Titre du livre

1 Nom 1 2 Nom 2

26 décembre 2023

1 La librairie create-vue

1.1 Première utilisation de la librairie create-vue

Commencez par créer un dossier à l'emplacement que vous souhaitez. Ouvrez un terminal à l'emplacement du dossier et entrez la commande suivante :

```
npm init vue@latest
```

Cette commande permet d'installer et d'exécuter la dernière version de create-vue qui permet de lancer la configuration d'une nouvelle application Vue.js. Lorsque vous entrez la commande pour la première fois vous aurez la demande de confirmation d'installer create-vue :

```
Need to install the following packages:
  create-vue@latest
Ok to proceed?
```

Vous devrez simplement répondre `y` ou `yes`. Viennent ensuite toutes les questions sur la configuration de l'application. La première est le nom que vous souhaitez donner à votre application :

```
Project name: > vue-project
```

Par défaut, le nom est prérempli avec vue-project mais vous pouvez bien sûr le changer.
La deuxième question est sur l'utilisation de TypeScript :

```
Add TypeScript? ... No / Yes
```

- Comme nous l'avons vu, choisissez oui.
- Ensuite répondez non pour `JSX`. Nous n'utiliserons pas `JSX` qui est un langage de template React.
- Répondez non pour `Vue Router, Pinia, Vitest` et `Cypress` car nous les verrons plus tard dans la formation.
- Répondez oui à `ESLint`, qui permet de contrôler la qualité du code et répondez oui à Prettier pour le formatage du code.

Vous devez en être là :

```
17:32:32 ✔ erwan:~/code$ npm init vue@latest
Need to install the following packages:
  create-vue@latest
Ok to proceed? (y) yes

Vue.js - The Progressive JavaScript Framework

✔ Project name: … dymaproject
✔ 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 Cypress for both Unit and End-to-End testing? … No / Yes
✔ Add ESLint for code quality? … No / Yes
✔ Add Prettier for code formatting? … No / Yes

Scaffolding project in /home/erwan/code/dymaproject...

Done. Now run:
```

1.2 Installation des dépendances

Pour le moment, create-vue a déclaré toutes les dépendances et les configurations nécessaires en suivant vos options. Aucune dépendance `JavaScript` n'a encore été installée par `npm`. Pour les installer, il suffit d'ouvrir un terminal et d'aller dans le dossier de votre application, par exemple :

```
cd dymaproject
```

Et ensuite de lancer l'installation des dépendances :

```
npm install
```

1.3 Utilisation du linter

Un linter est un outil d'analyse de code qui permet de détecter les erreurs et les problèmes de syntaxe. La configuration du linter, en l'occurrence `ESLint` est dans le fichier `.eslintrc.cjs` :

```
/* eslint-env node */
require('@rushstack/eslint-patch/modern-module-resolution')

module.exports = {
  root: true,
  'extends': [
    'plugin:vue/vue3-essential',
    'eslint:recommended',
    '@vue/eslint-config-typescript',
    '@vue/eslint-config-prettier/skip-formatting'
  ],
  parserOptions: {
    ecmaVersion: 'latest'
  }
}
```

create-vue a donc automatiquement créé la bonne configuration du linter pour une utilisation avec `Vue.js`. Pour exécuter le linter avec la configuration faites :

```
npm run lint
```

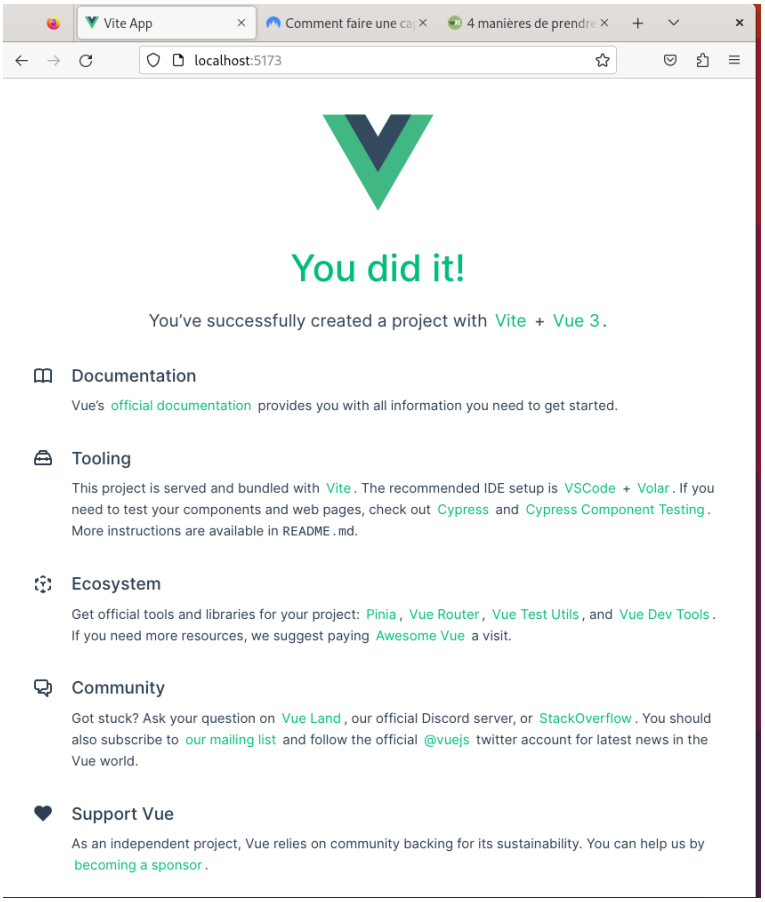
Cela exécutera le script `lint` déclaré dans le fichier `package.json`. Pour l'instant il y a bien sûr aucune indication car nous n'avons pas commencé à coder ! Mais exécuter cette commande de temps en temps lors du développement permet d'éviter certaines erreurs et de suivre les recommandations pour les bonnes pratiques en matière de syntaxe (appelées coding style).

1.4 Lancer le serveur de développement

Pour lancer le serveur de développement il suffit d'exécuter le script dev :

```
npm run dev
```

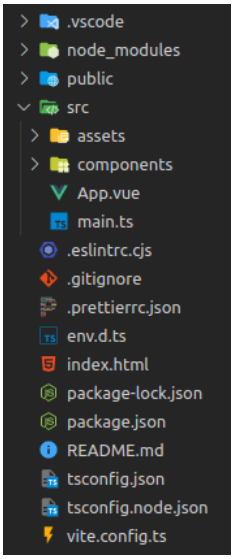
Cela va en fait exécuter vite qui va lancer son serveur de développement. Vous pourrez ainsi accéder à l'application `Vue.js` dans votre navigateur à l'adresse http ://localhost :5173/.



2 Architecture initiale

2.1 Architecture initiale d'un projet généré par create-vue

L'architecture initiale du projet généré est la suivante :



2.2 Détail des fichiers et des dossiers

- `vite.config.js` : fichier de configuration de Vite.
- `tsconfig.node.json` : fichier de configuration de Vite pour pouvoir utiliser TypeScript. Vous pouvez retirer vitest, cypress, playwright de la propriété types car nous ne les utiliserons pas pour le moment :

```
{
  "extends": "@vue/tsconfig/tsconfig.node.json",
  "include": ["vite.config.*"],
  "compilerOptions": {
    "composite": true,
    "types": ["node"]
  }
}
```

- `tsconfig.json` : fichier de configuration de TypeScript. Toutes les applications utilisant TypeScript ont ce fichier. Il permet de configurer notamment les options pour transpiler le TypeScript en JavaScript.
- `README.md` : décrit brièvement les différentes commandes possibles et la configuration recommandée de l'éditeur VS Code.
- `package.json` et `package-lock.json` : ces fichiers `JSON` permettent de décrire le projet et surtout de détailler les dépendances et leurs versions requises pour l'utiliser. Il décrit également les scripts pouvant être lancés avec npm run.

Les scripts disponibles sont :

- dev : qui permet de lancer le serveur de développement de Vite en local.
- build : qui permet de construire la version de l'application optimisée pour la production.
- preview : qui permet de visualiser la version de production en local (par exemple avant de la mettre réellement en production sur des serveurs).
- typecheck : qui permet de vérifier les types du code TypeScript sans transpiler le TypeScript en `JavaScript`.
- lint : qui permet d'exécuter ESLint avec les bonnes options.

- index.html : template `HTML` qui sera envoyé au navigateur.
- env.d.ts : fichier qui permet de charger des types supplémentaires pour utiliser TypeScript avec Vue.js.
- .gitignore : permet de lister les fichiers qui ne doivent pas être pris en compte par Git.
- .eslintrc : fichier de configuration de ESLint que nous avons déjà expliqué.
- src : dossier qui contient les fichiers sources de votre application (d'où le nom src pour source).
- public : dossier qui contient les fichiers qui n'ont pas besoin d'être traité par Vite ou par aucun outil et qui sont directement envoyés par le serveur au navigateur.
- src/main.ts : il s'agit du point d'entrée de notre application. Nous expliquerons plus loin dans le cours la fonction `createApp()` ainsi que `$mount`. Sachez simplement que nous importons notre composant racine App et créons l'instance racine de l'application ici.
- src/App.vue : il s'agit du composant référencé dans main.ts qui constitue la racine des vues de notre application. Il s'agit du premier composant rendu, et de lui découle les rendus de tous les autres composants. Nous étudierons la structure des composants dans une prochaine leçon.
- src/components : dossier qui contient les composants de notre application Vue.
- src/components/HelloWorld.vue : sous-composant de App.vue.
- src/assets : dossier contenant les ressources de l'application comme les images, le `CSS` etc.
- node_modules : contient toutes les dépendances installées par npm.

3 Comment fonctionne Vue ?

Nous allons voir les fichiers par ordre logique. Le navigateur reçoit en premier l'index.html.

3.1 Le fichier index.html

Le fichier index.html est ce que le navigateur va lire en premier :

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" href="/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vite App</title>
  </head>
  <body>
    <div id="app"></div>
    <script type="module" src="/src/main.ts"></script>
  </body>
</html>
```

Notez bien qu'il y a une `div` conteneur avec l'id app. Notez également que le fichier src/main.ts est chargé comme un module `JavaScript`. Le navigateur demande donc ce module au serveur.

3.2 Le fichier src/main.ts

Le fichier main.ts est le point d'entrée de l'application car c'est le premier fichier `JavaScript` chargé par le navigateur.

Note : le fichier est en .ts car c'est un fichier `TypeScript` mais il sera transpilé par Vite en `JavaScript` que ce soit pour le développement ou la production. En effet, le navigateur ne comprend que le langage `JavaScript` et pas le `TypeScript`.

```
import { createApp } from "vue";
import App from "./App.vue";

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

`import App from "./App.vue";` permet d'importer le composant App que nous verrons juste après. `createApp()` permet de créer une instance de votre application Vue.js. Elle prend en argument, le composant racine, c'est-à-dire le premier composant de votre application qui aura d'autres composants enfants.

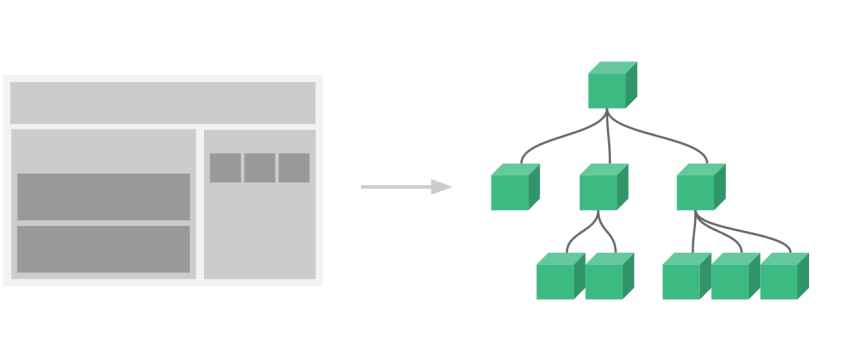
`mount()` permet de monter l'application instanciée par `createApp()` dans un conteneur qui est identifié par un sélecteur `CSS`. Ici, cela signifie que le composant racine App sera monté sur le conteneur dont l'id est app :

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

Le composant sera ainsi affiché (on dit plus souvent rendu) dans ce conteneur.

3.3 Le fichier src/App.vue

Le fichier `App.vue` est le composant racine. C'est le premier composant de l'application qui est chargé et instancié par `createApp()`. Ce composant est le sommet de ce qu'on appelle l'arbre des composants : à savoir l'organisation hiérarchique des composants de l'application. Une application a des dizaines voir des centaines de composants qui sont organisés en arbre à partir du composant racine :



Remarquez déjà que le composant App.vue est constitué de trois parties : `template, script` et `style`. Nous étudierons bientôt les composants en détail.

3.4 Aller plus loin sur le fonctionnement de Vue.js : le DOM virtuel

Nous allons d'abord effectuer quelques rappels avant d'approfondir le fonctionnement de Vue.js.

3.4.1 Le DOM (Document Object Model)

Le DOM permet de passer du HTML à un grand objet document qui est un arbre regroupant tous les éléments déclarés en HTML. Autrement dit, lorsque le navigateur reçoit le HTML de la page, il va le parser (c'est-à-dire l'analyser) et le transformer en DOM grâce à des algorithmes. Ainsi, les attributs HTML deviennent automatiquement des propriétés des objets du DOM. Par exemple :

```
<body id="page">
```

Devient un nœud sur l'objet document qui sera un objet body contenant une propriété id contenant la valeur page.

Il existe plusieurs types de noeuds :

Constante	Valeur	Description
`Node.ELEMENT_NODE`	`1`	Un noeud `Element` tel que `<p>` ou `<div>`.
`Node.TEXT_NODE`	`3`	Le `Text` actuel de l'`Element` ou `Attr`.
`Node.PROCESSING_INSTRUCTION_NODE`	`7`	Une `ProcessingInstruction` d'un document XML tel que la déclaration `<?xml-stylesheet ... ?>`.
`Node.COMMENT_NODE`	`8`	Un noeud `Comment`.
`Node.DOCUMENT_NODE`	`9`	Un noeud `Document`.
`Node.DOCUMENT_TYPE_NODE`	`10`	Un noeud `DocumentType` c'est-à-dire `<!DOCTYPE html>` pour des documents HTML5.
`Node.DOCUMENT_FRAGMENT_NODE`	`11`	Un noeud `DocumentFragment`.

3.4.2 Le DOM virtuel

Dans les applications Web modernes, il peut y avoir des centaines ou des milliers de nœuds sur lesquels sont enregistrés de nombreux gestionnaires d'événements. En conséquence, de très nombreuses mises à jour du DOM doivent être réalisées. Or, ces mises à jour sont très coûteuses en performance car plus le DOM est grand, plus les recherches et les changements sont coûteux. Vue.js utilise donc un DOM virtuel (appelé VDOM) qui est une représentation du DOM en JavaScript. Le DOM virtuel n'est donc qu'un simple, immense, objet JavaScript :

```
const vnode = {
  type: 'div',
  props: {
    id: 'hello'
  },
  children: [
    /* plein de vnodes */
  ]
}
```

Les noeuds du DOM virtuels sont appelés des noeuds virtuels ou vnode (pour virtual node). Vue.js va transformer le VDOM en DOM pour que le navigateur puisse afficher la page. Ce processus est appelé le montage (mount en anglais).

Lorsque des mises à jour sur la page doivent intervenir, le DOM virtuel est modifié avant que les modifications sur le DOM ne soient effectuées. Le DOM virtuel est une représentation légère du

DOM en JavaScript qui peut être utilisée par des algorithmes très performants de comparaison des différences entre DOM virtuel et DOM.

Dès lors que le DOM doit être modifié, un patch est appliqué par Vue.js de manière extrêmement optimisée pour ne réaliser que les changements absolument nécessaires. Par exemple, si aucune valeur affichée ne change, alors aucune mise à jour du DOM n'est déclenchée ce qui économise beaucoup en performance.

En résumé, Vue.js va créer au départ un DOM virtuel puis le convertir en DOM HTML pour l'affichage lors du montage. Lorsque des changements interviennent, par exemple un événement survient, les changements affectant le DOM virtuel sont analysés par des algorithmes et seuls les changements ayant un effet sur l'affichage sont propagés au DOM lors de la phase de mise à jour (patch ou diffing en anglais). Ainsi, seules les insertions, les modifications et les suppressions d'éléments HTML absolument nécessaires sont effectuées, ce qui permet un énorme gain de performance.

4 Les API composition et options

Vue.js propose deux API pour écrire des composants Vue : l'API options et l'API composition.

- L'API options est l'API originelle qui existe depuis la première version du framework.
- L'API composition est l'API avec laquelle Vue a été réécrite pour la version 3. C'est la nouvelle API qui est aujourd'hui recommandée et celle que nous verrons dans la formation.

4.1 L'API options

Dans l'ancienne API options, la logique du composant est définie dans un objet comportant des options comme data, methods ou mounted par exemple. Les propriétés sont définies sur l'objet this disponible dans les options.

Nous ne détaillerons pas plus car ce n'est pas l'API que nous utiliserons. Nous la présentons uniquement pour que vous puissiez la reconnaître lorsque vous la rencontrez. Voici à quoi ressemble un composant écrit avec l'API options :

```
<script>
export default {
  data() {
    return {
      count: 0
    }
  },
  methods: {
    increment() {
      this.count++
    }
  },
  mounted() {
    console.log(`La valeur initiale est ${this.count}.`)
  }
}
</script>

<template>
```

```
  <button @click="increment">Valeur du compteur : {{ count }}</button>
</template>
```

4.2 L'API composition

Avec l'API composition nous définissons la logique d'un composant en important des fonctions. Voilà le même exemple avec la nouvelle API :

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

const count = ref(0)
function increment() {
  count.value++
}

onMounted(() => {
  console.log(`La valeur initiale est ${count.value}.`)
})
</script>

<template>
  <button @click="increment">Valeur du compteur : {{ count }}</button>
</template>
```

4.3 Les Single-File Components (SFC)

Les SFC sont des composants écrits dans des fichiers qui terminent par l'extension .vue. Un SFC comporte la logique du composant (partie script en `JavaScript/TypeScript`), le template (en `HTML`) et le style (en `CSS` ou `Sass`).

Il est fortement recommandé d'utiliser les SFC pour écrire des composants Vue.js. C'est d'ailleurs ce que nous utiliserons tout le long de la formation.

5 Création d'un composant

5.1 Passer en mode Take Over

L'extension Volar est plus performante pour gérer l'utilisation de `TypeScript` avec Vue.js. Aussi, il faut configurer l'éditeur `VS Code` pour lui dire que nous utilisons Volar.

Pour ce faire, aller sur l'onglet `Extensions de VS Code`. Recherchez `@builtin typescript`. Cliquez sur `TypeScript and JavaScript Language Features`. Cliquez sur `Désactiver (espace de travail)`. Cela désactivera uniquement pour ce projet. Cliquez ensuite sur Rechargement requis.

5.2 Utilisation du mode strict de TypeScript

Pour activer l'option strict du compilateur `TypeScript` modifiez le fichier `tsconfig.json` :

```
{
  "extends": "@vue/tsconfig/tsconfig.web.json",
  "include": ["env.d.ts", "src/**/*", "src/**/*.vue"],
  "compilerOptions": {
    "strict": true,
    "baseUrl": ".",
    "paths": {
      "@/*": ["./src/*"]
    }
  },
  "references": [
    {
      "path": "./tsconfig.node.json"
    }
  ]
}
```

Cette option permet d'activer de nombreuses configurations plus strictes pour le contrôle des types par le compilateur. Cela permet d'améliorer sa capacité à détecter des erreurs.

5.3 Création du premier composant

Commençons par supprimer les composants mis en place automatiquement par create-vue pour la page de présentation lors du lancement de l'application.

1. Supprimez tous les fichiers et les dossiers dans `src/components/`.
2. Supprimez le contenu, mais pas le fichier de `src/App.vue`.

Comme nous l'avons vu, un composant monofichier (SFC, pour Single-file component) est un fichier .vue dont le nom est par convention en `PascalCase`. Un composant monofichier a donc un template qui comporte le `HTML`, une partie script qui comporte le JavaScript et une partie style qui comporte le `CSS`. Voici l'exemple minimal de la vidéo :

```
<template>
    <h1>Bonjour {{name}}</h1>
</template>

<script setup lang="ts">
    const name = 'Jean';
</script>

<style></style>
```

6 Vue.js sans Vite

6.1 Utiliser Vue.js sans Vite

Créez un dossier projet-vue.
Dans ce dossier, créez un fichier index.html :

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <script src="https://unpkg.com/vue@3"></script>
  </head>
  <body>
    <div id="app">
      <h1>Bonjour {{ name }}</h1>
    </div>

    <script>
      Vue.createApp({
        setup() {
          const name = 'test';
          return {
            name,
          };
        },
      }).mount('#app');
    </script>
  </body>
</html>
```

Ici, nous n'utilisons que la librairie Vue.js sans Vite et sans aucune autre librairie.

- Il n'y a aucune étape de `build`, nous ne pouvons pas utiliser Sass, TypeScript etc.
- Il n'y a pas de minification du code et pleins d'avantages apportés par Vite.

Cet exemple permet simplement de montrer comment utiliser Vue.js seul mais dans toute la formation nous n'utiliserons pas du tout cette méthode car il est impossible d'écrire une application complexe comme cela. Cette manière de faire peut être utile pour par exemple gérer une barre de recherche sur une application avec Symfony côté serveur.