

# A View on Vue3



## What's the issue with Vue2?

- Supports Typescript out of the box but with limited type inference inside vue components
- Multiple alternatives like Vue Class Components or Vue with TSX tried to improve Typescript handling
- Has some limitations of the reactivity system
- Template limitations make components more difficult to build and style
- Has duplicate functionality (two way data-binding etc.)

# Example Vue2 app

```
1 export default Vue.extend({
2   components: {
3     Player: () => import('./player.js'),
4   },
5   props: {
6     player2StartValue: {
7       type: Number as PropType<number>,
8       default: 10,
9     },
10  },
11  data() {
12    return {
13      flags: {
14        // hasStarted: false,
15      },
16      players: [
17        {
18          name: 'PLAYER1',
19          score: 0,
20        },
21        {
22          name: 'PLAYER2',
23          score: 0,
24        },
25      ],
26    } as Data;
27  },
28  computed: {
29    getTotalNumberOfClicks(): number {
30      const players = this.players as Player[];
31
32      const sumOfClicks = players.reduce((total, current) => {
33        const newTotal = total + current.score;
34
35        return newTotal;
36      }, 0);
37
38      return sumOfClicks;
39    },
40    hasStarted() {
41      return this.flags.hasStarted;
42    }
43  }
44});
```

```
1 export default Vue.extend({
2   props: {
3     player: {
4       type: Object as PropType<Player>,
5       default: () => ({
6         name: 'Ted',
7         score: 0,
8       }),
9     },
10  },
11  methods: {
12    handleClick(): void {
13      this.$emit('update-player');
14    },
15  },
16  template: `
17    <div>
18      <dl>
19        <dt>{{ player.name }}</dt>
20        <dd>{{ player.score }}</dd>
21      </dl>
22      <button
23        type="button"
24        @click="handleClick()"
25      >
26        Click
27      </button>
28    </div>
29  `,
30  mounted() {
31    this.$emit('update-player', this.player);
32  }
33});
```

New features & improvements

# Composition API

- Improves compatibility with Typescript
- Improves readability/code organization
- Provides data encapsulation
- Simplifies scope handling
- Further fragmentation of the Vue ecosystem
- Changes approach to unit testing
- Still actively changing (script setup improvements)

```
● ● ●  
1 export default defineComponent({  
2   setup() {  
3     const counter = ref<number>(0);  
4     const hasBeenClicked = computed<boolean>(() => counter.value !== 0);  
5  
6     function handleClick(): void {  
7       counter.value += 1;  
8     }  
9  
10    return {  
11      counter,  
12      hasBeenClicked,  
13      handleClick,  
14    };  
15  },  
16  template: `  
17    <h1>  
18      <template v-if="hasBeenClicked">Has started</template>  
19      <template v-else>Has not started</template>  
20    </h1>  
21    <button  
22      type="button"  
23      @click="handleClick"  
24    >  
25      Clicked {{ counter }} time(s)  
26    </button>  
27  `,  
28});
```

Example Composition API

## Better reactivity

- Vue3 uses proxy objects for reactivity
- Objects are now fully reactive by default without any change detection issues
- Objects and Arrays can be made partially reactive
- Makes destructuring slightly more complicated

# Multiple v-model attributes per component

- Vue3 allows multiple v-models per component
- Two-way data-binding is mostly used for form-elements within components and child to parent communication on custom components

• • •

```
1 export default defineComponent({
2   props: {
3     firstName: {
4       type: String as PropType<string>,
5       default: '',
6     },
7     lastName: {
8       type: String as PropType<string>,
9       default: '',
10    },
11  },
12  computed: {
13   firstNameValue: {
14     set(newValue: string): void {
15       this.$emit('update:firstName', newValue);
16     },
17     get(): string {
18       return this.firstName;
19     },
20   },
21 },
22 methods: {
23   handleInput(event: InputEvent): void {
24     if (event.target === null) {
25       throw new Error('input field missing');
26     }
27     const targetElement = event.target as HTMLInputElement;
28
29     this.$emit('update:lastName', targetElement.value);
30   },
31 },
32 template: `
33   <div>
34     <label for="first-name">First name: </label>
35     <input v-model="firstNameValue" id="first-name" />
36   </div>
37   <div>
38     <label for="last-name">Last name: </label>
39     <input :value="lastName" @input="handleInput" id="last-name" />
40   </div>
41 `,
42 });
});
```

# Event-validation and event-documentation

- Events can now be validated via the `emits`-property
- See at a glance which events are emitted by a component and check the validity of emitted event parameters
- Eslint shows warning when events that are used are missing in the `emits` property
- Can be simple string array for documentation only
- Can be an object of callback functions for validator functionality



```
1 export default defineComponent({
2   props: {
3     text: {
4       type: String as PropType<string>,
5       default: '',
6     },
7   },
8   emits: {
9     'update-text': (newValue: string) => {
10       const nonLatinCharactersRegex = /^[^\x00-\x7F]+/;
11
12       if (nonLatinCharactersRegex.test(newValue)) {
13         return false;
14       }
15
16       return true;
17     },
18     'toggle-button': null, // disable validation
19   },
20   computed: {
21     textValue: {
22       set(newValue: string): void {
23         this.$emit('update-text', newValue);
24       },
25       get(): string {
26         return this.text;
27       },
28     },
29   },
30   methods: {
31     handleButtonClick(): void {
32       this.$emit('toggle-button');
33     },
34   },
35   template: `
```

## Multiple root nodes in templates

- Templates can now have multiple root elements
- No more wrapper elements necessary
- Class-attribute placement needs to be handled manually
- Key placement slightly more complex when using template-tags with v-for



```
1 export default defineComponent({
2   data() {
3     return {
4       hasLoaded: false,
5     };
6   },
7   mounted() {
8     setTimeout(() => {
9       this.hasLoaded = true;
10      }, 5000);
11    },
12   template: `
13     <div v-if="!hasLoaded">
14       <marquee>
15         Table is loading. Please wait!
16       </marquee>
17     </div>
18     <table v-else>
19       <tr>
20         <th>Head 1</th>
21         <th>Head 2</th>
22       </tr>
23       <tr>
24         <td>Column 1</td>
25         <td>Column 2</td>
26       </tr>
27     </table>
28   `,
29   methods: {
30     handleLoad() {
31       this.$emit('load');
32     }
33   }
34 });
35 
```

# Teleport

- New tag to break out of the current component hierarchy
- Elements are appended to the body tag by default and act as regular child elements (receive props etc.)
- Avoids common stacking context issues and containing block issues
- Mainly useful for overlays, toast messages etc.
- Multiple teleports to the same target will be appended in the order they are called

```
● ● ●  
1 export default defineComponent({  
2   data() {  
3     return {  
4       overlayIsVisible: false,  
5       overlayNumberOfClicks: 0,  
6     };  
7   },  
8   methods: {  
9     closeOverlay(): void {  
10      this.overlayIsVisible = false;  
11      this.overlayNumberOfClicks = 0;  
12    },  
13    openOverlay(): void {  
14      this.overlayIsVisible = true;  
15    },  
16    clickInsideOverlay(): void {  
17      this.overlayNumberOfClicks += 1;  
18    },  
19  },  
20  template: `  
21    <div>  
22      <h1>  
23        Clicks inside modal: {{ overlayNumberOfClicks }}  
24      </h1>  
25      <button @click="overlayIsVisible ? closeOverlay() : openOverlay()">  
26        {{ overlayIsVisible ? 'Hide' : 'Show' }} modal  
27      </button>  
28      <teleport to="body" v-if="overlayIsVisible">  
29        <div class="overlay">  
30          <button @click="clickInsideOverlay">  
31            Click  
32          </button>  
33          <button @click="closeOverlay">  
34            Close modal  
35          </button>  
36        </div>  
37      </teleport>  
38    </div>  
39  `,  
40});
```

Example teleport feature



## Filter removal

- Filter syntax was originally taken from AngularJS
- Often used for output formatting of currencies, dates and strings
- Pure functions with no access to component state
- Can be replaced with regular functions or computed properties

```
● ● ●

1 export default Vue.extend({
2   filters: {
3     capitalizeLastLetters(value: string | null): string {
4       if (!value) {
5         return '';
6       }
7
8       const words: string[] = value.trim().split(/\s+/);
9       const wordsReversed: string[] = words
10      .map((word) => word.split('').reverse().join(''));
11
12      const wordsReversedCapitalized: string[] = wordsReversed.map((word) => {
13        const firstLetter = word.charAt(0).toUpperCase();
14        const remainingLetters = word.slice(1).toLowerCase();
15        const newWord = `${firstLetter}${remainingLetters}`;
16
17        return newWord;
18      });
19
20      const wordsCapitalized: string[] = wordsReversedCapitalized
21      .map((word) => word.split('').reverse().join(''));
22      const wordsCapitalizedString: string = wordsCapitalized.join(' ');
23
24      return wordsCapitalizedString;
25    },
26  },
27  data() {
28    return {
29      text: '',
30    };
31  },
32  template: `
33    <div class="root">
34      <input v-model="text" placeholder="Please enter a text" />
35      <p>
36        Filtered: {{ text | capitalizeLastLetters }}
37      </p>
38    </div>
39  `,
40});
```

Example filters Vue2

## v-if & v-for precedence swap

- Using both directives at the same time should be avoided and will result in a linter warning
- Vue2 executes **v-for** first and creates a temporary object which is then used by **v-if** to check if the condition is fulfilled.
- Vue3 executes **v-if** first and doesn't render anything as there is no age variable in scope, which causes a warning
- Can be refactored into a computed property that filters and caches the results
- For conditional rendering an additional computed property like hasResults can be used

```
 1 export default Vue.extend({
 2   data() {
 3     return {
 4       ageLimit: 5,
 5       people: [
 6         {
 7           name: 'Loona Loonie',
 8           age: 1,
 9         },
10         {
11           name: 'Ted Tenner',
12           age: 10,
13         },
14         {
15           name: 'Franklin Fiver',
16           age: 5,
17         },
18       ],
19     };
20   },
21   computed: {
22     filteredPeople() {
23       const people = this.people as Person[];
24       const ageLimit = this.ageLimit as number;
25
26       return people.filter((person) => person.age >= ageLimit);
27     },
28   },
29   template: `
30     <div class="root">
31       <dl v-for="person in people" v-if="person.age >= 5">
32         <dt>Name: </dt>
33         <dd>{{ person.name }}</dd>
34         <dt>Age: </dt>
35         <dd>{{ person.age }}</dd>
36       </dl>
37       <hr />
38       <dl v-for="person in filteredPeople">
39         <dt>Name: </dt>
40         <dd>{{ person.name }}</dd>
41         <dt>Age: </dt>
```

## .sync removal and v-model name changes in custom components

- .sync-attribute has been dropped in Vue3 and needs to be replaced with v-model
- Name of v-model props have changed and differ between components with single v-model and multiple v-models
- For custom components that use a single v-model attribute, the naming convention is the following:
  - Name of the v-model attribute on the parent component: **v-model** or **v-model:theValue**
  - Name of the emitted event: **update:theValue**
  - V-model prop name: **theValue**



```
1 const radioButtonStates = {
2   on: 'On',
3   off: 'Off',
4 } as const;
5 type RadioButtonStates = keyof typeof radioButtonStates
6
7 export default defineComponent({
8   props: {
9     modelValue: {
10       type: String as PropType<RadioButtonStates>,
11       default: 'off',
12     },
13   },
14   emits: [
15     'update:modelValue',
16   ],
17   computed: {
18     modelValueComputed: {
19       set(newValue: RadioButtonStates): void {
20         this.$emit('update:modelValue', newValue);
21       },
22       get(): string {
23         return this.modelValue;
24       },
25     },
26   },
27 }
```

Example single v-model

## .sync removal and v-model name changes in custom components

- For custom components that use multiple v-model attributes, the naming convention is the following:
- Name of the v-model attribute on the parent component: **v-model:nameOfVariable**
- Name of the emitted event: **update:nameOfVariable**
- V-model prop name: **nameOfVariable** not **modelValue:nameOfVariable**

```
● ● ●

1 export default defineComponent({
2   props: {
3     firstName: {
4       type: String as PropType<string>,
5       default: '',
6     },
7     lastName: {
8       type: String as PropType<string>,
9       default: '',
10    },
11  },
12  computed: {
13   firstNameValue: {
14     set(newValue: string): void {
15       this.$emit('update:firstName', newValue);
16     },
17     get(): string {
18       return this.firstName;
19     },
20   },
21 },
22 methods: {
23   handleInput(event: InputEvent): void {
24     if (event.target === null) {
25       throw new Error('input field missing');
26     }
27     const targetElement = event.target as HTMLInputElement;
28
29     this.$emit('update:lastName', targetElement.value);
30   },
31 },
32 template: `
33   <div>
34     <label for="first-name">First name: </label>
35     <input v-model="firstNameValue" id="first-name" />
36   </div>
37   <div>
38     <label for="last-name">Last name: </label>
39     <input :value="lastName" @input="handleInput" id="last-name" />
40   </div>
41 `,
42 });

```

## Other minor breaking changes

- Key placement on multi-root components
- Template tag rendering
- Browser support

