Vue.js 3 - Cours Avance

v-model, Slots, HTTP, Router

Ibrahim ALAME

Formation Vue.js

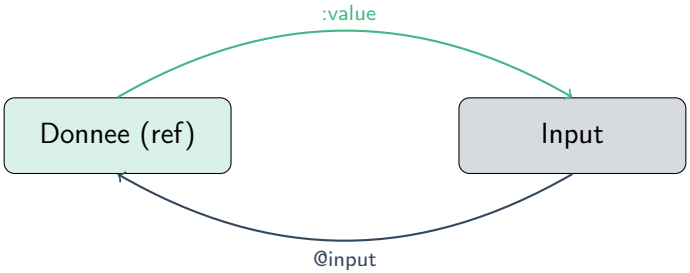
2025

Sommaire

v-model - Definition

Qu'est-ce que v-model ?

v-model cree une **liaison bidirectionnelle** entre un element de formulaire et une donnee reactive.



`v-model` = `:value` + `@input` combines !

v-model - Elements natifs

Input texte

```
<input v-model="nom" type="text">
<input v-model="email" type="email">
```

Textarea

```
<textarea v-model="message"></textarea>
```

Checkbox (boolean)

```
<input v-model="accepte" type="checkbox">
```

Radio

```
<input v-model="genre" type="radio"
       value="homme">
<input v-model="genre" type="radio"
       value="femme">
```

Select

```
<select v-model="pays">
  <option value="fr">France</option>
  <option value="be">Belgique</option>
</select>
```

v-model - Modificateurs

Les 3 modificateurs

Modificateur	Effet	Exemple
`.number`	Convertit en nombre	`v-model.number="age"`
`.trim`	Supprime les espaces	`v-model.trim="nom"`
`.lazy`	Sync sur blur	`v-model.lazy="email"`

```
<!-- Age sera un number, pas un string -->
<input v-model.number="age" type="number">

<!-- Espaces supprimes automatiquement -->
<input v-model.trim="nom" type="text">

<!-- Sync uniquement quand on quitte le champ -->
<input v-model.lazy="recherche" type="text">
```

v-model - Sur composants (simple)

Composant enfant

```
<script setup lang="ts">
defineProps<{
  modelValue: string
}>()

const emit = defineEmits<{
  (e: 'update:modelValue',
   v: string): void
}>()
</script>

<template>
  <input
    :value="modelValue"
    @input="emit('update:modelValue',
      $event.target.value)">
</template>
```

Parent

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

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

<template>
  <MonInput v-model="texte" />
  <p>{{ texte }}</p>
</template>
```

Convention

Prop : `modelValue`

Event : `update:modelValue`

v-model - Multiple (nomme)

```vue
<!-- Parent -->
<Formulaire
  v-model:nom="user.nom"
  v-model:email="user.email"
  v-model:age="user.age"
/>
```

```vue
<!-- Enfant -->
<script setup lang="ts">
defineProps<{ nom: string; email: string; age: number }>()

const emit = defineEmits<{
  (e: 'update:nom', v: string): void
  (e: 'update:email', v: string): void
  (e: 'update:age', v: number): void
}>()
</script>
```

Regle

`v-model:xxx` → prop `xxx` + emit `update:xxx`

v-model - defineModel() (Vue 3.4+)

Syntaxe simplifiee

Avant (verbose)

```
<script setup>
const props = defineProps<{
  modelValue: string
}>()
const emit = defineEmits<{
  (e: 'update:modelValue',
   v: string): void
}>()
</script>

<template>
  <input
    :value="props.modelValue"
    @input="emit(...)">
</template>
```

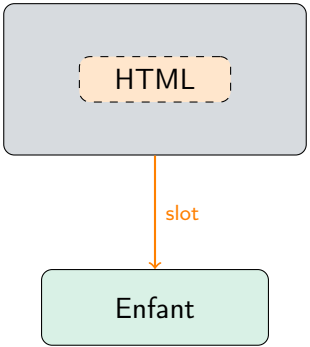
Apres (simple)

```
<script setup>
const model = defineModel<string>()

// Pour v-model nomme:
const nom = defineModel<string>('nom')
</script>

<template>
  <input v-model="model">
</template>
```

Slots - Definition

Qu'est-ce qu'un slot ?

Les **slots** permettent de passer du **contenu HTML** d'un composant parent vers un composant enfant. C'est un mecanisme de **projection de contenu**.



Slots - Slot par defaut

Composant Carte.vue

```
<template>
  <div class="carte">
    <slot>
      Contenu par defaut
    </slot>
  </div>
</template>
```

Utilisation

```
<!-- Avec contenu -->
<Carte>
  <h2>Titre</h2>
  <p>Description...</p>
</Carte>

<!-- Sans contenu (defaut) -->
<Carte />
```

Attention

`<Carte></Carte>` avec espaces/retours n'affiche PAS le defaut !
Utilisez `<Carte />` pour le contenu par defaut.

Slots - Slots nommes

Composant Layout.vue

```
<template>
  <header>
    <slot name="header">
      Header defaut
    </slot>
  </header>

  <main>
    <slot></slot>
  </main>

  <footer>
    <slot name="footer">
      Footer defaut
    </slot>
  </footer>
</template>
```

Utilisation

```
<Layout>
  <template #header>
    <h1>Mon Site</h1>
  </template>

  <p>Contenu principal</p>

  <template #footer>
    <p>Copyright 2025</p>
  </template>
</Layout>
```

Syntaxe

`#header` = `v-slot:header`

Slots - Scoped slots

Passer des donnees de l'enfant vers le parent

Enfant (Liste.vue)

```
<template>
  <ul>
    <li v-for="(item, i) in items"
        :key="i">
      <slot :item="item" :index="i">
        {{ item }}
      </slot>
    </li>
  </ul>
</template>
```

Parent

```
<Liste :items="['A','B','C']">
  <template #default="{ item, index }">
    {{ index + 1 }}. {{ item }}
  </template>
</Liste>
```

Resultat :

- 1. A
- 2. B
- 3. C

HTTP - Les methodes REST

CRUD avec HTTP

Operation	Methode	URL	Corps
Lire (tous)	GET	/users	Non
Lire (un)	GET	/users/1	Non
Creer	POST	/users	Oui
Modifier	PATCH	/users/1	Oui
Remplacer	PUT	/users/1	Oui
Supprimer	DELETE	/users/1	Non

API de test

`https://jsonplaceholder.typicode.com`

HTTP - GET (Lire)

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

interface User { id: number; name: string; email: string }

const users = ref<User[]>([])
const loading = ref(false)
const error = ref<string | null>(null)

async function fetchUsers() {
  loading.value = true
  try {
    const res = await fetch('https://jsonplaceholder.typicode.com/users')
    if (!res.ok) throw new Error(`Erreur ${res.status}`)
    users.value = await res.json()
  } catch (e) {
    error.value = e.message
  } finally {
    loading.value = false
  }
}
```

HTTP - POST (Creer)

```
async function createUser(userData) {
  const response = await fetch(
    'https://jsonplaceholder.typicode.com/users',
    {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(userData)
    }
  )

  if (!response.ok) {
    throw new Error('Erreur de creation')
  }

  const newUser = await response.json()
  return newUser
}
```

Points cles

HTTP - PATCH (Modifier)

```
async function updateUser(id: number, updates: Partial<User>) {
  const response = await fetch(
    `https://jsonplaceholder.typicode.com/users/${id}`,
    {
      method: 'PATCH',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(updates)
    }
  )

  return await response.json()
}

// Utilisation
await updateUser(1, { name: 'Nouveau nom' })
```

PATCH vs PUT

PATCH : modifie partiellement • **PUT** : remplace entierement

HTTP - DELETE (Supprimer)

```
async function deleteUser(id: number) {
  const response = await fetch(
    `https://jsonplaceholder.typicode.com/users/${id}`,
    {
      method: 'DELETE'
    }
  )

  if (response.ok) {
    // Retirer de la liste locale
    users.value = users.value.filter(u => u.id !== id)
  }
}
```

Bonne pratique

Apres un DELETE reussi, mettez a jour l'etat local immediatement pour une UI reactive.

HTTP - Pattern complet

```
<template>
  <!-- Chargement -->
  <p v-if="loading">Chargement...</p>

  <!-- Erreur -->
  <p v-else-if="error" class="error">{{ error }}</p>

  <!-- Donnees -->
  <ul v-else>
    <li v-for="user in users" :key="user.id">
      {{ user.name }}
      <button @click="deleteUser(user.id)">X</button>
    </li>
  </ul>
</template>
```

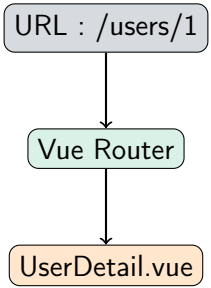
Les 3 etats

loading → error → data

Router - Introduction

Qu'est-ce que Vue Router ?

Vue Router est la bibliotheque officielle de routage pour creer des **Single Page Applications (SPA)**.



Installation : `npm install vue-router@4`

Router - Configuration

src/router/index.ts

```
import { createRouter, createWebHistory } from 'vue-router'
import Home from '@/views/Home.vue'
import Users from '@/views/Users.vue'
import UserDetail from '@/views/UserDetail.vue'

const router = createRouter({
  history: createWebHistory(),
  routes: [
    { path: '/', name: 'home', component: Home },
    { path: '/users', name: 'users', component: Users },
    { path: '/users/:id', name: 'user-detail', component: UserDetail },
    { path: '/:pathMatch(.*)*', component: NotFound }  // 404
  ]
})

export default router
```

Router - router-link et router-view

```
<template>
  <!-- Navigation -->
  <nav>
    <router-link to="/">Accueil</router-link>
    <router-link to="/users">Utilisateurs</router-link>
    <router-link :to="{ name: 'user-detail', params: { id: 1 }}">
      User 1
    </router-link>
  </nav>

  <!-- Zone d'affichage -->
  <router-view />
</template>
```

router-link

Cree un lien de navigation

Classe `.router-link-active`

router-view

Affiche le composant correspondant a la route

Router - Parametres de route

```
// Route: /users/:id
<script setup lang="ts">
import { useRoute } from 'vue-router'
import { ref, onMounted } from 'vue'

const route = useRoute()
const userId = route.params.id     // Parametre de route

const user = ref(null)

onMounted(async () => {
  const res = await fetch(`/api/users/${userId}`)
  user.value = await res.json()
})
</script>

<template>
  <h1>User {{ userId }}</h1>
  <p v-if="user">{{ user.name }}</p>
</template>
```

Router - Query parameters

```
<!-- Lien avec query -->
<router-link :to="{ path: '/search', query: { q: 'vue', page: 1 }}">
  Rechercher
</router-link>

<!-- URL: /search?q=vue&page=1 -->
```

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

const route = useRoute()

const searchQuery = route.query.q       // 'vue'
const page = route.query.page           // '1'
</script>
```

Difference

params : `/users/:id` → partie du chemin

query : `?key=value` → parametres optionnels

Router - Navigation programmatique

```
<script setup>
import { useRouter } from 'vue-router'

const router = useRouter()

// Naviguer vers une route
router.push('/')
router.push({ name: 'users' })
router.push({ name: 'user-detail', params: { id: 1 } })

// Avec query
router.push({ path: '/search', query: { q: 'vue' } })

// Remplacer (pas d'historique)
router.replace('/login')

// Retour arriere
router.back()

// Avancer
router.forward()
</script>
```

Router - Resume

Element	Usage
`<router-link to="/">`	Lien de navigation
`<router-view />`	Affiche le composant
`useRoute()`	Acces a la route actuelle
`useRouter()`	Navigation programmatique
`route.params.id`	Parametre de route
`route.query.q`	Query parameter
`router.push()`	Naviguer
`router.back()`	Retour arriere

Resume - v-model

Type	Syntaxe	Dans l'enfant
Simple	`v-model="x"`	`modelValue` + `update:modelValue`
Nomme	`v-model:nom="x"`	`nom` + `update:nom`
defineModel	`v-model="x"`	`const m = defineModel()`

Modificateurs : `.number` • `.trim` • `.lazy`

Resume - Slots

Type	Syntaxe
Par defaut	`<slot>Defaut</slot>`
Nomme (enfant)	`<slot name="header">`
Nomme (parent)	`<template #header>`
Scoped (enfant)	`<slot :item="item">`
Scoped (parent)	`<template #default="{ item }">`

Resume - HTTP

Action	Methode	Code
Lire	GET	`fetch(url)`
Creer	POST	`fetch(url, {method:'POST', body:...})`
Modifier	PATCH	`fetch(url, {method:'PATCH', body:...})`
Supprimer	DELETE	`fetch(url, {method:'DELETE'})`

Pattern : loading → error → data

Resume - Router

Template

- `<router-link to="/">`
- `<router-view />`
- `:to="{ name, params }"`

Script

- `useRoute()` → route actuelle
- `useRouter()` → navigation
- `route.params.id`
- `router.push('/path')`

Questions ?

Merci pour votre attention !