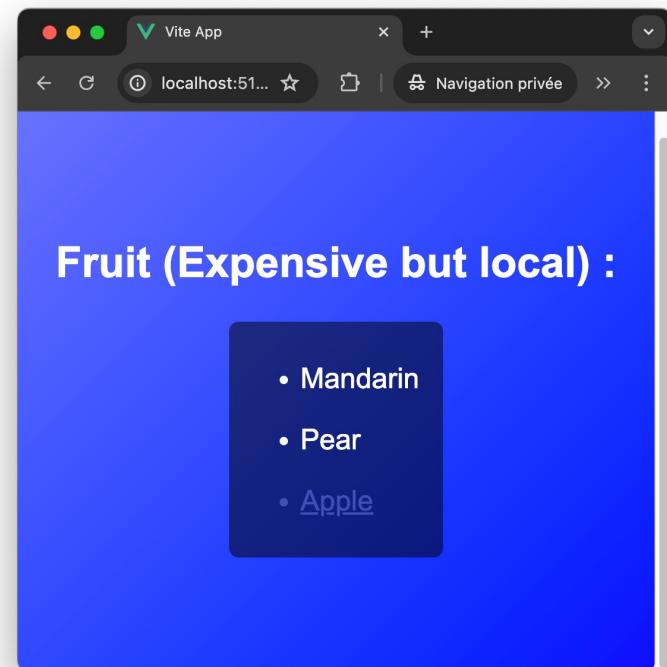
<span v-show="['FRANCE', 'SPAIN', 'BELGIUM'].indexOf(fruit.origin) === -1">N/A</span>
</td>
</tr>
</tbody>
```

Fruits (origin):			
name	price (€)	origin	flag
Mandarin	0.85	SPAIN	
Pear	1.75	BELGIUM	
Apple	2.56	FRANCE	

v-bind 1/2

- Lie dynamiquement un ou plusieurs attributs, ou une prop d'un composant à une expression
- <https://fr.vuejs.org/api/built-in-directives.html#v-bind>

```
<template>
<div class="container">
  <h2>Fruit (Expensive but local) :</h2>
  <div>
    <ul v-for="fruit in fruits" :key="fruit.name">
      <li v-bind:class="{
        'expensive': fruit.price > 2, 'underline': fruit.origin === 'FRANCE'
      }">
        {{ fruit.name }}
      </li>
    </ul>
  </div>
</div>
</template>
```

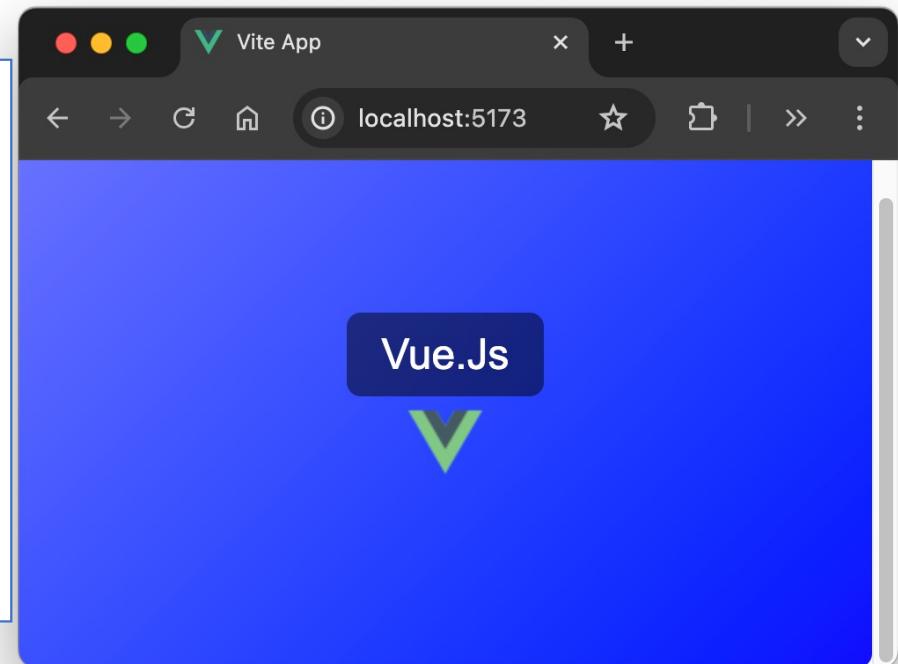


v-bind 2/2

- Le raccourci de v-bind est :attribut

```
<script setup>
const size = 12;
const url = "https://img.icons8.com/color/48/vue-js.png";
</script>

<template>
<div class="container">
  <div :style="{ fontSize: size }>Vue.Js</div>
  
</div>
</template>
```

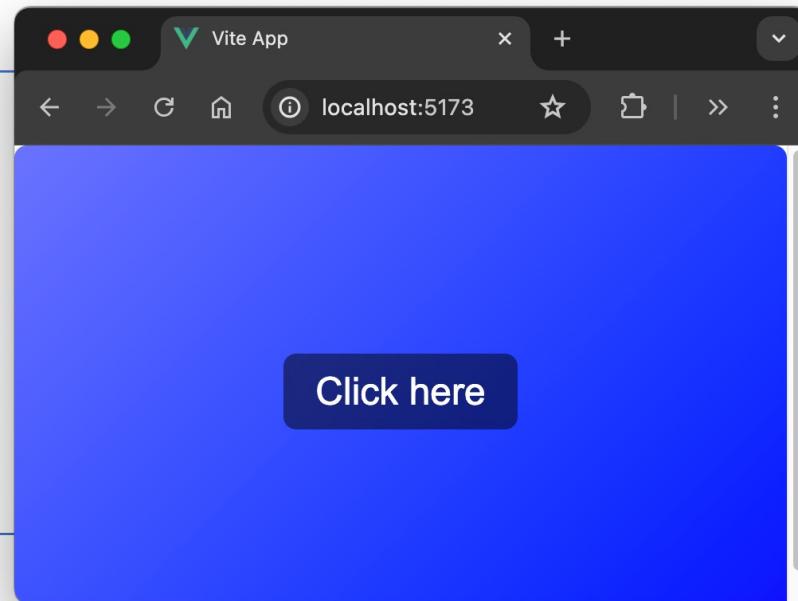


V-ON

- Attache un écouteur d'événements à l'élément
- Le raccourci est @
- <https://fr.vuejs.org/api/built-in-directives.html#v-on>

```
<script setup>
const myAction = () => alert('Hello');
</script>

<template>
  <div class="container">
    <button @click="myAction()">Click here</button>
  </div>
</template>
```



v-text

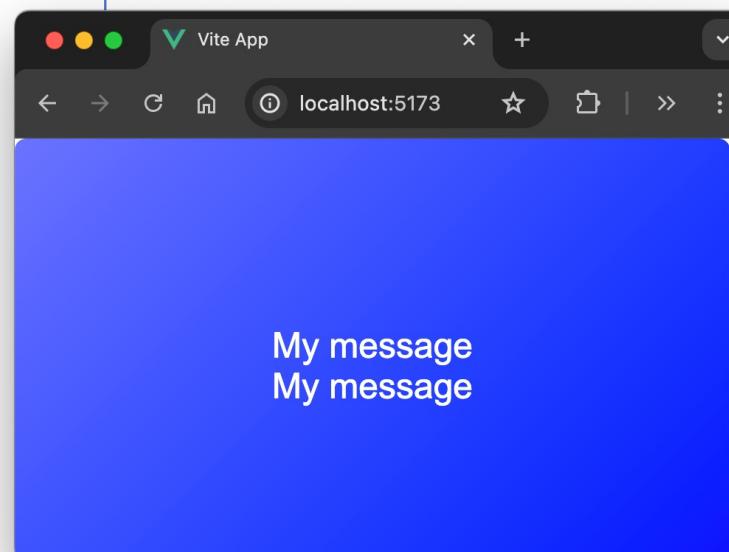
- Met à jour le contenu texte d'un élément
- <https://fr.vuejs.org/api/built-in-directives.html#v-text>

```
<script setup>
const msg = "My message";
</script>

<template>
  <div class="container">

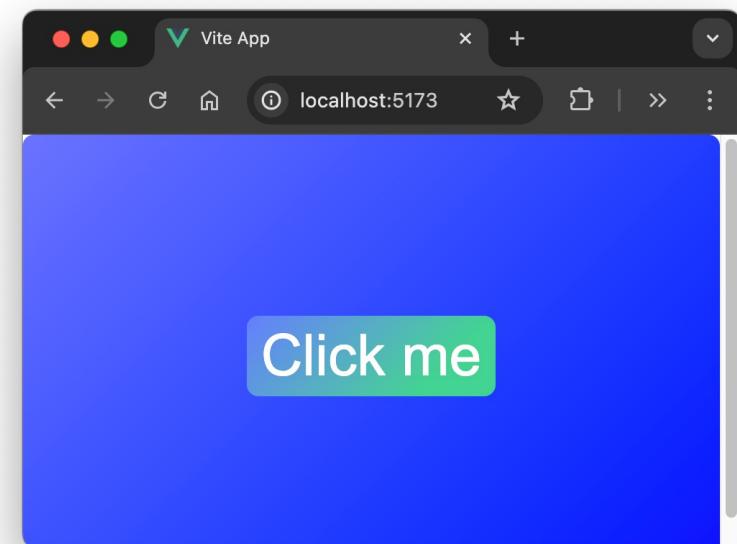
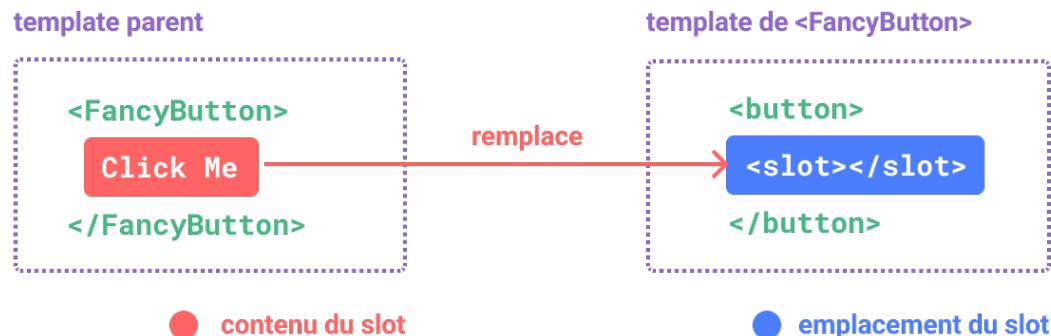
    <span v-text="msg"></span>
    <!-- same as -->
    <span>{{ msg }}</span>

  </div>
</template>
```



v-slot 1/2

- Désigne les slots nommés ou les slots scopés qui s'attendent à recevoir des props
- <https://fr.vuejs.org/guide/components/slots.html>



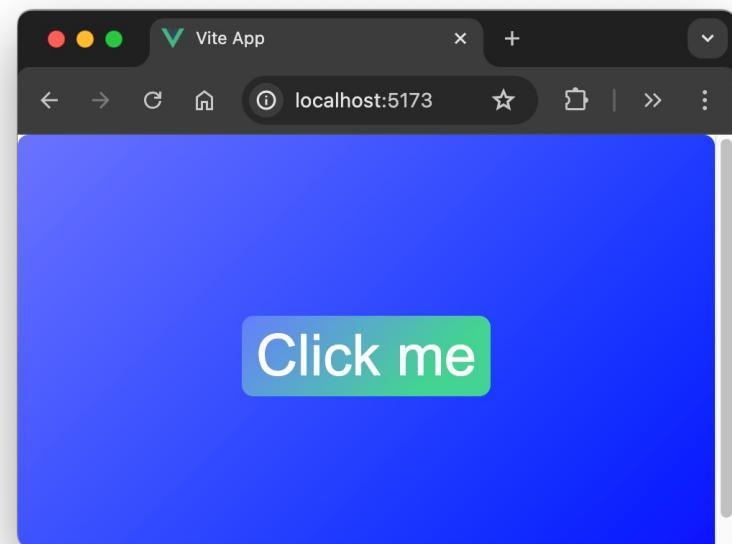
v-slot 2/2

```
<script setup></script>
<template>
  <button class="fancy-btn">
    <slot/>
  </button>
</template>
```

▼ MyButton.vue

```
<script setup>
import MyButton from "@/components/MyButton.vue";
</script>
<template>
  <div class="container">
    <MyButton>
      Click me <!-- slot content -->
    </MyButton>
  </div>
</template>
```

▼ App.vue



v-pre

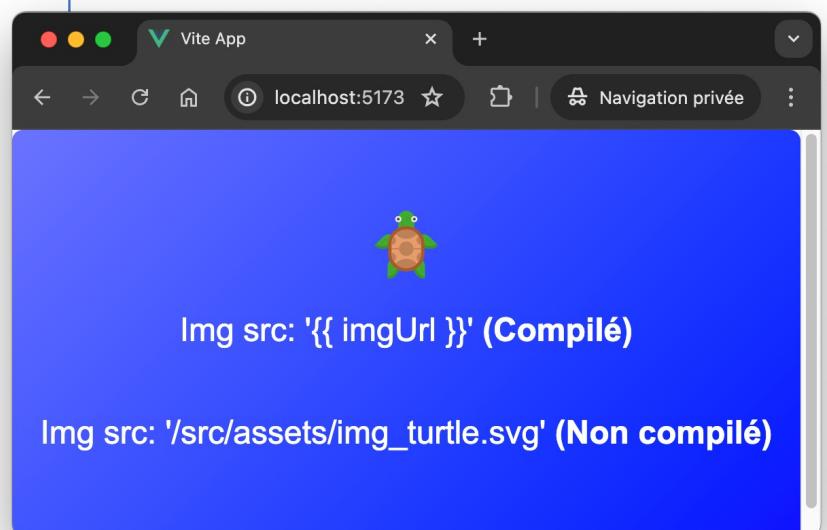
- Ignore la compilation pour cet élément et tous ses enfants
- <https://fr.vuejs.org/api/built-in-directives.html#v-pre>

```
<script setup>
import imgUrl from '@/assets/img_turtle.svg';
</script>

<template>
  <div class="container">

    
    <p v-pre>Img src: '{{ imgUrl }}' <strong>(Non compilé)</strong></p>
    <p>Img src: '{{ imgUrl }}' <strong>(Compilé)</strong></p>

  </div>
</template>
```



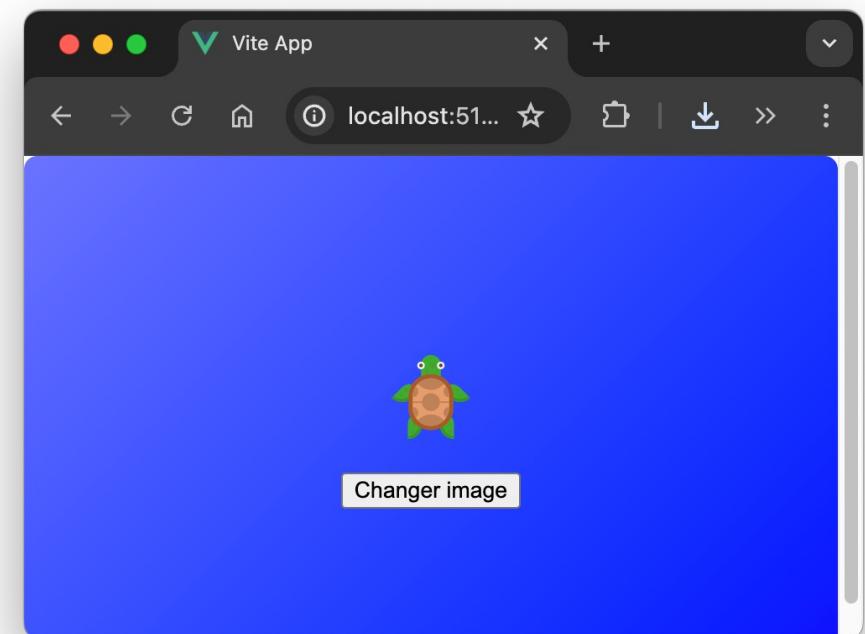
v-once

- Rend l'élément et le composant une seule fois, et ignore les mises à jour futures
- <https://fr.vuejs.org/api/built-in-directives.html#v-once>

```
<script setup>
import {ref} from "vue";
import imgFish from '@/assets/img_fish.svg';

const imgUrls = [imgTurtle, imgTiger, imgFish];
const imgIndex = ref(0);
const changelImg = () => {
  imgIndex.value = Math.floor(Math.random() * imgUrls.length);
};
</script>

<template>
<div class="container">
  
  <button @:click="changelImg">Changer image</button>
</div>
</template>
```



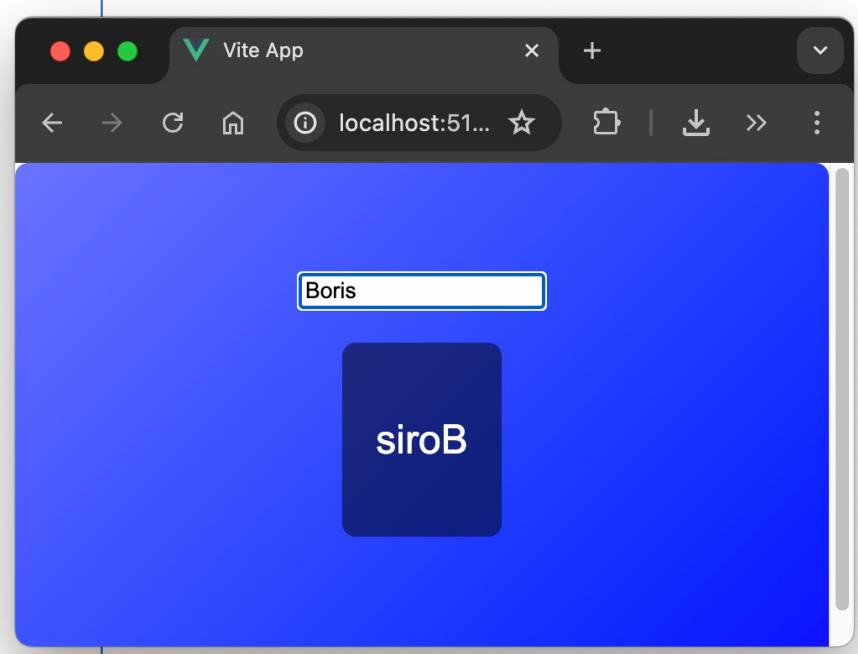
v-memo

- Optimise la performance en mémorisant les sous-arbres de rendu
 - Aide à éviter les rendus inutiles selon certaines conditions
- <https://fr.vuejs.org/api/built-in-directives.html#v-memo>

```
<script setup>
import {ref} from "vue";
const text = ref("");
const expensiveComputation = (input) => {
  //Supposons que cette fonction soit coûteuse en termes de calcul
  return input.split(").reverse().join(");
};

</script>

<template>
<div class="container">
  <input v-model="text" placeholder="Saisir prénom">
  <div v-memo="[text]">
    <!-- Cette partie sera re-rendue seulement si 'text' change -->
    <p>{{ expensiveComputation(text) }}</p>
  </div>
</div>
</template>
```

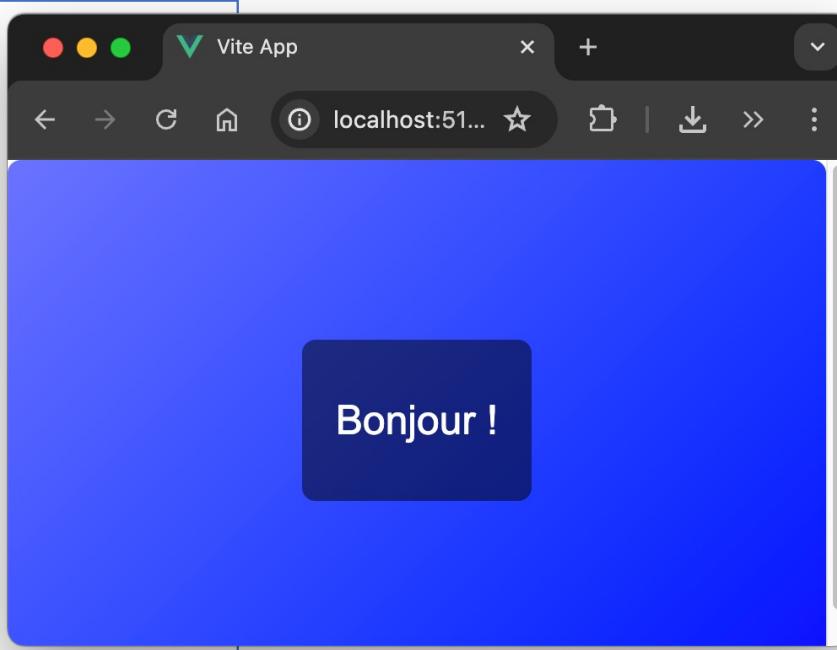


v-cloak

- Utilisée pour cacher un template non compilé jusqu'à ce qu'il soit prêt.
- <https://fr.vuejs.org/api/built-in-directives.html#v-cloak>

```
<template>
  <div class="container">
    <div v-cloak>
      <p>Bonjour !</p>
    </div>
  </div>
</template>

<style scoped>
[v-cloak] {
  display: none;
}
</style>
```

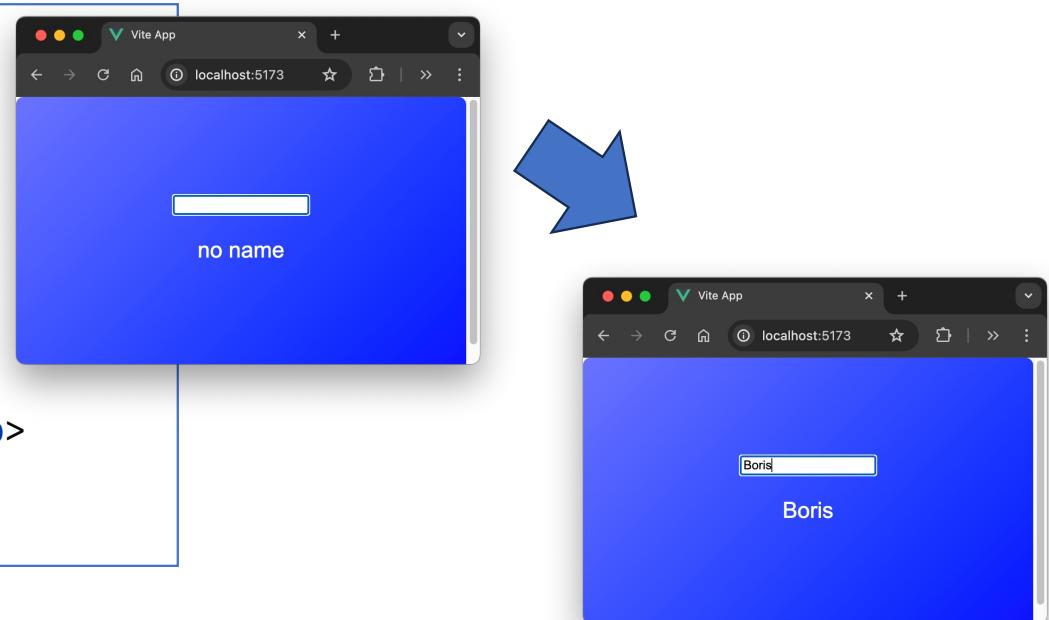


v-model (liaisons bidirectionnelles)

- Crée une liaison bidirectionnelle sur un élément de saisie de formulaire ou un composant
- <https://fr.vuejs.org/api/built-in-directives.html#v-model>

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

<template>
  <div class="container">
    <input type="text" v-model="firstName">
    <p>{{ firstName.length ? firstName : 'no name' }}</p>
  </div>
</template>
```



VUE.JS

Atelier 3 : Afficher une liste de catégories



Atelier : Afficher une liste catégories 1/3

- À partir de l'application **Tutorials** et des 2 composants créés au début :
 - affichez **uniquement** le composant **categories** en page d'accueil
 - Utilisez son sélecteur pour le placer judicieusement dans le fichier **App.vue**
- Créez un tableau nommé **categories** dans la partie script
 - Il doit contenir une dizaine de catégories
 - (Java, Javascript, Typescript, VueJs, C#, Angular, Ruby, Python, C++, Rust, VBA...)
- Dans le template, affichez la liste des categories. Utilisez une directive de structure pour afficher l'ensemble sous forme de liste

Atelier : Framework CSS 2/3

- Pour rendre l'ensemble plus agréable à regarder, utilisez un Framework CSS que vous intégrerez à l'application grâce à npm ou au CDN

Atelier : Aperçu 3/3

Tutorials App

Liste des Catégories

Java

Javascript

TypeScript

PHP

C#

Angular

Ruby

Python

C++

Rust

VBA

VUE.JS

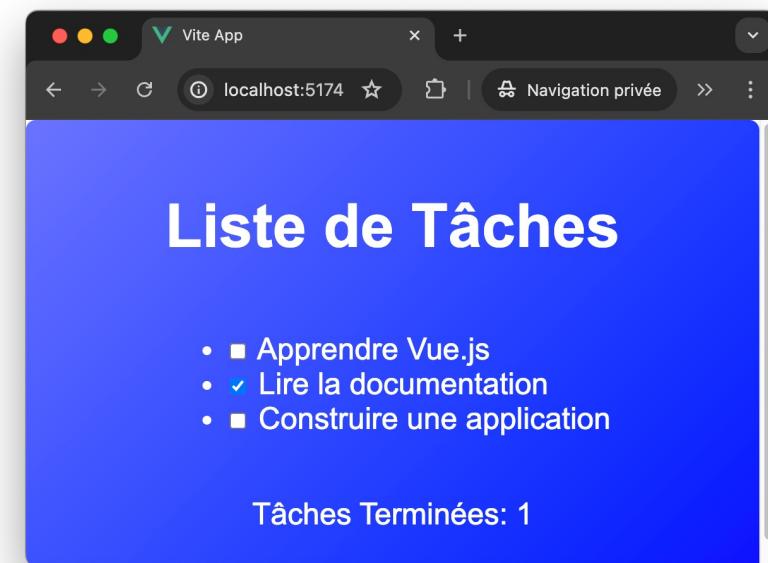
Propriétés calculées



Propriété calculée 1/2

- Les expressions dans le template doivent rester des opérations simples
 - Mettre trop de logique dans vos Template peut les encombrer et les rendre difficiles à maintenir
- Pour des logiques complexes, utilisez des propriétés calculées (**computed**)

```
<template>
  <div class="container">
    <h1>Liste de Tâches</h1>
    <ul>
      <li v-for="task in tasks" :key="task.id">
        <input type="checkbox" v-model="task.completed">
        {{ task.title }}
      </li>
    </ul>
    <p>Tâches Terminées: {{ completedTasksCount }}</p>
  </div>
</template>
```



Propriété calculée 2/2

- Les propriétés calculées sont mises en cache
 - <https://fr.vuejs.org/guide/essentials/computed#computed-caching-vs-methods>
- les propriétés calculées sont mises en cache **en fonction de leurs dépendances réactives**

```
<script setup>
import {computed, ref} from 'vue';
const tasks = ref([
  {id: 1, title: 'Apprendre Vue.js', completed: false},
  {id: 2, title: 'Lire la documentation', completed: true},
  {id: 3, title: 'Construire une application', completed: false},
]);

const completedTasksCount = computed(() => {
  return tasks.value.filter(task => task.completed).length;
});
</script>
```

VUE.JS

Références réactives



Références réactives avec `ref()` 1/2

- Sont des objets spéciaux qui permettent à Vue de réagir aux changements de données et de mettre à jour le DOM en conséquence
- `ref()` est une fonction qui crée une référence réactive à une valeur
 - `ref()` prend l'argument et le renvoie enveloppé dans un objet `ref` avec une propriété `.value`

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

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

count.value++;
console.log(count.value)    // 1
</script>
```

Références réactives avec `ref()` 2/2

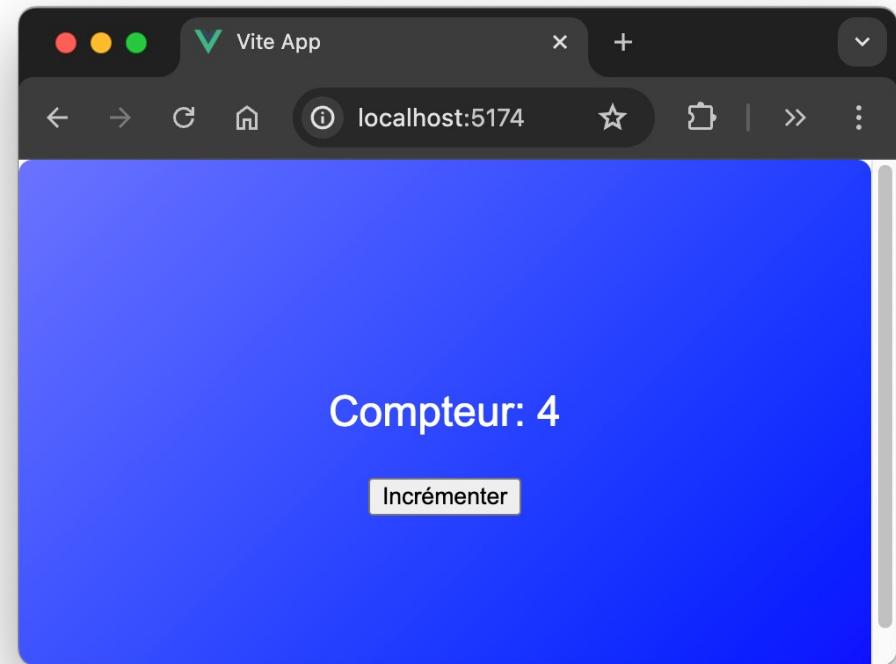
- Il est obligatoire d'utiliser `ref()`, sinon la données affichée ne s'actualise pas

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

const increment = () => {
  count.value++;
  console.log(count.value)
};

</script>

<template>
  <div class="container">
    <p>Compteur: {{ count }}</p>
    <button @click="increment">Incrémenter</button>
  </div>
</template>
```



Actualisation du DOM asynchrone

- Lorsque vous modifiez un état réactif, le DOM est automatiquement mis à jour
 - les mises à jour du DOM ne sont pas appliquées de manière synchrone
 - Vue s'assure que chaque composant soit mis à jour une seule fois, quel que soit le nombre de modifications d'état que vous avez effectuées
 - Vue met en mémoire tampon ces mises à jour jusqu'au prochain "tick" du cycle de mises à jour
- Il est possible d'attendre que la mise à jour du DOM soit terminée après un changement
 - <https://fr.vuejs.org/guide/essentials/reactivity-fundamentals#dom-update-timing>

```
import { nextTick } from 'vue'

async function increment() {
  count.value++
  await nextTick()
  // Maintenant le DOM est mis à jour
}
```

Références réactives avec `reactive()` 1/2

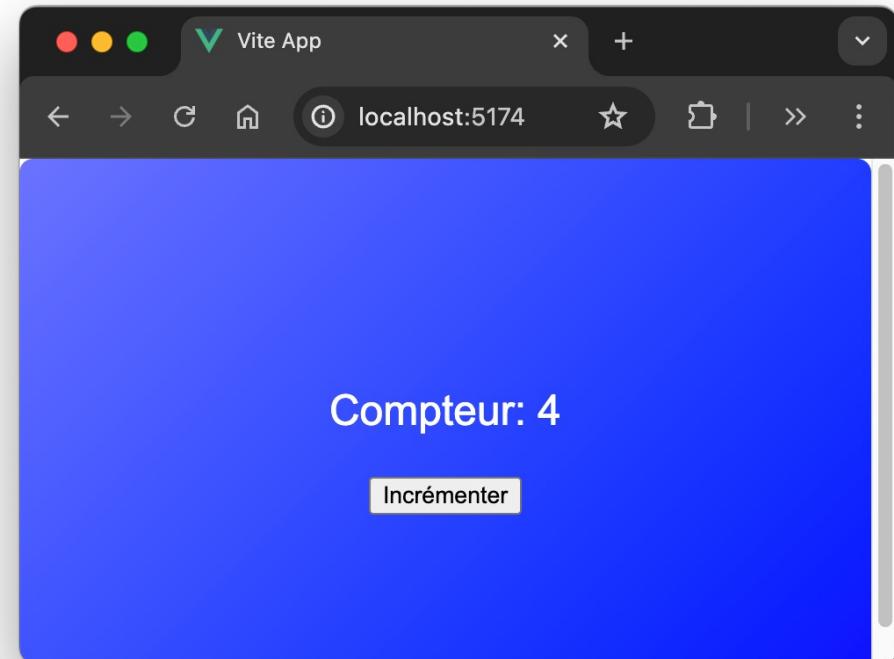
- `reactive()` permet de rendre un objet complet réactif

```
<script setup>
import {reactive} from 'vue';

const state = reactive({count: 0})

const increment = () => {
  state.count++;
  console.log(state.count);
};

</script>
<template>
  <div class="container">
    <p>Compteur: {{ state.count }}</p>
    <button @click="increment">Incrémenter</button>
  </div>
</template>
```



Références réactives avec **reactive()** 2/2

L'API **reactive()** a quelques limitations :

- <https://fr.vuejs.org/guide/essentials/reactivity-fundamentals.html#limitations-of-reactive>

Remplacement impossible de l'intégralité de l'objet

- **reactive()** fonctionne sur l'accès aux propriétés et doit donc conserver la même référence à l'objet



```
let state = reactive({count: 0})
// la référence précédente ({ count: 0 }) n'est plus suivie
// (la connexion de réactivité est perdue !)
state = reactive({ count: 1 })
```



```
const state = reactive({ count: 0 });
// Fonction pour mettre à jour les propriétés de l'objet réactif
const updateState = () => {
  state.count = 1; // Modification directe de la propriété
};
```

VUE.JS

Atelier 4 : Afficher tutoriels et commentaires



Atelier : Afficher tutoriels et commentaires 1/3

- À partir de l'application Tutorials créée précédemment, ajoutez un tableau contenant des tutoriels au composant tutorials
- Un tutoriel est composé d'un id, un titre, une description, un contenu, un auteur, une catégorie, une date de création, une liste de commentaires :
 - Un Auteur est composé d'un id, d'un nom, d'un prénom, d'un email
 - Une Catégorie représente une des catégories déclarée précédemment
- Les Commentaires peuvent être afficher grâce à un bouton
- Avec l'aide du framework CSS, affichez une liste des tutoriels dans le template du composant tutoriels :
 - Affichez la date sous un format lisible
 - Afficher le nom de l'auteur en majuscule

Atelier : Aperçu 2/3

Tutorials App

Initiation au langage Java
Lorem ipsum dolor sit amet, consectetur adipisciing elit
Jane WAAAAA j.wassa@tuto.fr

En ligne depuis le mercredi 2 décembre 2025

Commentaires 2

Super tuto ! Bravo Jane

Très intéressant, merci pour le partage

Details Partager

Initiation au macro VBA
Lorem ipsum dolor sit amet, consectetur adipisciing elit
Boris SAU b.sau@tuto.fr

En ligne depuis le mardi 16 septembre 2025

Commentaires 1

Excel est vraiment génial avec VBA

Details Partager

Afficher une liste avec Material Angular
Lorem ipsum dolor sit amet, consectetur adipisciing elit
John DOE j.doe@tuto.fr

En ligne depuis le mardi 30 janvier 2024

Commentaires

Details Partager

Atelier : Tri personnalisé 3/3

- Vous pouvez mettre en place un tri sur les tutoriels :
 - Plus récent
 - Plus ancien
 - De A à Z
 - De Z à A

Tutoriels :

Tri :
Plus récents ▾

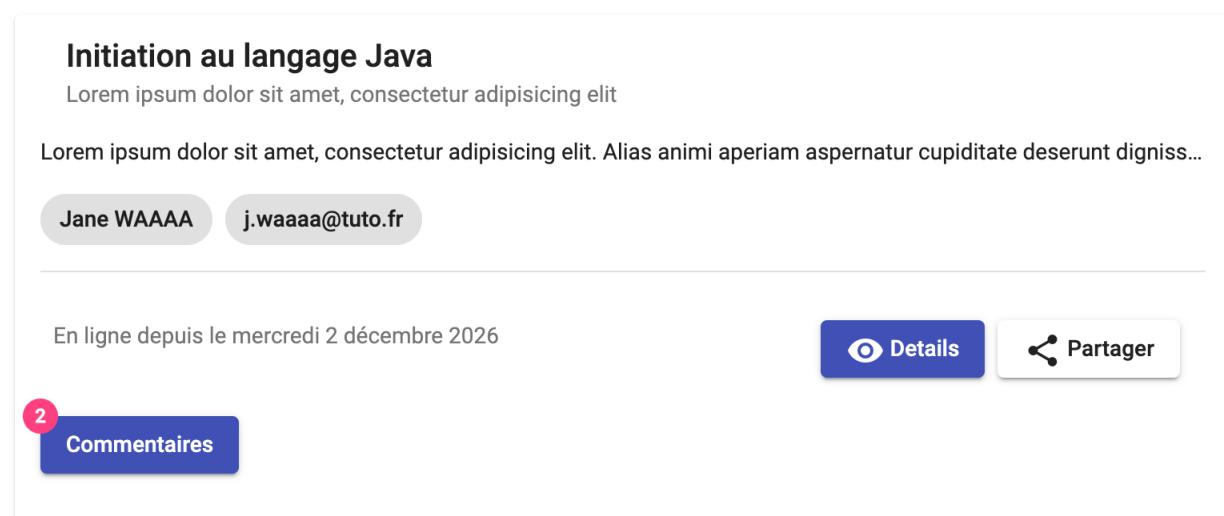


Initiation au langage Java
Lorem ipsum dolor sit amet, consectetur adipisicing elit
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Alias animi aperiam aspernatur cupiditate deserunt digniss...
Jane WAAAAA j.waaaa@tuto.fr

En ligne depuis le mercredi 2 décembre 2026

Commentaires 2

Details **Partager**



VUE.JS

Composant & Communication



Les props avec defineProps 1/2

- Utilisé pour déclarer une propriété d'entrée dans un composant
 - permet de passer des données du composant parent au composant enfant

```
<script setup>
import {defineProps} from 'vue';

const props = defineProps({
  color: {
    type: String,
    default: 'red',
  },
  circle: Boolean,
  rounded: Boolean
});
</script>
```

▼ Shape.vue

```
<template>
<section class="board">

<Shape
  name="shape 2"
  :circle="true"
  :rounded="true"
/>

</section>
</template>
```

▼ App.vue



<https://stackblitz.com/edit/vuejs-exemple-props-events>

Les props avec defineProps 2/2

- Lorsque vous passez une information à un composant, vous utilisez **name**

```
<Shape  
  name="shape 2"  
/>
```

- Lorsque vous passez une variable à un composant, vous utilisez **:name**

```
<script setup>  
import {defineProps} from 'vue';  
  
const shapeName = 'My shape';  
  
<template>  
<Shape  
  :name="shapeName"  
/>  
</template>
```

Les évènements avec defineEmits 1/3

- Il est possible d'émettre des évènements personnalisés à partir de composant enfants
 - <https://fr.vuejs.org/api/sfc-script-setup.html#defineprops-defineemits>
- Ces événements peuvent être émis vers le composant parent pour notifier des changements

```
<template>
  <div @click="handleClick">
    <span class="legend">{{ name }}</span>
  </div>
</template>
```

```
<script setup>
import {defineEmits, defineProps} from 'vue';

const props = defineProps(..);

const emit = defineEmits(['shapeClicked']);

const handleClick = () => {
  emit('shapeClicked', props.name);
};
</script>
```

Les évènements avec defineEmits 2/3

- Le composant parent :
 - Définit un évènement (buttonClicked) correspondant à l'évènement du composant enfant
 - L'évènement @Output permet de notifier le composant parent

```
<script setup>

const buttonClicked = (nameReceived) => {
  console.log(nameReceived)
};

</script>
```

```
<template>

<Shape
  name="Forme 1"
  @shapeClicked="buttonClicked"
/>

</template>
```

Les évènements avec defineEmits 3/3

Synthèse globale

```
<script setup>

const onClick = (nameReceived) => {
  console.log(nameReceived)
};

</script>

<template>

<Shape
  name="Forme 1"
  @shapeClicked="onClick"
/>

</template>
```

▼ Parent.vue

```
<template>
<div @click="handleClick">
  <span class="legend">{{ name }}</span>
</div>
</template>

<script setup>
import {defineEmits, defineProps} from 'vue';

const props = defineProps({ ... });

const emit = defineEmits(['shapeClicked']);

const handleClick = () => {
  emit('shapeClicked', props.name);
};

</script>
```

▼ Enfant.vue

VUE.JS

Atelier 5 : Mettre en place une architecture hiérarchique de composants



Atelier : Architecture hiérarchique 1/2

Mettre en place une architecture hiérarchique dans le composant tutorials :

- Créez un nouveau composant comments
 - possède un paramètre en entrée : un objet tutorial
- Le composant tutorials est le composant parent des composants comments
 - fournit à chaque enfant un objet commentaire

Atelier : Architecture hiérarchique 2/2

Composant comments

Composant tutorials

The screenshot shows a list of three tutorials. The first tutorial, titled "Initiation au langage Java", has a green box highlighting its comments section. The comments area contains two entries: "Super tuto ! Bravo Jane" and "Très intéressant, merci pour le partage". The second tutorial, "Initiation au macro VBA", also has a comments section with one entry: "Bonne initiation à VBA". The third tutorial, "Afficher une liste avec Material Angular", has no visible comments.

Initiation au langage Java
Lorem ipsum dolor sit amet, consectetur adipisicing elit
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Alias animi aperiam aspernatur cupiditate deserunt digniss...
Jane WAAAA j.waaaa@tuto.fr
En ligne depuis le mercredi 2 décembre 2026 [Details](#) [Partager](#)

Commentaires
2
Super tuto ! Bravo Jane
Très intéressant, merci pour le partage

Initiation au macro VBA
Lorem ipsum dolor sit amet, consectetur adipisicing elit
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Alias animi aperiam aspernatur cupiditate deserunt digniss...
Boris SAU b.sau@tuto.fr
En ligne depuis le mardi 16 septembre 2025 [Details](#) [Partager](#)

Afficher une liste avec Material Angular
Lorem ipsum dolor sit amet, consectetur adipisicing elit

VUE.JS

Le routeur



Introduction

- Depuis 1993, les applications web ont bien changé
 - Nombre important de données à afficher
 - Nombre croissant d'utilisateurs habituées à une réaction instantanée au clic
 - Chaque page était affichée entièrement à chaque rechargement du navigateur
 - Surcharge réseau et mauvaises performances
- La Single Page Application (SPA) : concept d'application web monopage
 - Permet d'augmenter l'expérience utilisateur
 - Application composée d'une seule page qui charge les éléments à la demande
 - Accessible sur tablette, smartphone, ordinateurs, avec une consommation réseau raisonnable
- Le routeur va permettre la navigation au sein de la SPA



Ajout du routeur à la création (create-vue)

- Lors de création d'un projet, create-vue propose d'ajouter un routeur
 - Un routeur est créé dans **router/index.js**



```
import { createRouter, createWebHistory } from 'vue-router'  
import HomeView from '../views/HomeView.vue'  
  
const router = createRouter({  
  history: createWebHistory(import.meta.env.BASE_URL),  
  routes: [ ... ]  
})  
export default router
```

- Ce routeur est utilisé dans **main.js**

```
import { createApp } from 'vue'  
import App from './App.vue'  
import router from './router'  
  
const app = createApp(App)  
app.use(router)  
app.mount('#app')
```

Ajout du routeur sur un projet existant

Il est également possible d'ajouter le routeur ultérieurement

- <https://router.vuejs.org/guide/>

1. Il sera alors nécessaire d'installer vue-router dans votre projet

```
$ npm install vue-router@4
```

2. Puis, vous pourrez créer l'instance de **router** avec **createRouter()**

- *Voir slide précédente*

3. Pour terminer, vous pourrez enregistrer le **router** comme un plugin

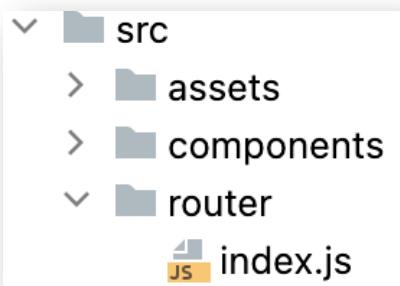
```
createApp(App)  
  .use(router)  
  .mount('#app')
```

Les liens 1/2

- Pour définir un lien, il est nécessaire d'utiliser le composant **RouterLink**

```
<nav>
  <RouterLink to="/">Home</RouterLink>
  <RouterLink to="/about">About</RouterLink>
</nav>
```

- Au clic sur le lien, VueJs parcourt le tableau à la recherche d'une occurrence de routes et effectue la navigation vers le composant chargé du rendu



```
import { createRouter, createWebHistory } from 'vue-router'
import HomeView from '../views/HomeView.vue'

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [ ... ]
})
export default router
```

Les liens 2/2

- Pour définir un lien avec des paramètres :

```
<nav>
<RouterLink :to="{name: 'category-details', params: { myCategoryId : category.id }}">
</RouterLink>
</nav>
```



Route active

- Il est possible de déterminer la route active avec la balise
 - Avec **exact-active-class**

```
<nav>
  <RouterLink to="/linkA" exact-active-class="active">Home</RouterLink>
  <RouterLink to="/linkB" exact-active-class="active">About</RouterLink>
</nav>
```

- Et avec un style CSS :

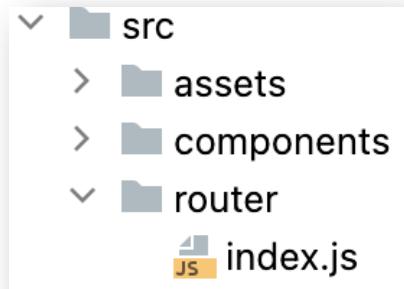
```
<style scoped>
  .active {
    border: solid;
  }
</style>
```



- <https://router.vuejs.org/guide/essentials/active-links.html>

Le fichier du routeur 1/2

- Le fichier du routeur exporte un tableau de routes



```
import { createRouter, createWebHistory } from 'vue-router'
import HomeView from '../views/HomeView.vue'

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/',
      name: 'home',
      component: HomeView
    }
  ]
})
export default router;
```

- Une route est un objet JavaScript qui contient une URL et un composant

Le fichier du routeur 2/2

- Les routes doivent être ajoutées dans le tableau

```
routes: [
  {
    path: '/linkA',
    component: ComponentA
  },
  {
    path: '/linkB',
    component: () => import('ComponentB.vue')
  }
]
```



localhost:5173/linkA

localhost:5173/linkB

- L'utilisation de **import** permet un **Lazy Loading**
 - <https://router.vuejs.org/guide/advanced/lazy-loading.html>
 - Réduit le bundle principal en séparant les composants importé dynamiquement
 - C'est une Promise asynchrone

Les routes nommées

- Lorsque vous crées des routes, vous pouvez les nommer
 - <https://router.vuejs.org/guide/essentials/named-routes.html>

```
routes: [
  {
    path: '/',
    name: 'home',
    component: HomeView
  }, ...
]
```

- Dans le code, cela permettra de naviguer vers une route nommée

```
<script setup>

import router from "@/router/index.js";
router.push({ name: 'home' })

</script>
```

Le Composant RouterView

- Le composant RouterView expose un emplacement qui peut être utilisé pour afficher les composants des routes
- VueJs traite les routes dans l'ordre du tableau
 - jusqu'à ce qu'il rencontre la route correspondante au RouterLink
 - Une fois trouvé, il n'évalue pas les autres routes du tableau

```
<nav>
  <RouterLink to="/" >Home</RouterLink>
  <RouterLink to="/about" exact-active-class="active">About</RouterLink>
</nav>

<RouterView/>
```

- Le contenu s'affichera dans à la place de RouterView
- <https://router.vuejs.org/guide/advanced/router-view-slot.html#RouterView-slot>

La route Joker

- Si le routeur ne trouve pas de correspondance, il faut afficher un 404
 - <https://router.vuejs.org/guide/essentials/dynamic-matching.html#Catch-all-404-Not-found-Route>

```
import NotFound from "@/components/NotFound.vue";

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    ....
    {
      path: '/:pathMatch(.*)*',
      component: NotFound
    }
  ]
})
export default router;
```



Les paramètres de routes 1/2

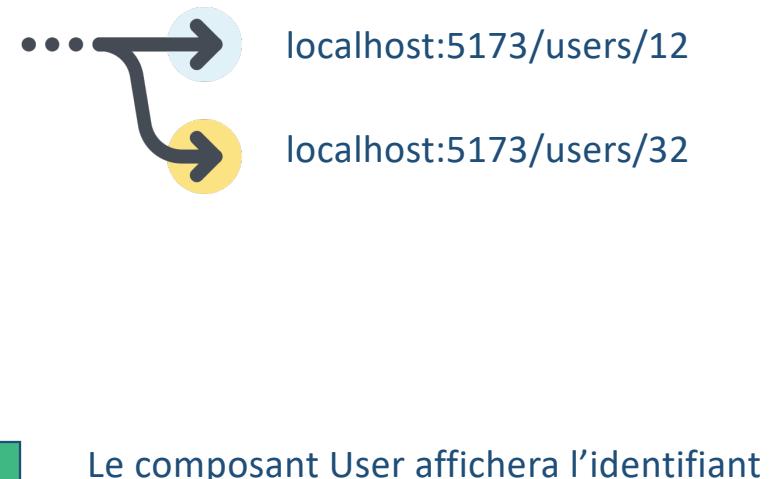
- Les routes peuvent définir des paramètres dynamique
 - <https://router.vuejs.org/guide/essentials/dynamic-matching.html>

```
import User from "@/components/User.vue";

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {path: '/users/:id', component: User}, ...
  ]
})
```

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

▼ User.vue



Les paramètres de routes 2/2

- Parfois, le changement du paramètre ne rafraîchira pas votre composant
 - Votre composant déjà affiché ne sera pas détruit puis réaffiché
- Un **watcher** devra être utilisé (<https://router.vuejs.org/guide/essentials/dynamic-matching.html>)

```
<template>
  <div>
    User {{ userId }}
  </div>
</template>
```

```
<script setup>
import {ref, watch} from 'vue'
import {useRoute} from 'vue-router'

const route = useRoute()
const userId = ref(route.params.id)
watch( () => route.params.id,
  (newId, oldId) => {
  userId.value = newId;
})
</script>
```

Navigation programmatique

- En plus d'utiliser la balise **<router-link>**, Il est possible de naviguer avec le code
 - <https://router.vuejs.org/guide/essentials/dynamic-matching.html>

```
<script setup>
import router from "@/router/index.js";
...
...
```

```
// avec un chemin
router.push('/users/eduardo')

// avec un objet contenant une chemin avec paramètre
router.push({ path: '/users/eduardo' })

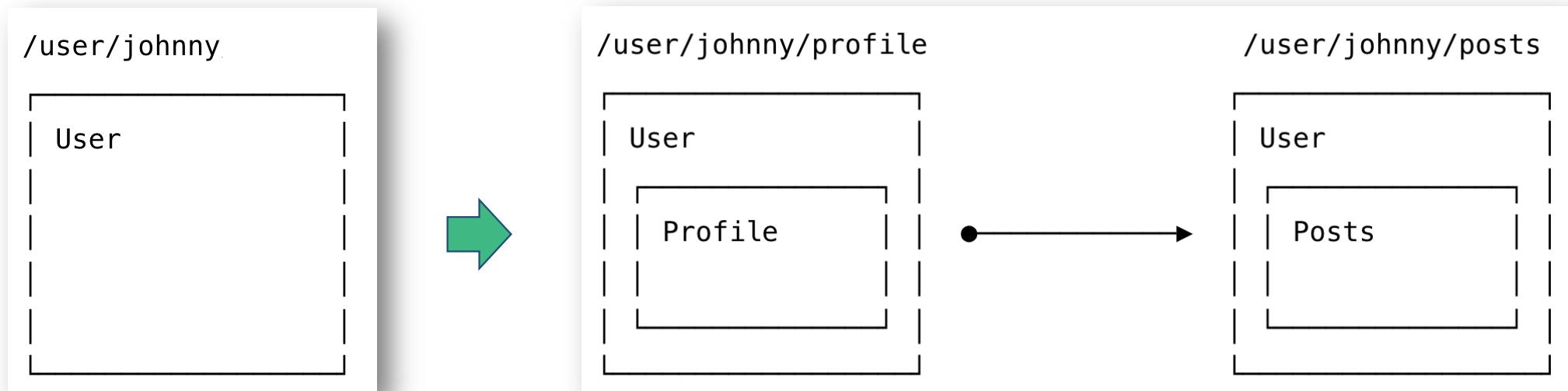
// avec des paramètres d'URL : /user/eduardo
router.push({ name: 'user', params: { username: 'eduardo' } })

// avec des paramètres de requête : /register?plan=private
router.push({ path: '/register', query: { plan: 'private' } })

// avec une ancre : /about#team
router.push({ path: '/about', hash: '#team' })
```

Les routes imbriquées 1/4

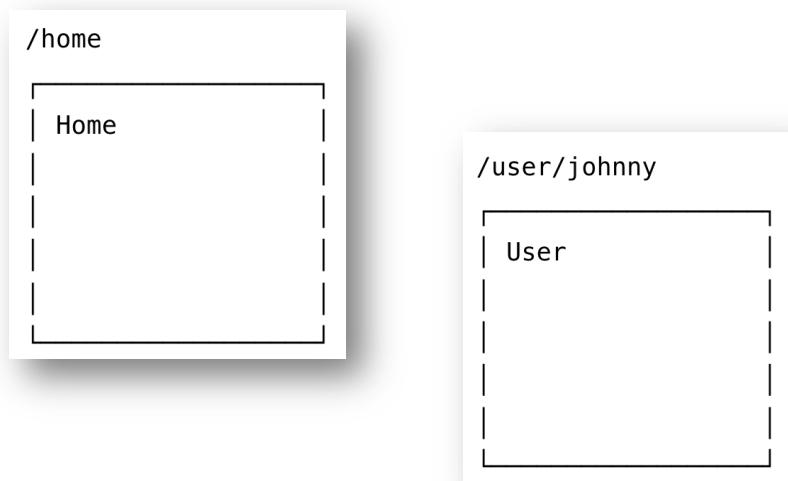
- Il est possible de définir une navigation entre des composants
- Il est aussi possible de définir des sous-navigation au sein d'un même composant
 - <https://router.vuejs.org/guide/essentials/nested-routes.html>



Les routes imbriquées 2/4

Exemple d'un routeur

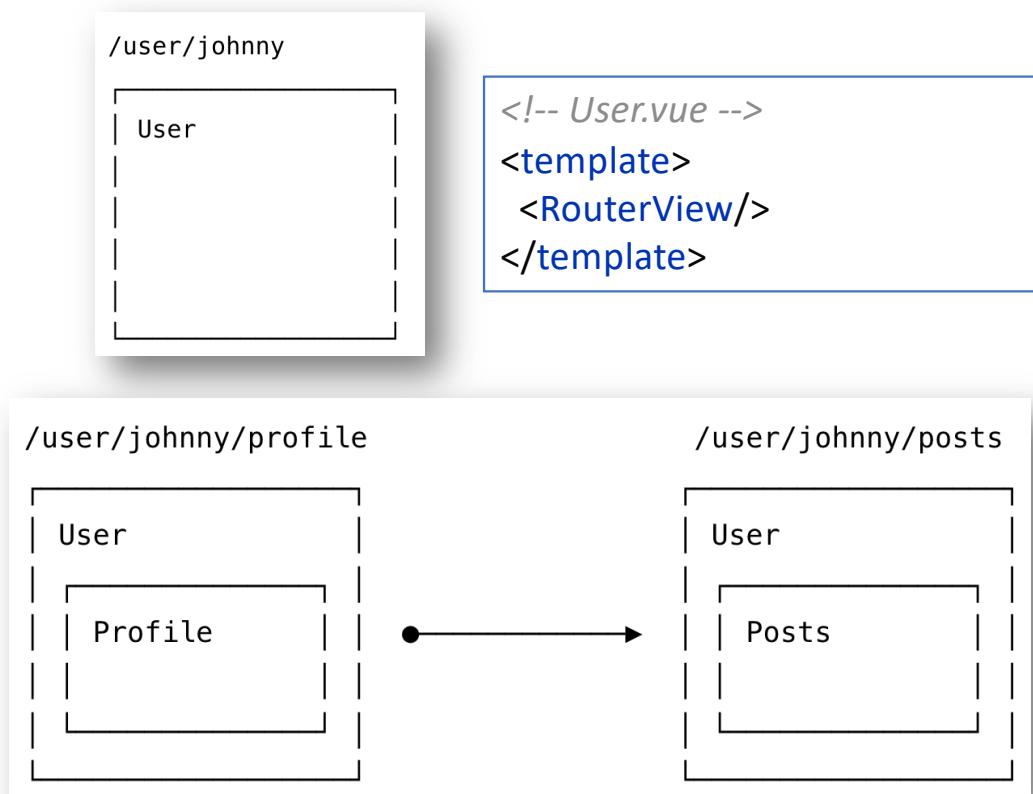
```
<!-- App.vue -->
<template>
  <RouterView/>
</template>
```



```
const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/',
      name: 'home',
      component: Home
    },
    {
      path: '/users/:id',
      component: User
    }
  ]
})
export default router;
```

Les routes imbriquées 3/4

Exemple d'un routeur avec imbrication



```
const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/user/:id',
      component: User,
      children: [
        {
          path: 'profile',
          component: UserProfile,
        },
        {
          path: 'posts',
          component: UserPosts,
        },
      ],
    }
  ]
})
```

Les routes imbriquées 4/4

- Le nommage des routes est utilisables avec les routes imbriquées

```
const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/user/:id',
      name: 'user-parent',
      component: User,
      children: [{ path: "", name: 'user', component: UserHome }],
    },
  ]
})
```

Démo : <https://stackblitz.com/edit/vuejs-exemple-routeur-imbrique>

Mise en page et vues nommées

- Il est possible de découper votre page en plusieurs <router-view>
 - <https://router.vuejs.org/guide/essentials/named-views.html>
- Vous allez pouvoir nommer vos router-view :

```
<router-view class="view left-sidebar" name="left" />
<router-view class="view main-content" />
<router-view class="view right-sidebar" name="right" />
```

Démo : <https://stackblitz.com/edit/vuejs-exemple-multiples-vues>

```
export const router = createRouter({
  history: createWebHistory(),
  routes: [
    ...
    {
      path: '/other',
      components: {
        default: MainComponent,
        left: NavComponent,
        right: WeatherComponent,
      },
    },
  ],
})
export default router;
```

Passer des props

- Il est possible de passer des paramètres à un composant
 - Plutôt que paramétriser le routeur sur une route précise (= car fort couplage)
 - <https://router.vuejs.org/guide/essentials/passing-props.html>
- Définir **props** à **true** permet de récupérer l'id en tant que props
 - Sans utiliser et dépendre du routeur

```
const routes = [
  {
    path: '/user/:id',
    component: User,
    props: true
  }
]
```



```
<!-- User.vue -->
<script setup>
defineProps({
  id: String
})
</script>

<template> User {{ id }} </template>
```

VUE.JS

Atelier 6 : Mettre en place une navigation



Atelier : navigation 1/2

- Mettez en place une barre de navigation sur la page d'accueil, avec les liens suivants :
 - Page d'accueil
 - Voir les catégories
 - Voir les tutoriels
 - Créer une catégorie
 - Créer un tutoriel
- Créez les composants supplémentaires que vous jugerez nécessaires et utilisez les balises **RouterLink** et **RouterView**

Atelier : navigation 2/2

Tutorials App Catégories Tutoriels Ajouter Tutoriel

Tutoriels :

Tri :
Plus récents

Initiation au langage Java
Lorem ipsum dolor sit amet, consectetur adipisicing elit
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Alias animi aperiam aspernatur cupiditate deserunt digniss...
Jane WAAAA j.waaaa@tuto.fr

En ligne depuis le mercredi 2 décembre 2026

[Details](#) [Partager](#)

2 [Commentaires](#)

Initiation au macro VBA
Lorem ipsum dolor sit amet, consectetur adipisicing elit
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Alias animi aperiam aspernatur cupiditate deserunt digniss...
Boris SAU b.sau@tuto.fr

En ligne depuis le mardi 16 septembre 2025

[Details](#) [Partager](#)

VUE.JS

Le protocole HTTP



Le protocole HTTP (L'Hypertext Transfer Protocol) 1/2

- HTTP est un protocole qui autorise le transfert de fichiers sur le web
 - typiquement entre un navigateur et un serveur afin que des utilisateurs puissent les consulter
- Dans le cadre d'une URI, la partie "**http://**" s'appelle le "**schema**"
 - commence la plupart du temps au début d'une adresse comme <https://www.sauvageboris.fr>
 - "**https://**" indique au navigateur de requêter le document au travers du protocole HTTP
- HTTP est textuel (communication en texte clair) et sans état (aucune communication n'est au courant des communications précédentes)
 - Utilisable également dans le cadre de services REST pour la communication de serveur à serveur

Le protocole HTTP (L'Hypertext Transfer Protocol) 2/2

- HTTP permet de récupérer des ressources telles que des documents HTML
 - Il est à la base de tout échange de données sur le Web
- C'est un protocole de type client-serveur :
 - les requêtes sont initiées par le destinataire (généralement un navigateur web)
- Un document HTML est construit à partir de différents sous-documents qui sont récupérés
 - du texte, des fichiers CSS, des scripts et bien plus
- Les clients et serveurs communiquent par l'échange de messages individuels
 - Les requêtes et les réponses

HTTP : les échanges client/serveur 1/2

- Lorsqu'un client veut communiquer avec un serveur, il réalise les étapes suivantes :

1. Il ouvre une connexion TCP vers le serveur

2. Il envoie un message HTTP

```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

3. Il lit la réponse envoyée par le serveur

```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html
<!DOCTYPE html.....
```

4. Il ferme ou réutilise la connexion pour les requêtes suivantes

HTTP : les échanges client/serveur 2/2

- Une requête comprend les éléments suivants :

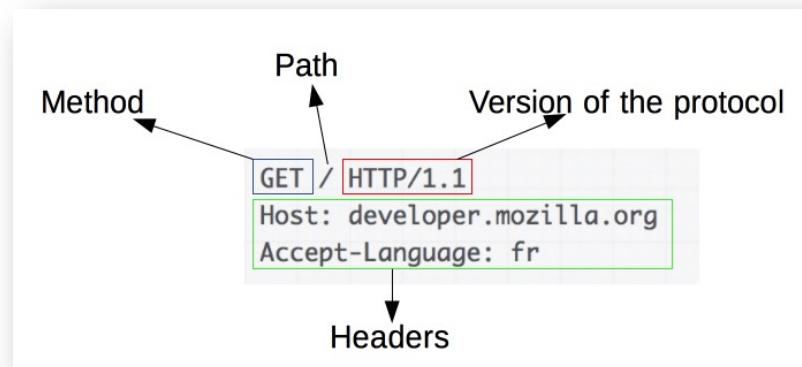
Un verbe HTTP qui définit l'opération que le client souhaite effectuer

l'URL de la ressource

La version du protocole HTTP

Les en-têtes optionnels qui transmettent des informations pour les serveurs

Un corps pour certaines méthodes comme POST



- Une réponse comprend les éléments suivants:

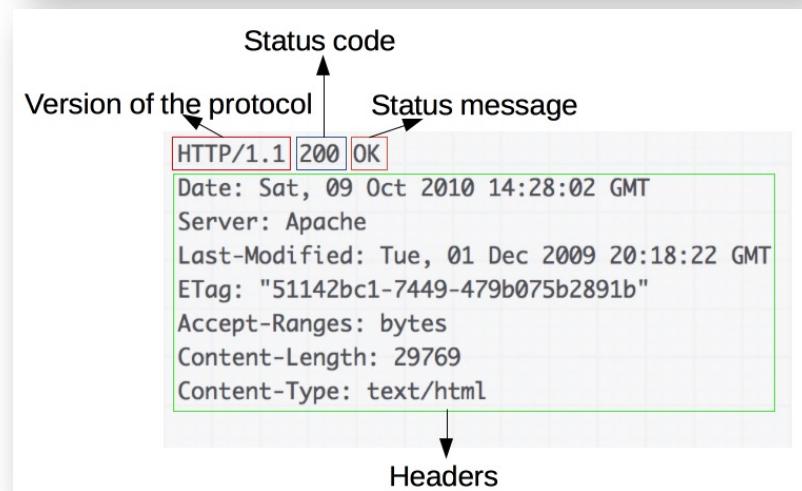
La version du protocole HTTP utilisé

Un code de statut, qui indique si la requête a réussi ou non

Un message de statut qui est une description rapide

Les en-têtes HTTP, comme pour les requêtes

Éventuellement un corps contenant la ressource récupérée



HTTP : les méthodes

- HTTP définit des méthodes de requête qui indiquent l'action que l'on souhaite réaliser sur la ressource indiquée
 - Aussi appelé verbes HTTP
- **GET** demande une représentation de la ressource spécifiée
 - Les requêtes GET doivent uniquement être utilisées afin de récupérer des données
- **POST** est utilisée pour envoyer une entité vers la ressource indiquée
 - Cela entraîne généralement un changement d'état ou des effets de bord sur le serveur
- **PUT** remplace toutes les représentations actuelles de la ressource visée par le contenu de la requête
- **DELETE** supprime la ressource indiquée
- **PATCH** est utilisée pour appliquer des modifications partielles à une ressource

HTTP : les codes de réponse

- Les codes de statut de réponse HTTP indiquent si une requête HTTP a été exécutée avec succès ou non
- Les réponses sont regroupées en 5 classes :
 - Les réponses informatives (100 - 199)
 - Les réponses de succès (200 - 299)
 - Les messages de redirection (300 - 399)
 - Les erreurs du client (400 - 499)
 - Les erreurs du serveur (500 - 599)
- <https://developer.mozilla.org/fr/docs/Web/HTTP>Status>
- <https://httpwg.org/specs/rfc9110.html#status.codes>
- <https://httpstatusdogs.com/>
- <https://http.cat/>

VUE.JS

VueJS et API Rest



API REST 1/2

- Une API RESTful (Representational State Transfer)
- Ensemble de principes d'architecture pour la conception et le développement d'interfaces d'application web (API) qui utilisent le protocole HTTP de manière cohérente
- Ces principes sont basés sur le modèle du www (World Wide Web)
 - tel qu'il est utilisé par les navigateurs pour accéder aux sites web
- Ressources importantes
 - <https://www.restapitutorial.com/>
 - <http://blog.octo.com/designer-une-api-rest/>
 - <https://github.com/interagent/http-api-design>
 - <https://martinfowler.com/articles/richardsonMaturityModel.html>

API REST 2/2

Exemple d'une API REST

Une API qui concerne une liste de contacts pourrais renvoyer ces données :

Avec HTTP GET sur
<http://url-du-serveur/api/contacts>

```
{  
  "contacts": [  
    {  
      "id": 1,  
      "name": "John Doe",  
      "email": "john@example.com"  
    },  
    {  
      "id": 2,  
      "name": "Jane Smith",  
      "email": "jane@example.com"  
    }  
  ]  
}
```

Principes API Rest 1/3

Les verbes HTTP permettent de donner du sens à vos demandes

- Généralement, les 4 verbes HTTP principaux sont utilisés comme suit :
 - **POST** : Créez une nouvelle ressource
 - Également un verbe fourre-tout pour les opérations qui ne rentrent pas dans les autres catégories
 - **GET** : Lire une ressource spécifique (par un identifiant) ou une collection de ressources
 - **PUT** : Mettre à jour une ressource spécifique (par un identifiant) ou une collection de ressources
 - Également utilisable pour créer une ressource spécifique si l'identifiant de la ressource est connu au préalable
 - **DELETE** : Supprimer une ressource spécifique par un identifiant
- Ces opérations sont souvent appelé CRUD (**C**reate **R**ead **U**pdate **D**elete)

Principes API Rest 2/3

- **Les codes de réponse HTTP permettent d'indiquer l'état de la réponse**
 - Les codes d'état de réponse font partie de la spécification HTTP
- **200 OK**
 - Code de statut de réussite général. C'est le code le plus courant. Utilisé pour indiquer le succès
- **201 CREATED**
 - La création a réussi (via POST ou PUT). Définissez l'en-tête Location pour qu'il contienne un lien vers la ressource nouvellement créée (sur POST). Le contenu du corps de la réponse peut être présent ou non
- **204 NO CONTENT**
 - Indique le succès avec un corps vide. Souvent utilisé pour les opérations DELETE et PUT
- **500 INTERNAL SERVER ERROR**
 - Ne renvoyez jamais ceci intentionnellement. Erreurs que le consommateur ne peut pas résoudre de son côté

Principes API Rest 3/3

- **400 BAD REQUEST**
 - Erreur générale lorsque l'exécution de la demande entraînerait un état invalide. (validation de domaine, données manquantes, etc)
- **401 UNAUTHORIZED**
 - Réponse au code d'erreur en cas de jeton d'authentification manquant ou non valide
- **403 FORBIDDEN**
 - Code d'erreur lorsque l'utilisateur n'est pas autorisé à effectuer l'opération ou que la ressource n'est pas disponible
- **404 NOT FOUND**
 - La ressource demandée n'est pas trouvée ou n'existe pas. Utilisé aussi pour masquer une ressource si (401 ou 403)
- **405 MÉTHODE NON AUTORISÉE**
 - Indique que l'URL demandée existe, mais que la méthode HTTP demandée n'est pas applicable. Par exemple, POST /users/12345
- **409 CONFLICT**
 - Chaque fois qu'un conflit de ressources serait provoqué par la satisfaction de la demande. Les entrées en double par exemple

Vuejs – Consommer une API JSON

- VueJs est un Framework idéal pour consommer une API RESTful

Il est possible de consommer une API REST de plusieurs façon :

- Avec l'API javascript Fetch
 - https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch
- Avec la bibliothèque axios
 - <https://axios-http.com/fr/docs/intro>
- Nous verrons deux exemples ci-dessous

Exemple avec Fetch API

- Il n'est pas nécessaire d'installer de bibliothèque
 - L'API Fetch est un client HTTP basé sur les promesses et compatibles avec Node.js

```
try {  
  const response = await fetch('https://your-api');  
  const contacts = await response.json();  
  console.log(contacts)  
} catch (error) {  
  console.error(error);  
}
```

```
fetch('https://your-api')  
  .then(response => response.json())  
  .then(contacts => {  
    console.log(contacts);  
  })  
  .catch(error => {  
    console.error(error);  
});
```

Version `async/await`
ECMAScript 2017 (ES8)

Version ECMAScript 2015 (ES6)

Exemple avec Axios

- Il est nécessaire d'installer la bibliothèque axios
 - Axios est un client HTTP basé sur les promesses et compatibles avec Node.js

```
$ npm install axios
```

```
import axios from 'axios';
try {
  const response = await axios.get('https://your-api');
  const contacts = response.data;
  console.log(contacts);
} catch (error) {
  console.error(error);
}
```

Version `async/await`
ECMAScript 2017 (ES8)

```
axios.get('https://your-api')
  .then(response => {
    const contacts = response.data;
    console.log(contacts);
  })
  .catch(error => {
    console.error(error);
});
```

Version ECMAScript 2015 (ES6)

VUE.JS

Les services



Les services

- Il est possible de centraliser les appels réseaux dans des services
- Ce sont des classes Javascript qui centralisent les méthodes du CRUD
- L'avantage est de pouvoir réutiliser une même méthode dans plusieurs composants

```
class PostService {  
    async getAll() {}  
    async get(id) {}  
    async create(data) {}  
    async update(id, data) {}  
    async delete(id) {}  
    async findByTitle(title) {}  
}  
export default new PostService();
```

Exemple 1/2

- Un exemple avec la récupération et l'affichage de posts

```
class PostService {  
  
    async getAll() {  
        const response = await fetch('https://jsonplaceholder.typicode.com/posts');  
        return await response.json();  
    }  
}  
export default new PostService();
```

▼ services
post.service.js

Exemple 2/2

- Le service précédent est utilisable au sein des composants :

```
<script setup>
import {onMounted, ref} from "vue";
import postService from "@/services/post.service.js";

const posts = ref(null);
const fetchPosts = async () => {
  try {
    posts.value = await postService.getAll();
  } catch (error) {
    console.error('Failed to load posts:', error);
  }
};

onMounted(() => fetchPosts());
</script>
```

```
<template>
<div>
  <h1>Posts</h1>
  <ul v-if="posts">
    <li v-for="post in posts" :key="post.id">
      <h2>{{ post.title }}</h2>
      <p>{{ post.body }}</p>
    </li>
  </ul>
</div>
</template>
```

VUE.JS

Atelier 7 : Mettre en place un service



Atelier : service et appels réseaux

- Créez deux services pour effectuer la gestion des catégories et des tutoriels
 - Les composants ne doivent plus contenir de tableau JSON
- Mise en place de l'API Rest avec json-server
- Implémentez les deux appels GET pour afficher les catégories et les tutoriels

VUE.JS

Les formulaires



Les formulaires

Pour gérer des formulaires, il faut synchroniser l'état des éléments de saisi avec l'état correspondant en Javascript

- <https://fr.vuejs.org/guide/essentials/forms>

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

const message = ref('')
</script>

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

Message is:

edit me

- **v-model** nous aide à gérer cette liaison bidirectionnelle facilement

Les cases à cocher 1/2

- Il est possible d'utiliser une case à cocher

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

const checked = ref(true)
</script>

<template>
<input type="checkbox" id="checkbox" v-model="checked" />
<label for="checkbox">{{ checked }}</label>
</template>
```



Les cases à cocher 2/2

- Il est aussi possible d'utiliser plusieurs cases à cocher

```
<script setup>
import { ref } from 'vue'
const checkedNames = ref([])
</script>

<template>
<div>Checked names: {{ checkedNames }}</div>
<input type="checkbox" id="jack" value="Jack" v-model="checkedNames" />
<label for="jack">Jack</label>

<input type="checkbox" id="john" value="John" v-model="checkedNames" />
<label for="john">John</label>

<input type="checkbox" id="mike" value="Mike" v-model="checkedNames" />
<label for="mike">Mike</label>
</template>
```

Checked names: ["Jack", "John"]
 Jack John Mike

Les boutons radio

- Il est possible d'utiliser des boutons radio

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

const picked = ref('One')
</script>

<template>
<div>Picked: {{ picked }}</div>

<input type="radio" id="one" value="One" v-model="picked" />
<label for="one">One</label>

<input type="radio" id="two" value="Two" v-model="picked" />
<label for="two">Two</label>
</template>
```

Picked: One
 One Two

Les listes déroulantes

- Il est possible d'utiliser des listes déroulantes

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

const selected = ref('')
</script>

<template>
Selected: {{ selected }}

<select v-model="selected">
  <option disabled value="">Please select one</option>
  <option>A</option>
  <option>B</option>
  <option>C</option>
</select>
</template>
```

Selected: Please select one ▾

Selected ✓ Please select one

A
B
C

Formulaire : Exemple complet

- <https://stackblitz.com/edit/vuejs-exemple-formulaire>

Exemple de Formulaire

Civilité Prénom Nom

Date d'anniversaire Email

Mot de passe Confirmation

Accepter les CGU

S'inscrire **Réinitialiser**



Atelier ensemble : ajout d'une catégorie

- Ajoutons une catégorie en utilisant l'API Rest (en HTTP POST)

VUE.JS

Atelier 8 : Mettre en place un formulaire d'ajout



Atelier : formulaire d'ajout 1/2

- Mettre en place un formulaire pour ajouter un tutoriel
- Commencez par définir un formulaire avec les champs :
 - Titre, Description, Contenu
 - Date de création
- Lors de la soumission du formulaire, vous pourrez :
 - Récupérer les données du formulaire
 - Envoyer les données à l'API grâce à votre tutorialService
 - Rediriger l'utilisateur



Atelier : formulaire d'ajout 2/2

- Ensuite, envisagez de compléter le formulaire avec la sélection d'une catégorie

Ajouter un tutoriel

Titre

Description

Choisir une catégorie

Tutoriel

Choisir une date

MM/DD/YYYY

Ajouter

Titre
Créer un formulaire en Javascript

Description
Mise en place d'un formulaire réactif avec ReactiveFormsModule

Choisir une catégorie
Angular

Tutoriel
Mettre en place un formulaire pour ajouter un tutoriel
Le formulaire devra être réactifs (ReactiveFormsModule)
Utilisez formGroup, formControlName et (ngSubmit)
Commencez par définir un formulaire avec les champs :
Titre, Description, Contenu (<https://material.angular.io/components/form-field/overview>)
Date de création (<https://material.angular.io/components/datepicker/overview>)
Vous pouvez créer une interface CreateTutorial contenant les informations nécessaires à envoyer

Choisir une date
30/01/2030

MM/DD/YYYY

Ajouter



VUE.JS

La validation des formulaires



La validation des formulaires

- Les formulaires doivent pouvoir être validés
- Vuejs se focalise sur une validation manuelle
 - En javascript natif, il est possible de valider chaque données saisie
- Il existe des bibliothèques qui permettent d'automatiser cette partie :
 - Vuevalidate (<https://github.com/vuelidate/vuelidate>)
 - VeeValidate (<https://vee-validate.logaretm.com/v4/>)
- **Notez que l'utilisation de bibliothèques externes augmente la taille de votre application**



VeeValidate : Exemple complet

- <https://stackblitz.com/edit/vuejs-exemple-validation-formulaire>

Exemple de Formulaire

Civilité ⓘ

Civilité est requise

Prénom ⓘ

Prénom est requis

Nom ⓘ

Nom est requis

Date d'anniversaire jj/mm/aaaa ⓘ

Date de naissance est requise

Email ⓘ

Email est requis

Mot de passe ⓘ

Mot de passe est requis

Confirmation ⓘ

Confirmation du mot de passe est requise

Accepter les CGU
Vous devez accepter les conditions générales

S'inscrire **Réinitialiser**



VUE.JS

Atelier 9 : Mettre en place une validation



Atelier : validation du formulaire d'ajout

Sur le formulaire d'ajout de tutoriel, ajouter des contraintes sur le valeurs

- Affichez des messages d'erreurs si vous champs ne sont pas valide
- Désactivez le bouton d'ajout si le formulaire est invalide
- Si vous le souhaitez, vous pouvez créez un validateur pour la date

Ajouter un tutoriel

Titre *
Le titre ne doit pas être vide

Description *
La description ne doit pas être vide

Choisir une catégorie *
La categorie ne doit pas être vide

Tutoriel *
Le tutoriel ne doit pas être vide

Choisir une date *
17/10/2023
La date ne peut pas être passée

Ajouter

Ajouter un tutoriel

Titre
Créer un formulaire en Javascript

Description
Mise en place d'un formulaire reactif avec ReactiveFormsModule

Choisir une catégorie
Angular

Tutoriel
Mettre en place un formulaire pour ajouter un tutoriel
Le formulaire devra être réactifs (ReactiveFormsModule)
Utilisez formGroup, formControlName et (ngSubmit)
Commencez par définir un formulaire avec les champs :
Titre, Description, Contenu (<https://material.angular.io/components/form-field/overview>)
Date de création (<https://material.angular.io/components/datepicker/overview>)

Vous pouvez créer une interface CreateTutorial contenant les informations nécessaires à envoyer

Choisir une date
30/01/2030
MM/DD/YYYY

Ajouter



VUE.JS

Les tests



Les tests

- Les tests automatisés vous aident à prévenir les régressions et force à décomposer
 - Découper votre application en fonctions, modules, classes et composants testables
- Il est important que vous puissiez détecter ces problèmes avant de livrer
- Il est recommandé de commencer à écrire des tests dès que vous le pouvez
 - Plus vous attendrez, plus votre application aura des dépendances : il sera difficile de commencer

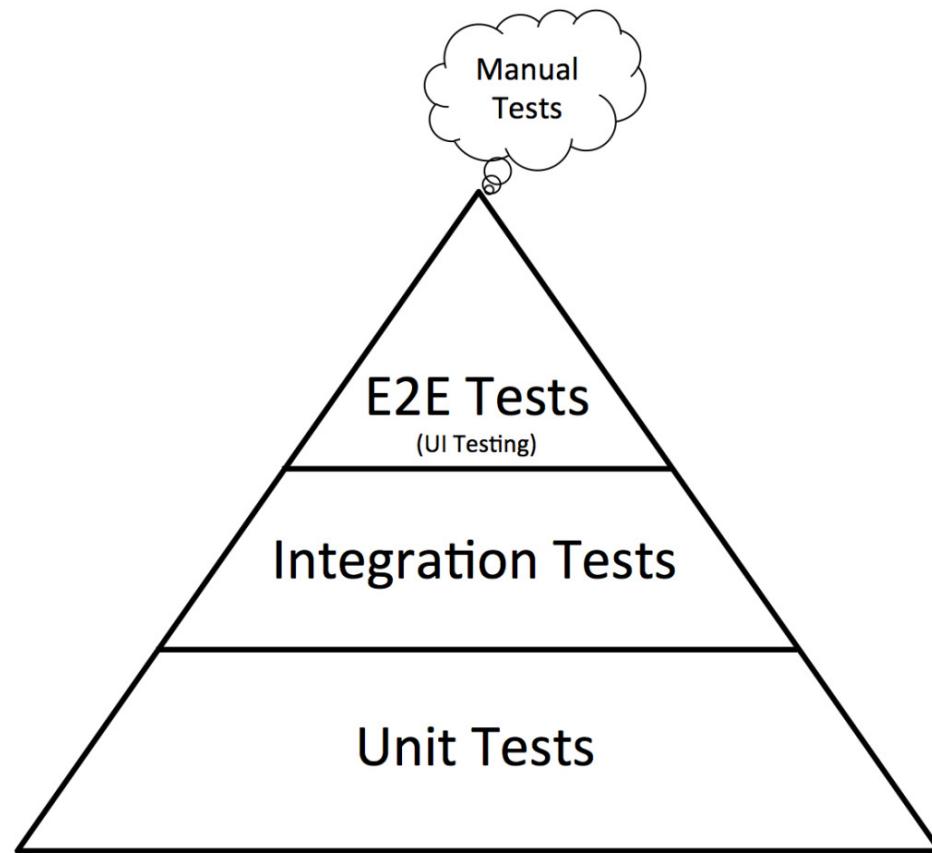
Types de tests 1/2

Vous pouvez mettre en place les types de tests suivants :

- **Unitaire**
 - Vérifie que les entrées d'une fonction, classe, ou composable donné produisent les sorties ou effets de bord attendus
- **Composant**
 - Vérifie que le montage, le rendu, les interactions et le comportement d'un composant ont lieu comme prévu
 - Ces tests exercent plus de code que des tests unitaires, sont plus complexes et requièrent plus de temps pour s'exécuter
- **End-to-end**
 - Vérifie des fonctionnalités qui traversent plusieurs pages et émettent des vraies requêtes réseau sur votre application Vue construite pour la production
 - Ces tests impliquent souvent la mise en place d'une base de données ou d'un autre backend

Chaque type de test joue un rôle dans la stratégie de test de votre application, et vous protégera de différents problèmes

Types de tests 2/2



Tests unitaires : introduction 1/4

- Écrits pour vérifier que des petites unités de code isolées fonctionnent comme prévu
- Couvre généralement une seule fonction, classe, composable ou module
- Concentré sur l'exactitude logique et ne concernent qu'une petite partie des fonctionnalités globales de l'application
- Peuvent simuler de grandes parties de l'environnement de votre application
 - par exemple : l'état initial, les classes complexes, les modules tierce partie et les requêtes réseau
- Déetectent des problèmes concernant la logique métier d'une fonction et son exactitude logique

Tests unitaires : exemple 2/4

- Cette fonction **increment** est autonome et facile de appeler
 - Si une assertion échoue, le problème est contenu dans la fonction **increment**

```
// helpers.js
export function increment(current, max = 10) {
  if (current < max) {
    return current + 1
  }
  return current
}
```

```
// helpers.spec.js
import { increment } from './helpers'

describe('increment', () => {
  test('increments the current number by 1', () => {
    expect(increment(0, 10)).toBe(1)
  })

  test('does not increment the current number over the max', () => {
    expect(increment(10, 10)).toBe(10)
  })

  test('has a default max of 10', () => {
    expect(increment(10)).toBe(10)
  })
})
```

Tests unitaires : mise en place 3/4

- À l'installation avec **create-vue**, le **Framework de test Vitest** est proposé



- Une fois installé, la commande **npm run test:unit** est disponible

```
mbp-de-boris:tester sauvageb$ npm run test:unit

> tester@0.0.0 test:unit
> vitest

DEV v1.6.0 /Users/sauvageb/Desktop/vuejs/tester

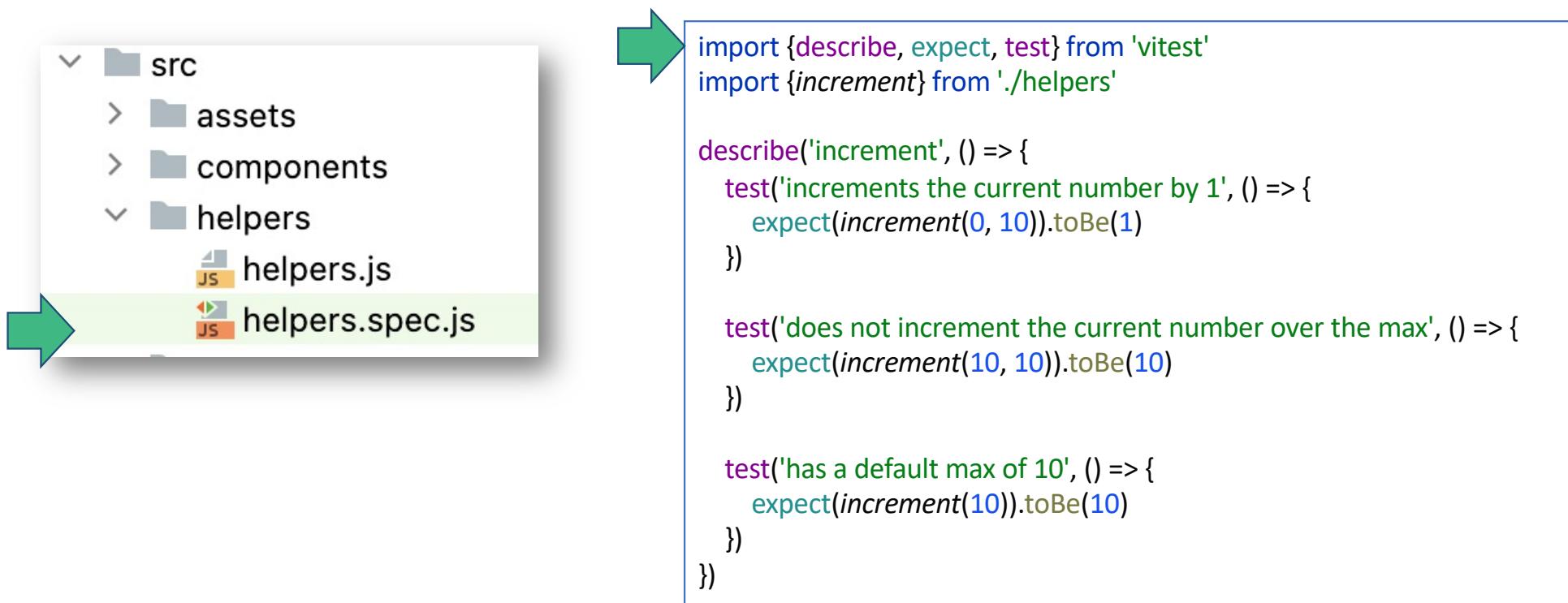
✓ src/helpers/helpers.spec.js (3)
✓ src/components/_tests_/HelloWorld.spec.js (1)

Test Files 2 passed (2)
Tests 4 passed (4)
Start at 20:46:45
Duration 1.26s (transform 83ms, setup 0ms, collect 197ms, tests 29ms, environment 1.49s, prepare 224ms)

PASS Waiting for file changes...
press h to show help, press q to quit
```

Tests unitaires : mise en place 4/4

- Vous pouvez ainsi créer des fichiers et leurs tests associés



Composables et Composants

Avec VueJs, Il existe 2 cas où vous testez unitairement des fonctionnalités :

- Composables
 - <https://fr.vuejs.org/guide/reusability/composables.html>

Les composables sont des fonctions spécifiques à Vue qui exploitent la Composition API (Vue 3)

Ils permettent d'extraire et de réutiliser la logique d'état et d'effet dans vos composants Vue

- Composants
 - <https://fr.vuejs.org/guide/scaling-up/testing#unit-testing-components>

Les composants sont les éléments de base de toute application Vue.js. Ils encapsulent le rendu, la logique de gestion de l'état et les interactions utilisateur

Tester un composant 1/2

- Un composant peut être testé de deux façons :

Boîte blanche : Test unitaire

- Ils ont "conscience" des détails d'implémentation et des dépendances du composant
- Ils se concentrent sur l'isolation du composant testé
- Ils impliquent en général de **simuler** certains sinon tous les enfants de votre composant ainsi que d'initialiser l'état de plugins et dépendances (ex. Pinia)

Boîte noire : Test de composant

- Ils n'ont pas "conscience" des détails d'implémentation du composant
- Ils simulent le moins possible afin de tester l'intégration de vos composants et le système entier
- Ils font généralement le rendu HTML de l'ensemble des sous-composants
- Sont considérés plus comme un "**test d'intégration**"

Tester un composant 2/2

- La configuration officielle créée par create-vue est basée sur Vite
 - Il est recommandé d'utiliser un framework basé sur Vite
- **Vitest** est le framework de test unitaire conçu par l'équipe Vue / Vite
 - Il s'intègre aux projets basés sur Vite avec un minimum d'effort et est ultrarapide
- **Jest** est un framework de test unitaire populaire
 - Pas recommandé sauf en cas de migration d'un projet existant
 - <https://fr.vuejs.org/guide/scaling-up/testing#unit-testing-components>

Les Test End-to-End 1/2

- Les tests unitaires offrent aux développeurs un certain degré de confiance
- Les tests End-to-end (E2E) offrent une couverture sur l'utilisation réelle
 - Ce qui est sans doute l'aspect le plus important d'une application
- ils détectent des problèmes critiques qui peuvent être impossibles à détecter avec des tests unitaires ou des tests de composants
- Les tests End-to-end reposent entièrement sur le test de votre application
 - en naviguant dans des pages entières dans un navigateur réel

Les Test End-to-End 2/2

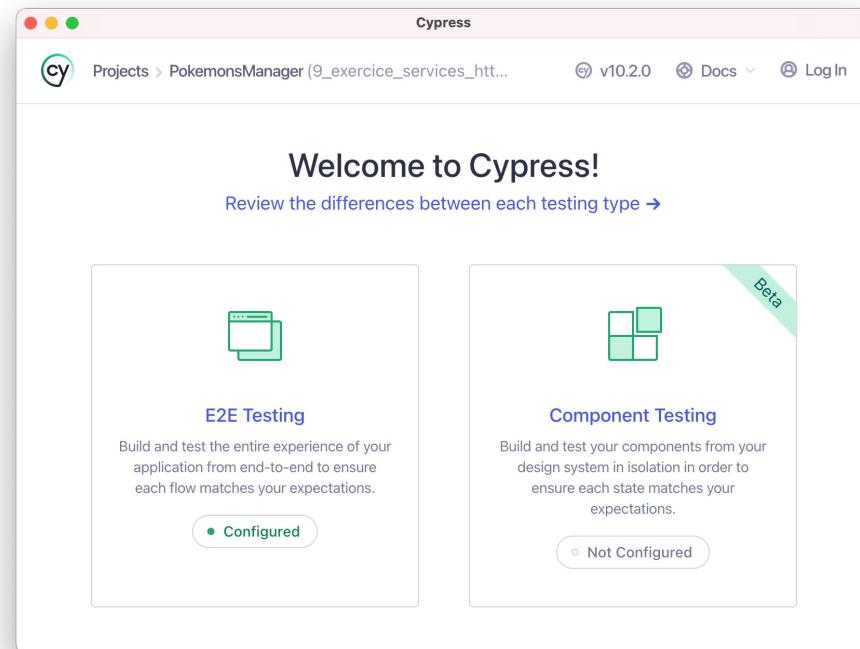
- Cypress est la solution recommandée et proposée à l'installation via **create-vue**
 - <https://fr.vuejs.org/guide/scaling-up/testing#unit-testing-components>
- Il existe d'autres alternatives (Playwright, Nightwatch, WebdriverIO)

```
Vue.js – The Progressive JavaScript Framework

✓ Nom du projet : ... tester ]  
✓ 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) ? > - Utilisez les flèches et appuyez sur la touche Entrée pour valider  
  Non  
> Cypress  
    Nightwatch  
    Playwright
```

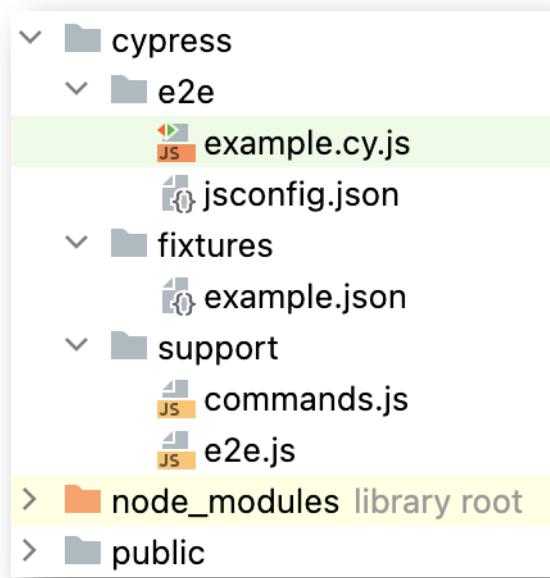
Cypress

- Cypress est un outil open source, d'automatisation de test pour le frontend
- Il permet d'écrire tout type de tests :
 - Tests de bout en bout
 - Test d'intégration
- Installation au démarrage du projet



Installation 1/2

- Une fois installé, Cypress est automatiquement intégré au projet



Installation 2/2

- Le fichier cypress.config.js définit la configuration de Cypress

```
import { defineConfig } from 'cypress'

export default defineConfig({
  e2e: {
    specPattern: 'cypress/e2e/**/*.{cy,spec}.{js,jsx,ts,tsx}',
    baseUrl: 'http://localhost:4173'
  }
})
```



Il est possible de définir un port fixe pour le lancement des tests

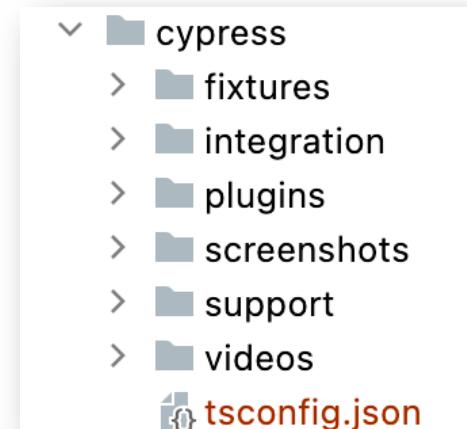
- Il est aléatoire par défaut

- <https://docs.cypress.io/guides/references/configuration#Configuration-File>

Les dossiers 1/2

Une fois intégré, Cypress est composé de plusieurs dossiers :

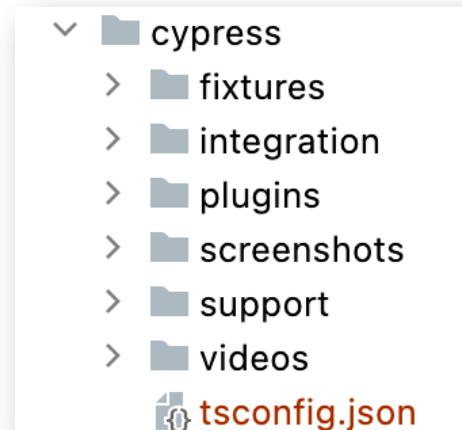
- **fixtures**
 - contient des collections de données pour mocker, accessible avec `cy.intercept`
- **integration**
 - contient les tests d'intégration
- **plugins**
 - Permet d'étendre des fonctionnalités avec des listeners/fonctions
 - <https://docs.cypress.io/api/plugins/configuration-api#Usage>
 - <https://docs.cypress.io/api/plugins/writing-a-plugin#Plugins-API>



Les dossiers 2/2

Une fois intégré, Cypress est composé de plusieurs dossiers :

- **screenshots**
 - contient les impressions d'écrans des tests exécutés (avec cypress run)
- **support**
 - Permet d'ajouter des fonctions personnalisées comme `cy.login()`
- **videos**
 - contient les vidéos des tests exécutés (avec cypress run)

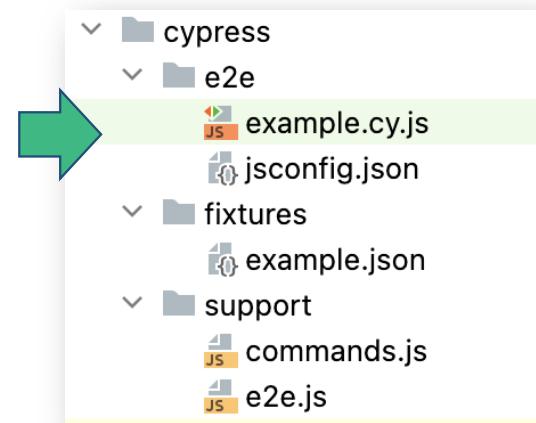


Focus sur les tests

- Un test par défaut est créé dans le dossier e2e:

```
// https://on.cypress.io/api

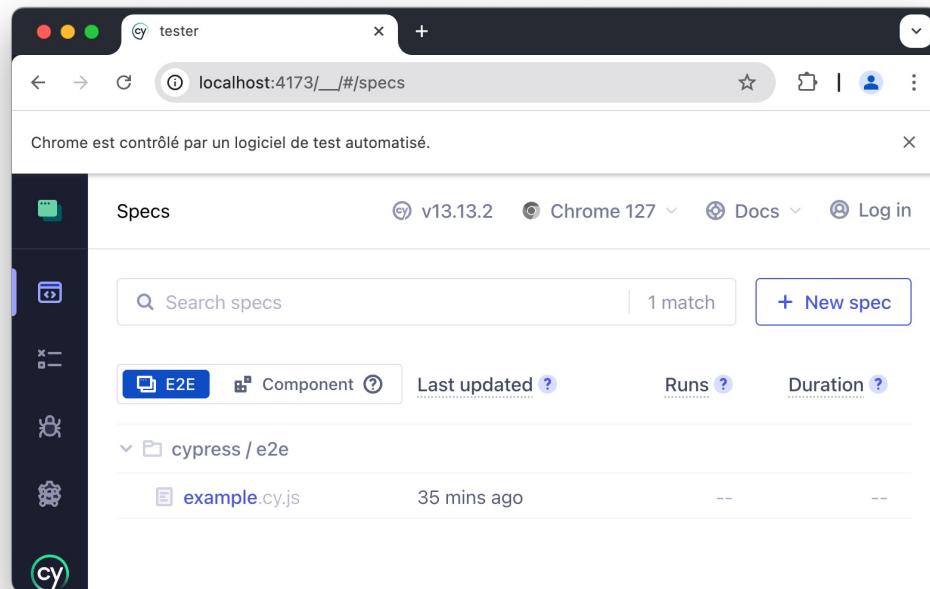
describe('My First Test', () => {
  it('visits the app root url', () => {
    cy.visit('/')
    cy.contains('h1', 'You did it!')
  })
})
```



- La variable **cy** est Cypress
- La méthode **visit** permet d'ouvrir la racine du site
- La méthode **contains** permet de vérifier que la page contient un élément
- Il est possible de créer plusieurs test dans un seul fichier, avec **it()**

Lancement graphique 1/2

- Les tests sont exécutables grâce à la commande **npm run test:e2e:dev**

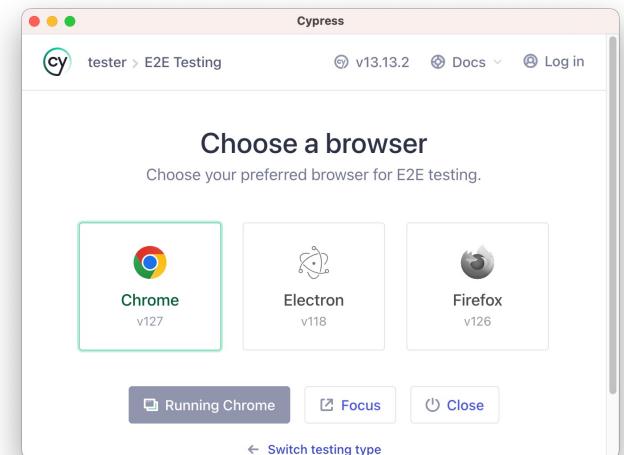


- Cypress s'ouvre et affiche la liste de vos tests

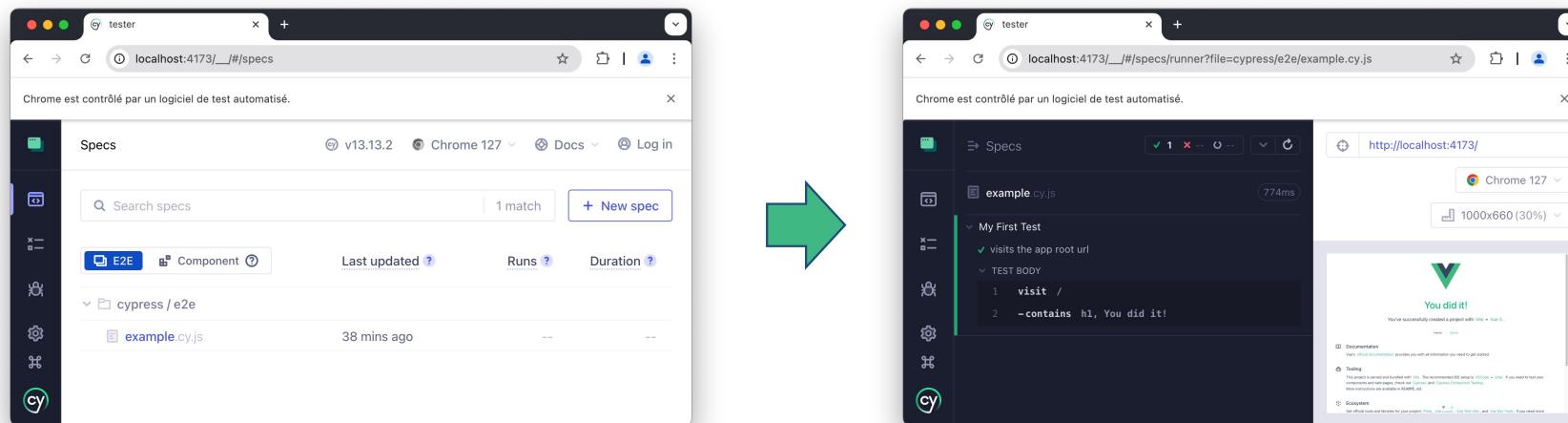


Lancement graphique 2/2

- Les tests sont exécutables sur différents navigateurs



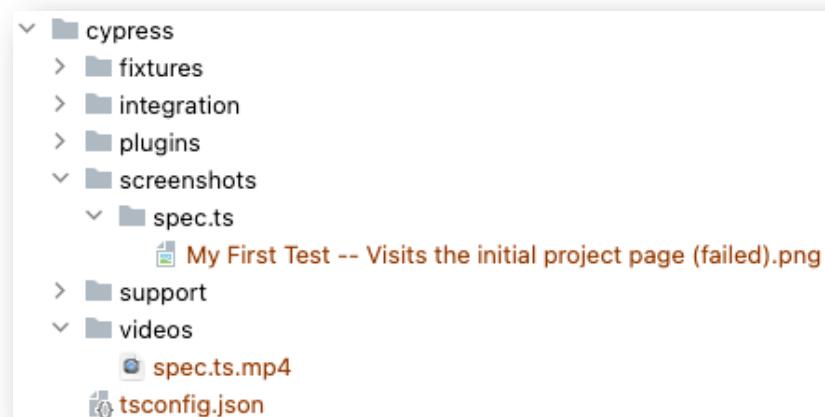
- Cypress lance ensuite les tests dans le navigateur choisi



Les enregistrements

- Il faut savoir que Cypress est capable d'avoir plusieurs sorties

- Console
- Photo
- Vidéo



VUE.JS

Les composables



Exemple d'un composable 1/4

- Exemple du suivi de la souris dans un composant
 - Basé sur la Composition API, implémentée directement dans un composant

Afin de réutiliser ce code,
nous allons extraire la logique
dans un fichier externe :

La fonction composable

```
<script setup>
import {onMounted, onUnmounted, ref} from 'vue'

const x = ref(0)
const y = ref(0)

function update(event) {
  x.value = event.pageX
  y.value = event.pageY
}

onMounted(() => window.addEventListener('mousemove', update))
onUnmounted(() => window.removeEventListener('mousemove', update))
</script>

<template>Mouse position is at: {{ x }}, {{ y }}</template>
```

Exemple d'un composable 2/4

- Voici la fonction composable :

```
// mouse.js (Fonction composable)
import { ref, onMounted, onUnmounted } from 'vue'
export function useMouse() {
  const x = ref(0)
  const y = ref(0)

  function update(event) {
    x.value = event.pageX
    y.value = event.pageY
  }

  onMounted(() => window.addEventListener('mousemove', update))
  onUnmounted(() => window.removeEventListener('mousemove', update))

  return { x, y }
}
```

```
// Utilisation dans un composant Example.vue
<script setup>
  import { useMouse } from './mouse.js'

  const { x, y } = useMouse()
</script>

<template>Mouse position is at: {{ x }}, {{ y }}</template>
```

Exemple d'un composable 3/4

- La fonctionnalité **useMouse()** peut être utilisée dans n'importe quel composant
- Les composables peuvent également être imbriqués
 - une fonction composable peut appeler une ou plusieurs autres fonctions compossables
 - Le nom de l'**API Composition** est issue de ce paradigme
- Il est possible d'extraire la logique d'ajout et de suppression d'un écouteur d'événement DOM dans son propre composable :
 - events.js
 - mouse.js

Exemple d'un composable 4/4

- Notre composable mouse.js est simplifiable grâce au composable event.js

```
// mouse.js
import { ref } from 'vue'
import { useEventListener } from './event'

export function useMouse() {
  const x = ref(0)
  const y = ref(0)

  useEventListener(window, 'mousemove', (event) => {
    x.value = event.pageX
    y.value = event.pageY
  })

  return { x, y }
}
```

```
// event.js
import { onMounted, onUnmounted } from 'vue'

export function useEventListener(target, event, callback) {
  // au lieu d'utiliser target, vous pouvez aussi
  // utiliser un sélecteur CSS pour trouver l'élément cible
  onMounted(() => target.addEventListener(event, callback))
  onUnmounted(() => target.removeEventListener(event, callback))
}
```

Exemple d'un composable asynchrone 1/2

- Il est possible de définir des paramètres à un composable
 - Le composable useMouse() ne prend aucun argument
- Le composant ci-dessous possède une logique transformable en composable

```
<script setup>
import { ref } from 'vue'
const data = ref(null)
const error = ref(null)

fetch('your-url')
  .then((res) => res.json())
  .then((json) => (data.value = json))
  .catch((err) => (error.value = err))
</script>
```

```
<template>
<div v-if="error">Oops! Error encountered: {{ error.message }}</div>
<div v-else-if="data">
  Data loaded:
  <pre>{{ data }}</pre>
</div>
<div v-else>Loading...</div>
</template>
```

Exemple d'un composable asynchrone 2/2

- Le composable `fetch.js` extrait la logique de récupération des données
 - Il est plus facile d'utiliser `useFetch()` dans nos composants

```
//fetch.js
import { ref } from 'vue'

export function useFetch(url) {
  const data = ref(null)
  const error = ref(null)

  fetch(url)
    .then((res) => res.json())
    .then((json) => (data.value = json))
    .catch((err) => (error.value = err))

  return { data, error }
}
```

```
<script setup>
import { useFetch } from './fetch.js'

const { data, error } = useFetch('...')

</script>
```

Aller plus loin avec les composables

- La documentation permet de lire des articles et de tester de nombreux exemples :
 - <https://fr.vuejs.org/guide/reusability/composables.html>
- Conventions de nommages et les bonnes pratiques
- Organisation du code avec les composables
- Comparaison systématiques avec la version de Vue 2

VUE.JS

FIN

