Travaux Dirigés

Vue.js : Réactivité et Directives

Mini-Projet Todo List avec Vue.js

Ibrahim ALAME

Formation Vue.js - 2025

Objectifs

Durée : 2 heures

Mettre en pratique les concepts de Vue.js :

- Création d'un projet Vue.js avec Vite
- Réactivité avec `ref()` et `computed()`
- Liaison de données avec `v-bind` et `v-model`
- Gestion des événements avec `v-on`
- Directives structurelles `v-if`, `v-for`
- Modificateurs d'événements

Prérequis : TD1 (Todo List en JavaScript), Chapitre 4 (sections 1-4)

Contents

1 Description du projet

Dans le TD1, vous avez créé une application Todo List en JavaScript pur avec manipulation directe du DOM. Dans ce TD, nous allons **recréer la même application avec Vue.js** pour constater la différence d'approche et les avantages de la réactivité.

1.1 Comparaison des approches

JavaScript pur (TD1)	**Vue.js (TD2)**
Manipulation directe du DOM	DOM géré automatiquement
`document.createElement()`	Template déclaratif
`element.innerHTML`	Interpolation `{{ }}`
`addEventListener()`	`@click`, `@keyup`
Mise à jour manuelle	Réactivité automatique
150 lignes de JS	50 lignes de Vue

Table 1: JavaScript pur vs Vue.js

1.2 Fonctionnalités à implémenter

Les mêmes que le TD1 :

1. Ajouter une nouvelle tâche
2. Marquer une tâche comme complétée
3. Supprimer une tâche
4. Filtrer les tâches (toutes / actives / complétées)
5. Afficher un compteur de tâches actives
6. Effacer les tâches complétées

2 Mise en place du projet

2.1 Créer le projet Vue.js

```
# Creer un nouveau projet Vue.js
npm create vue@latest

# Repondre aux questions :
# Project name: td2-todo-vue
# Add TypeScript? Yes
# Add JSX Support? No
# Add Vue Router? No
# Add Pinia? No
# Add Vitest? No
```

```
# Add ESLint? Yes
# Add Prettier? Yes

# Installer les dependances
cd td2-todo-vue
npm install

# Lancer le serveur de developpement
npm run dev
```

Astuce

Le serveur de développement Vite démarre sur http://localhost:5173 avec rechargement automatique (HMR).

2.2 Structure du projet

Après création, votre projet a cette structure :

```
td2-todo-vue/
 src/
    App.vue           <- Composant principal
    main.ts           <- Point d'entrée
    components/       <- Vos composants
    assets/           <- CSS, images
 index.html
 package.json
 tsconfig.json
 vite.config.ts
```

2.3 Nettoyer le projet

Supprimez le contenu par défaut et remplacez src/App.vue par :

```
<script setup lang="ts">
// Logique ici
</script>

<template>
  <div class="container">
    <h1>Ma Todo List Vue.js</h1>
  </div>
</template>

<style scoped>
.container {
  max-width: 600px;
  margin: 50px auto;
  font-family: 'Segoe UI', sans-serif;
}
</style>
```

3 Phase 1 : Structure de base (20 min)

3.1 Exercice 1 : Définir les données réactives

Créez les données réactives nécessaires pour l'application.

3.1.1 Objectifs

- Définir une interface TypeScript pour les tâches
- Créer un tableau réactif de tâches avec `ref()`
- Créer une référence pour le champ de saisie
- Créer une référence pour le filtre actif

3.1.2 Instructions

Dans la section `<script setup>`, ajoutez :

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

// Interface pour une tache
interface Task {
  id: number
  text: string
  completed: boolean
}

// Donnees reactives
const tasks = ref<Task[]>([])
const newTaskText = ref('')
const currentFilter = ref<'all' | 'active' | 'completed'>('all')

// Compteur pour generer des IDs uniques
let nextId = 1
</script>
```

Important

N'oubliez pas d'importer `ref` depuis `'vue'` !

3.2 Exercice 2 : Créer le template de base

Créez la structure HTML du template.

```
<template>
  <div class="container">
    <header>
      <h1>Ma Liste de Taches</h1>
    </header>
```

```
    <main>
      <!-- Formulaire d'ajout -->
      <div class="add-task">
        <input
          type="text"
          placeholder="Nouvelle tache..."
        >
        <button>Ajouter</button>
      </div>

      <!-- Liste des taches -->
      <ul class="task-list">
        <!-- Les taches seront affichees ici -->
      </ul>

      <!-- Pied de page avec filtres -->
      <footer class="task-footer">
        <span>0 taches actives</span>
        <div class="filters">
          <button>Toutes</button>
          <button>Actives</button>
          <button>Completees</button>
        </div>
        <button>Effacer completees</button>
      </footer>
    </main>
  </div>
</template>
```

3.3 Exercice 3 : Ajouter les styles

Ajoutez les styles CSS (identiques au TD1) :

```
<style scoped>
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

.container {
  max-width: 600px;
  margin: 50px auto;
  background: white;
  border-radius: 10px;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
  overflow: hidden;
}

header {
```

```
  background: #42b883;
  color: white;
  padding: 20px;
  text-align: center;
}

main {
  padding: 20px;
}

.add-task {
  display: flex;
  gap: 10px;
  margin-bottom: 20px;
}

.add-task input {
  flex: 1;
  padding: 10px;
  border: 2px solid #ddd;
  border-radius: 5px;
  font-size: 16px;
}

.add-task input:focus {
  outline: none;
  border-color: #42b883;
}

button {
  padding: 10px 20px;
  background: #42b883;
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  font-size: 16px;
}

button:hover {
  opacity: 0.9;
}

.task-list {
  list-style: none;
  margin-bottom: 20px;
}

.task-item {
  display: flex;
  align-items: center;
```

```
  padding: 15px;
  border-bottom: 1px solid #ddd;
}

.task-item:hover {
  background: #fafafa;
}

.task-item.completed .task-text {
  text-decoration: line-through;
  opacity: 0.6;
}

.task-item input[type="checkbox"] {
  margin-right: 15px;
  width: 20px;
  height: 20px;
  cursor: pointer;
}

.task-text {
  flex: 1;
  font-size: 16px;
}

.task-delete {
  background: #f44336;
  padding: 5px 10px;
  font-size: 14px;
}

.task-footer {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding-top: 15px;
  border-top: 2px solid #ddd;
  font-size: 14px;
}

.filters {
  display: flex;
  gap: 5px;
}

.filters button {
  background: transparent;
  color: #333;
  border: 1px solid #ddd;
  padding: 5px 15px;
  font-size: 14px;
```

```
}

.filters button.active {
  background: #42b883;
  color: white;
  border-color: #42b883;
}

.clear-btn {
  background: #f44336;
  font-size: 14px;
  padding: 5px 15px;
}
</style>
```

4 Phase 2 : Liaison de données (25 min)

4.1 Exercice 4 : v-model pour le champ de saisie

Liez le champ de saisie à la variable `newTaskText` avec `v-model`.

4.1.1 Instructions

Modifiez l'input dans le template :

```
<input
  type="text"
  v-model="newTaskText"
  placeholder="Nouvelle tache..."
>
```

4.1.2 Test

Ajoutez temporairement sous l'input pour vérifier la liaison :

```
<p>Texte saisi : {{ newTaskText }}</p>
```

Astuce

Avec `v-model`, la variable est automatiquement mise à jour quand vous tapez. C'est la **liaison bidirectionnelle** !

4.2 Exercice 5 : Fonction d'ajout de tâche

Implémentez la fonction pour ajouter une tâche.

4.2.1 Dans le script

```
function addTask() {
  // Verifier que le texte n'est pas vide
  const text = newTaskText.value.trim()
  if (!text) return

  // Creer la nouvelle tache
  tasks.value.push({
    id: nextId++,
    text: text,
    completed: false
  })

  // Vider le champ de saisie
  newTaskText.value = ''
}
```

4.2.2 Dans le template

Connectez le bouton et la touche Entrée :

```
<div class="add-task">
  <input
    type="text"
    v-model="newTaskText"
    @keyup.enter="addTask"
    placeholder="Nouvelle tache..."
  >
  <button @click="addTask">Ajouter</button>
</div>
```

Important

Notez la différence avec JavaScript pur :

- Pas de `document.getElementById()`
- Pas de `addEventListener()`
- Syntaxe déclarative : `@click="addTask"`

4.3 Exercice 6 : Afficher les tâches avec v-for

Utilisez `v-for` pour afficher la liste des tâches.

```
<ul class="task-list">
  <li
    v-for="task in tasks"
    :key="task.id"
    class="task-item"
  >
    <input type="checkbox">
```

```
    <span class="task-text">{{ task.text }}</span>
    <button class="task-delete">Supprimer</button>
  </li>
</ul>
```

Important

L'attribut `:key` est **obligatoire** avec `v-for`. Utilisez toujours un identifiant unique (ici `task.id`).

5 Phase 3 : Interactivité (30 min)

5.1 Exercice 7 : Marquer comme complétée

Ajoutez la fonctionnalité pour marquer une tâche comme complétée.

5.1.1 Dans le script

```
function toggleTask(task: Task) {
  task.completed = !task.completed
}
```

5.1.2 Dans le template

```
<li
  v-for="task in tasks"
  :key="task.id"
  class="task-item"
  :class="{ completed: task.completed }"
>
  <input
    type="checkbox"
    :checked="task.completed"
    @change="toggleTask(task)"
  >
  <span class="task-text">{{ task.text }}</span>
  <button class="task-delete">Supprimer</button>
</li>
```

Astuce

La syntaxe `:class="{ completed: task.completed }"` ajoute la classe `completed` conditionnellement.

5.2 Exercice 8 : Supprimer une tâche

Implémentez la suppression de tâches.

5.2.1 Dans le script

```
function deleteTask(taskId: number) {
  tasks.value = tasks.value.filter(t => t.id !== taskId)
}
```

5.2.2 Dans le template

```
<button class="task-delete" @click="deleteTask(task.id)">
  Supprimer
</button>
```

5.3 Exercice 9 : Compteur de tâches avec computed

Utilisez `computed()` pour calculer le nombre de tâches actives.

5.3.1 Dans le script

```
import { ref, computed } from 'vue'

// ... autres declarations ...

const activeCount = computed(() => {
  return tasks.value.filter(t => !t.completed).length
})
```

5.3.2 Dans le template

```
<span>{{ activeCount }} tache(s) active(s)</span>
```

Astuce

`computed()` est **mis en cache** : il ne recalcule que si `tasks` change. Beaucoup plus performant qu'une méthode !

6 Phase 4 : Filtres (25 min)

6.1 Exercice 10 : Filtrer les tâches

Implémentez le système de filtres.

6.1.1 Dans le script

```
// Taches filtrees (computed)
const filteredTasks = computed(() => {
  switch (currentFilter.value) {
    case 'active':
      return tasks.value.filter(t => !t.completed)
    case 'completed':
      return tasks.value.filter(t => t.completed)
    default:
      return tasks.value
  }
})

// Changer le filtre
function setFilter(filter: 'all' | 'active' | 'completed') {
  currentFilter.value = filter
}
```

6.1.2 Dans le template

Modifiez le `v-for` pour utiliser `filteredTasks` :

```
<ul class="task-list">
  <li
    v-for="task in filteredTasks"
    :key="task.id"
    class="task-item"
    :class="{ completed: task.completed }"
  >
    <!-- ... contenu ... -->
  </li>
</ul>
```

Modifiez les boutons de filtre :

```
<div class="filters">
  <button
    :class="{ active: currentFilter === 'all' }"
    @click="setFilter('all')"
  >
    Toutes
  </button>
  <button
    :class="{ active: currentFilter === 'active' }"
    @click="setFilter('active')"
  >
    Actives
  </button>
  <button
    :class="{ active: currentFilter === 'completed' }"
    @click="setFilter('completed')"
  >
    Completees
```

```
  </button>
</div>
```

6.2 Exercice 11 : Effacer les tâches complétées

6.2.1 Dans le script

```
function clearCompleted() {
  tasks.value = tasks.value.filter(t => !t.completed)
}

// Computed pour afficher/masquer le bouton
const hasCompleted = computed(() => {
  return tasks.value.some(t => t.completed)
})
```

6.2.2 Dans le template

```
<button
  v-show="hasCompleted"
  class="clear-btn"
  @click="clearCompleted"
>
  Effacer completees
</button>
```

Astuce

`v-show` masque l'élément avec `display: none` sans le supprimer du DOM. Idéal pour les éléments qui togglent souvent.

7 Phase 5 : Améliorations (20 min)

7.1 Exercice 12 : Message si liste vide

Affichez un message quand il n'y a pas de tâches.

```
<!-- Liste vide -->
<p v-if="tasks.length === 0" class="empty-message">
  Aucune tache. Ajoutez-en une !
</p>

<!-- Liste des taches -->
<ul v-else class="task-list">
  <!-- ... -->
</ul>
```

Ajoutez le style :

```
.empty-message {
  text-align: center;
  color: #888;
  padding: 40px;
  font-style: italic;
}
```

7.2 Exercice 13 : Double-clic pour éditer

Bonus

Ajoutez la possibilité d'éditer une tâche en double-cliquant dessus.

7.2.1 Dans le script

```
const editingId = ref<number | null>(null)
const editText = ref('')

function startEdit(task: Task) {
  editingId.value = task.id
  editText.value = task.text
}

function saveEdit(task: Task) {
  if (editText.value.trim()) {
    task.text = editText.value.trim()
  }
  editingId.value = null
}

function cancelEdit() {
  editingId.value = null
}
```

7.2.2 Dans le template

```
<li v-for="task in filteredTasks" :key="task.id" ...>
  <input type="checkbox" ...>

  <!-- Mode affichage -->
  <span
    v-if="editingId !== task.id"
    class="task-text"
    @dblclick="startEdit(task)"
  >
    {{ task.text }}
  </span>

  <!-- Mode edition -->
  <input
    v-else
    v-model="editText"
    @keyup.enter="saveEdit(task)"
    @keyup.escape="cancelEdit"
    @blur="saveEdit(task)"
    class="edit-input"
    autofocus
  >

  <button class="task-delete" @click="deleteTask(task.id)">
    Supprimer
  </button>
</li>
```

8 Code complet

Solution

Voici le code complet de App.vue :

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

interface Task {
  id: number
  text: string
  completed: boolean
}

// Donnees reactives
const tasks = ref<Task[]>([])
const newTaskText = ref('')
const currentFilter = ref<'all' | 'active' | 'completed'>('all')
let nextId = 1

// Computed
const activeCount = computed(() =>
  tasks.value.filter(t => !t.completed).length
)

const hasCompleted = computed(() =>
  tasks.value.some(t => t.completed)
)

const filteredTasks = computed(() => {
  switch (currentFilter.value) {
    case 'active':
      return tasks.value.filter(t => !t.completed)
    case 'completed':
      return tasks.value.filter(t => t.completed)
    default:
      return tasks.value
  }
})

// Fonctions
function addTask() {
  const text = newTaskText.value.trim()
  if (!text) return
  tasks.value.push({ id: nextId++, text, completed: false })
  newTaskText.value = ''
}

function toggleTask(task: Task) {
  task.completed = !task.completed
}

function deleteTask(id: number) {
  tasks.value = tasks.value.filter(t => t.id !== id)
}
```

9 Comparaison finale

Aspect	JavaScript (TD1)	Vue.js (TD2)
Lignes de JS/TS	120	50
Manipulation DOM	Manuelle	Automatique
Mise à jour UI	`renderTasks()`	Réactive
Événements	`addEventListener`	`@click`
Liaison données	Manuelle	`v-model`
Filtres	Fonction + re-render	`computed`

Table 2: Bilan comparatif

10 Critères d'évaluation

Critère	Points
Projet Vue.js fonctionnel	2
Données réactives correctes (ref)	2
Ajouter une tâche (v-model + @click)	3
Affichage avec v-for et :key	2
Marquer comme complétée	2
Supprimer une tâche	2
Compteur avec computed	2
Filtres fonctionnels	3
Code propre et typé	2
Total	**20**

Table 3: Grille d'évaluation

11 Pour aller plus loin

Bonus

Extensions possibles :

1. **LocalStorage** : Sauvegarder avec `watch()` et `onMounted()`
2. **Animations** : Utiliser `<Transition>` et `<TransitionGroup>`
3. **Composants** : Extraire `TaskItem.vue` et `TaskFilters.vue`
4. **Props/Emits** : Communication parent-enfant

Ces extensions seront abordées dans le TD3 (Composants).

Bon travail !

Vous avez découvert la puissance de Vue.js : moins de code, plus de lisibilité, réactivité automatique !