Vue.js : utilisation de **TypeScript**

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

Vue.js

Remarque

- Pour créer un projet avec **Vite**, il existe trois commandes différentes.
- Nous utiliserons la dernière commande qui génère moins de composants

Vue.js

Première commande, exécuter la commande

```
npm init vue@latest
```

Vue.js

Première commande, exécuter la commande

```
npm init vue@latest
```

Deuxième commande, exécuter la commande

```
npm create vue@latest
```

Vue.js

Et ensuite, répondre aux différentes questions ainsi

```
Vue.js - The Progressive JavaScript Framework

√ Project name: ... cours-vue-ts
√ 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? ... No / Yes / Cypress / Playwright
√ Add ESLint for code quality? ... No / Yes
√ Add Prettier for code formatting? ... No / Yes

Scaffolding project in <chemin-projet>

Done. Now run:

 cd cours-vue-ts
 npm install
 npm run dev
```

Troisième commande, exécuter la commande

```
npm init vite@latest
```

Troisième commande, exécuter la commande

```
npm init vite@latest
```

Ensuite

```
✓ Project name: ... cours-vue-ts
? Select a framework: » - Use arrow-keys. Return to submit.
    Vanilla
>   Vue
    React
    Preact
    Lit
    Svelte
    Solid
    Qwik
    Others
```

Troisième commande, exécuter la commande

```
npm init vite@latest
```

Ensuite

```
✓ Project name: ... cours-vue-ts
? Select a framework: » - Use arrow-keys. Return to submit.
    Vanilla
>   Vue
    React
    Preact
    Lit
    Svelte
    Solid
    Qwik
    Others
```

Et enfin

```
✓ Project name: ... cours-vue-ts
✓ Select a framework: » Vue
? Select a variant: » - Use arrow-keys. Return to submit.
>   TypeScript
    JavaScript
    Customize with create-vue 
    Nuxt 
```

Vue.js

Ouvrez le dossier `cours-vue-ts` avec VSC et installez les dépendances avec

```
npm install
```

Vue.js

Ouvrez le dossier `cours-vue-ts` avec VSC et installez les dépendances avec

```
npm install
```

Pour lancer l'application, exécutez

```
npm run dev
```

Vue.js

Arborescence d'un projet **Vue.js**

- `node_modules` : contenant les fichiers nécessaires de la librairie **Node.js** pour un projet **Vue.js**
- `src` : contenant les fichiers sources de l'application
- `package.json` : contenant l'ensemble de dépendance de l'application
- `public` : contenant les fichiers statiques à intégrer systématiquement dans `dist`
- `vite.config.js` : fichier de configuration de **Vite**
- `index.html` : point d'entrée de l'application

Vue.js

Si vous aviez choisi **TypeScript** comme langage, vous auriez

- `tsconfig.json` : fichier de configuration principal pour **TypeScript** pouvant contenir les paramètres de compilation, les chemins d'inclusion et d'exclusion, les options de module...
- `package.node.json` : fichier de configuration permettant d'activer des fonctionnalités **TypeScript** spécifiques à **Node.js**, comme les modules CommonJS, les cibles ECMAScript...

Vue.js

Que contient `src` ?

- `assets` : contenant les fichiers statiques à intégrer dans `dist` s'ils sont explicitement référencés dans le projet
- `components` : contient un composant `HelloWorld.vue`
- `main.ts`) : fichier référencé par `index.html` et permettant de charger l'application **Vue.js** dans l'élément ayant l'identifiant `app`
- `App.vue` : fichier référencé par `main.ts` et utilisant le composant `HelloWorld.vue`

Vue.js

Contenu d'`index.html`

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale
      =1.0" />
    <title>Vite + Vue + TS</title>
  </head>
  <body>
    <div id="app"></div>
    <script type="module" src="/src/main.ts"></script>
  </body>
</html>
```

Vue.js

Contenu de `src/main.ts`

```
import { createApp } from 'vue'
import './style.css'
import App from './App.vue'

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

Vue.js

En plus de ces extensions

- **Volar**
- **Vue 3 Snippets**

Vue.js

En plus de ces extensions

- **Volar**
- **Vue 3 Snippets**

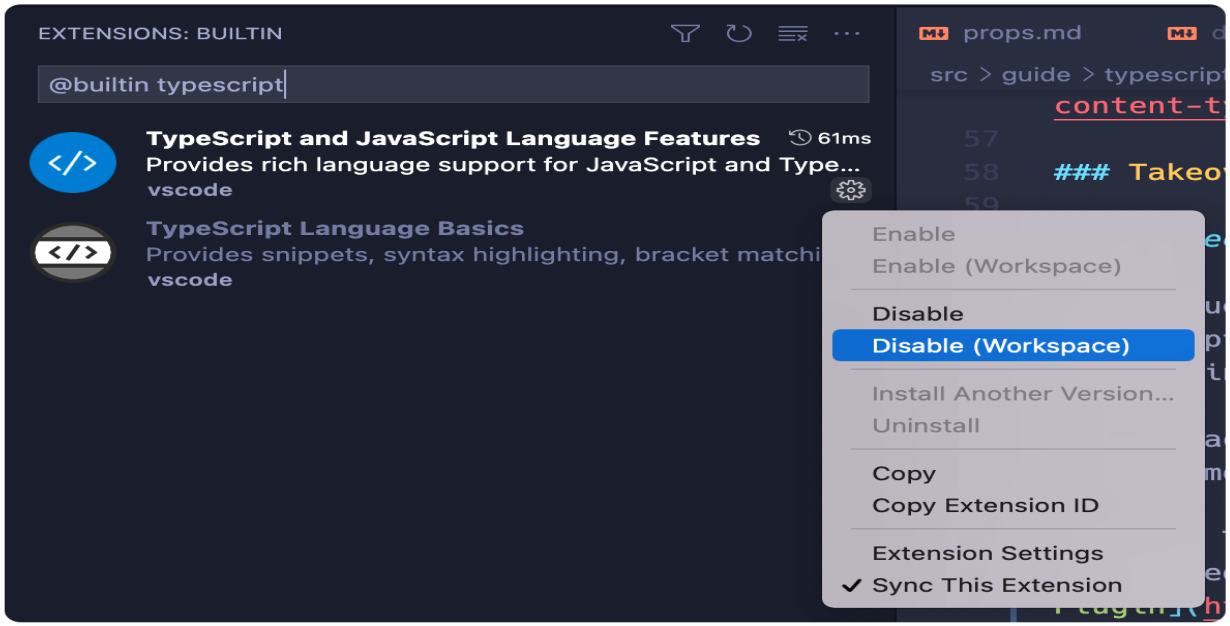
Installer l'extension suivante pour la prise en charge de **TypeScript** dans un projet **Vue.js**

TypeScript Vue Plugin

Vue.js

Pour résoudre quelques interférences entre les extensions et laisser Volar prendre en charge les fichiers Vue et TS (Extrait de la doc. officielle)

1. Dans l'espace de travail de votre projet, affichez la palette de commandes avec `Ctrl + Shift + P` (macOS : `Cmd + Shift + P`).
2. Tapez `built` et sélectionnez "Extensions : Afficher les extensions intégrées".
3. Tapez `typescript` dans la zone de recherche d'extension (ne supprimez pas le préfixe `@builtin`).
4. Cliquez sur la petite icône d'engrenage de "Fonctionnalités du langage TypeScript et JavaScript", et sélectionnez "Désactiver (espace de travail)".
5. Rechargez l'espace de travail. Le mode Prise de contrôle sera activé lorsque vous ouvrirez un fichier Vue ou TS.



Vue.js

Remarque

- Pour créer un projet avec **Vite**, il existe trois commandes différentes.
- Nous utiliserons la dernière commande qui génère moins de composants

Vue.js

Pour la suite, considérons le code simplifié suivant de `HelloWorld` (avec API Composition)

```
<script setup lang="ts">
import { ref } from 'vue'

defineProps<{ msg: string }>()

const count = ref(0)
</script>

<template>
  <h1>{{ msg }}</h1>

  <div class="card">
    <button type="button" @click="count++">count is {{ count }}</button>
  </div>
</template>
```

Vue.js

Créons également `HelloWorldOption` : la version (avec API Option) de `HelloWorld`

```
<script lang="ts">
export default {
    data() {
        return {
            count: 0
        }
    },
    props: {
        msg: String
    }
}
</script>

<template>
    <h1>{{ msg }}</h1>

    <div class="card">
        <button type="button" @click="count++"> count is {{ count }} </button>
    </div>
</template>
```

Vue.js

N'oublions pas de déclarer `HelloWorldOption` comme enfant de `App`

```
<script setup lang="ts">

import HelloWorld from './components/HelloWorld.vue'
import HelloWorldOption from './components/HelloWorldOption.vue'

</script>

<template>
  <div>
    <a href="https://vitejs.dev" target="_blank">
      <img src="/vite.svg" class="logo" alt="Vite logo" />
    </a>
    <a href="https://vuejs.org/" target="_blank">
      <img src="./assets/vue.svg" class="logo vue" alt="Vue logo" />
    </a>
  </div>
  <HelloWorld msg="Vite + Vue" />
  <HelloWorldOption msg="Vite + Vue" />
</template>
```

Vue.js

N'oublions pas de déclarer `HelloWorldOption` comme enfant de `App`

```
<script setup lang="ts">

import HelloWorld from './components/HelloWorld.vue'
import HelloWorldOption from './components/HelloWorldOption.vue'

</script>

<template>
  <div>
    <a href="https://vitejs.dev" target="_blank">
      <img src="/vite.svg" class="logo" alt="Vite logo" />
    </a>
    <a href="https://vuejs.org/" target="_blank">
      <img src="./assets/vue.svg" class="logo vue" alt="Vue logo" />
    </a>
  </div>
  <HelloWorld msg="Vite + Vue" />
  <HelloWorldOption msg="Vite + Vue" />
</template>
```

Relancez le projet et vérifiez que les deux compteurs s'affichent.

Vue.js

Pour une meilleure inférence de type et détection des erreurs, Vue.js recommande de définir les composants API Options avec `defineComponent`

```
<script lang="ts">
import { defineComponent } from 'vue'

export default defineComponent({
    data() {
        return {
            count: 0
        }
    },
    props: {
        msg: String
    }
})
</script>
```

Vue.js

Créons une interface `Produit` dans `src/models/produit.model.ts`

```
export interface Produit {
    nom: string;
    prix: number;
    quantite: number;
}
```

Vue.js

Créons une interface `Produit` dans `src/models/produit.model.ts`

```
export interface Produit {
    nom: string;
    prix: number;
    quantite: number;
}
```


Exercice

Refaire l'exercice **primeur-produit** (rappel de l'énoncé dans la slide suivante) en utilisant l'interface `Produit` et `defineComponent`.

Vue.js

Exercice : **primeur-produit**

- Créez deux composants `PrimeurComponent` et `ProduitComponent` : `PrimeurComponent` est le composant parent des composants `ProduitComponent`
- Le composant `PrimeurComponent` a un attribut `produits` : à déclarer dans la fonction `data`.
- Utilisez `v-for` pour créer autant de composants `ProduitComponent` que d'éléments dans le tableau `produits` : chaque composant `produit` reçoit un le nom, le prix et la quantité qu'il doit afficher.

Vue.js

contenu de `PrimeurComponent`

```
<template>
    <ul v-for="elt in produits">
        <ProduitComponent  :produit="elt" />
    </ul >
</template>

<script  lang="ts">
import { defineComponent } from 'vue'
import { Produit } from '../models/produit.model.ts'
import ProduitComponent from './Produit.vue'
export default defineComponent({
    components: {
        ProduitComponent
    },
    data() {
        let produits: Produit[] = [
            { nom: "banane", prix: 3, quantite: 10 },
            { nom: "fraise", prix: 10, quantite: 20 },
            { nom: "poivron", prix: 5, quantite: 10 }
        ]
        return {
            produits
        }
    },
})
</script>
```

Vue.js

contenu de `ProduitComponent`

```
<template>
    <div>
        {{ produit?.nom }} {{  produit?.prix }} {{ produit?.quantite }}
    </div>
</template>

<script  lang="ts">
import { defineComponent } from 'vue'
import type { PropType } from 'vue'
import { Produit } from '../models/produit.model.ts'

export default defineComponent ({
    props: {
        produit: Object as PropType<Produit>
    }
})
</script>
```

Vue.js

Exercice : **primeur-produit**

- Dans `PrimeurComponent`, déclarez un attribut `total` dans `data`.
- Dans `ProduitComponent`, ajoutez une zone de saisie et un bouton.
- En choisissant une quantité et appuyant sur le bouton, le total sera mis à jour et le bouton sera désactivé.

Contenu de `PrimeurComponent`

```
<template>
    <h1>Total {{  total }}</h1>
    <ul v-for="(elt, ind) in produits">
        <ProduitComponent  :produit="elt" @send-data="calculer($event, ind)" />
    </ul >
</template>
<script  lang="ts">
import { defineComponent } from 'vue'
import { Produit } from '../models/produit.model.ts'
import ProduitComponent from './Produit.vue'
export default defineComponent({
    components: {
        ProduitComponent
    },
    data() {
        let produits: Produit[] = [
            { nom: "banane", prix: 3, quantite: 10 },
            { nom: "fraise", prix: 10, quantite: 20 },
            { nom: "poivron", prix: 5, quantite: 10 }
        ]
        return {
            total: 0,
            produits
        }
    },
    methods: {
        calculer(qte: number, ind: number) {
            this.total += qte* this.produits[ind].prix
        }
    }
})
</script>
```

Contenu de `ProduitComponent`

```
<template>
    <div>
        {{ produit?.nom }} {{ produit?.prix }} {{ produit?.quantite }}
        <input type="number" v-model="qteCommandee">
        <button @click="envoyer()">
            Envoyer
        </button>
    </div>
</template>

<script  lang="ts">
import { defineComponent } from 'vue'
import type { PropType } from 'vue'
import { Produit } from '../models/produit.model.ts'

export default defineComponent({
    data() {
        return {
            qteCommandee: 0
        }
    },
    props: {
        produit: Object as PropType<Produit>
    },
    methods: {
        envoyer() {
            this.$emit('sendData', this.qteCommandee)
        }
    }
})
</script>
```