

Компоненти. v-model

*Батьківський компонент передає дочірньому дані через **властивості-атрибути***

```
//----- Дочірній компонент -----
<template>
    . . . Опис розмітки . .
</template>

<script>
    export default {
        . . . .
        props: {
            властивість_1: {
                . . . опис властивості . .
            }
        . . . .
    },
    </script>
    <style> . . . . . </style>
```

```
//----- Батьківський компонент -----
<template>
    . . . . . .
    <дочірній
        властивість = “ значення_string ”
        : властивість = “ обчислюване значеня ”
    />
    . . . . .
</template>
```

```
<script>
    . . . . .
</script>
<style>
    . . . . .
</style>
```

Батьківський компонент передає дочірньому дані через властивості

```
//----- Дочірній компонент -----
<template>
    . . . Опис розмітки . . .
</template>

<script>
    export default {
        . . . . .
        props: {
            властивість_1: {
                ... опис властивості ...
            }
        . . . . .
    },
</script>
<style> . . . . . . . . . </style>
```

//----- Батьківський компонент -----

```
<template>
    . . .
<дочірній>
    властивість = “ значення string ”
    : властивість = “ обчислюване значення ”
/>
    . . .

</template>

<script>
    . . .
</script>
<style>
    . . .
</style>
```

Батьківський компонент передає дочірньому дані через властивості

```
//----- Дочірній компонент -----
<template>
    . . . Опис розмітки . . .
</template>

<script>
    export default {
        .....
        props: {
            властивість_1: {
                ... опис властивості ...
            }
        .....
    },
</script>
<style> . . . . . . . . . </style>
```

```
//----- Батьківський компонент -----
<template>
    . . .
    <дочірній>
        . . .
        <div>
            <div> властивість = “ значення string ”
            . . .
            : <div> властивість = “ обчислюване значення ”
        </div>
        . . .
    </дочірній>
    . . .
</template>

<script>
    . . .
</script>
<style>
    . . .
</style>
```

- 1) батьківський передає дочірньому компоненту дані через властивості-атрибути у *props* дочірнього
 - 2) при присвоєнні нових значень у батьківському компоненті дочірній автоматично оновлюється

Передача вхідних параметрів-властивостей (передача компоненту даних)

The diagram illustrates the flow of props from the parent component (`App.vue`) to the child component (`BlogPost.vue`). A red arrow points from the `title` prop in `App.vue` to its declaration in `BlogPost.vue`. A green arrow points from the `likes` prop in `App.vue` to its declaration in `BlogPost.vue`.

Дочірній компонент (`BlogPost.vue`):

```
src > components > BlogPost.vue > {} "BlogPost.vue" > template > div.t
1  <template>
2  <div class="task-container">
3  <div>{{ title }}</div>
4  <div>{{ likes }}</div>
5  <button @click="onLike">Like</button>
6  <button @click="onDislike">Dislike</button>
7 </div>
8 </template>
9
10 <script>
11 export default {
12   name: 'BlogPost',
13
14   props: {
15     title: {
16       type: String,
17       default: 'Guest',
18     },
19     likes: {
20       type: Number,
21       default: 0,
22     },
23   },
24
25   > methods: { ...
26 }
```

Батьківський компонент (`App.vue`):

```
src > App.vue src M > {} "App.vue"
1  You, 42 seconds ago | 1 author (You)
2  <template>
3  <blog-post title="Моя подорож з Vue" :likes="25" />
4
5  <script>
6  import BlogPost from './components/BlogPost.vue'
7
8  export default {
9    name: 'App',
10   components: {
11     BlogPost,
12   },
13 }
14 </script>
15
16 > <style>...
17
18
19
20
21
22
23
24
25
26
```

UI preview at the bottom shows the component rendering with the title "Моя подорож з Vue" and likes count 25.

Передача **даних** від дочірнього компонента у батьківський

Дочірній компонент

```
<template>
  . . .
</template>
<script>
export default {
  name: ". . . . ."
  methods: {
    метод () {
      .....
      this.$emit('назва_події', дані )
      .....
    },
    .....
  }
}
</script>
<style> ... </style>
```

1. Дочірній компонент
генерує **подію** і за
потреби передає **дані**

Передача **даних** від дочірнього компонента у батьківський

Дочірній компонент	Батьківський компонент
<pre><template> . . . </template> <script> export default { name: "." methods: { метод () { . . . this.\$emit('назва_події', дані) . . . }, . . . } } <script> <style> ... </style></pre>	<pre><template> . . . <дочірній_компонент @назва_події=" обробник_події " /> . . . </template> <script> export default { name: "." methods: { . . . обробник_події (дані) { . . . }, . . . } } <script> <style> ... </style></pre>

1. Дочірній компонент генерує подію і за потреби передає дані

2. Для дочірнього компонента призначаємо обробника події

Передача **даних** від дочірнього компонента у батьківський

Дочірній компонент	Батьківський компонент
<pre><template> . . . </template> <script> export default { name: "...", methods: { метод () { this.\$emit('назва_події', дані) }, } }, </script> <style> ... </style></pre> <p>1. Дочірній компонент генерує подію і за потреби передає дані</p>	<pre><template> . . . <дочірній компонент @назва_події=" обробник_події " /> . . . </template> <script> export default { name: "...", methods: { обробник_події (дані) { }, } } </script> <style> ... </style></pre> <p>2. Для дочірнього компонента призначаємо обробника події</p> <p>3. Викликається обробник події</p>

Передача **даних** від дочірнього компонента у батьківський

Дочірній компонент	Батьківський компонент
<pre><template> . . . </template> <script> export default { name: "...", methods: { метод () { this.\$emit('назва_події', дані) }, } } <script> ... </script> <style> ... </style></pre> <p>1. Дочірній компонент генерує подію і за потреби передає дані</p>	<pre><template> . . . <дочірній компонент @назва_події=" обробник_події " /> . . . </template> <script> export default { name: "...", methods: { обробник_події (дані) { }, } } <script> ... </script> <style> ... </style></pre> <p>2. Для дочірнього компонента призначаємо обробника події</p> <p>3. Викликається обробник події</p> <p>4. Обробник події отримує передані разом з подією дані</p>

▼ PostCard.vue components U ●

src > components > ▼ PostCard.vue > {} "PostCard.vue"

```
1  <template>
2  <div class="container">
3  <div class="logo-container">
4  
5  <div v-else class="logo">{{ firstLetter }}</div>
6  </div>
7  <div class="content">
8  <div>{{ cardData.userName }}</div>
9  <div>{{ cardData.message }}</div>
10 <div>
11   <button @click="$emit('like',cardData.id)">
12     Like
13   </button>
14   <span>{{ cardData.likes }}</span>
15   <button @click="$emit('dislike',cardData.id)">
16     DisLike
17   </button>
18 </div>
19 </div>
20 </div>
21 </template>
22
23 <script>
24 > export default { ...
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39 }
40 </script>
```

... ●

▼ PostsPanel.vue components U X

src > components > ▼ PostsPanel.vue > {} "PostsPan...

```
1  <template>
2  <post-card
3  <div class="list-item" v-for="card in postsList" :key="card.id" :card-data="card" @like="onLike" @dislike="onDislike">
4  </post-card>
5  </div>
6  </template>
7
8 <script>
9 import PostCard from './PostCard.vue'
10 export default {
11   components: { PostCard },
12   name: 'PostsPanel',
13   props: { ... },
14   methods: {
15     onLike(cardId) { ... },
16     onDislike(cardId) { ... },
17   },
18 }
19 </script>
```

▼ PostCard.vue components U ● Дочірній компонент

src > components > ▼ PostCard.vue > {} "PostCard.vue"

```
1  <template>
2    <div class="container">
3      <div class="logo-container">
4        
5        <div v-else class="logo">{{ firstLetter }}</div>
6      </div>
7      <div class="content">
8        <div>{{ cardData.userName }}</div>
9        <div>{{ cardData.message }}</div>
10       <div>
11         <button @click="$emit('like', cardData.id)">
12           Like
13         </button>
14         <span>{{ cardData.likes }}</span>
15         <button @click="$emit('dislike', cardData.id)">
16           DisLike
17         </button>
18       </div>
19     </div>
20   </div>
21 </template>
22
23 <script>
24 > export default { ...
25
26
27
28
29
30
31
32
33 </script>
```

1. Дочірній компонент генерує подію і передає дані

▼ PostsPanel.vue components U ● Батьківський компонент

src > components > ▼ PostsPanel.vue > {} "PostsPan

```
1  <template>
2    <post-card
3      v-for="card in postsList"
4        :key="card.id"
5        :card-data="card"
6        @like="onLike"
7        @dislike="onDislike"
8    >
9  </template>
10
11 <script>
12 import PostCard from './PostCard.vue'
13 export default {
14   components: { PostCard },
15   name: 'PostsPanel',
16
17   > props: { ...
18
19
20
21
22
23
24   methods: {
25     > onLike(cardId) { ...
26
27     },
28     > onDislike(cardId) { ...
29
30     },
31
32   }
33 </script>
```

▼ PostCard.vue components U ● Дочірній компонент

src > components > ▼ PostCard.vue > {} "PostCard.vue"

```
1  <template>
2  .  .  . <div class="container">
3  .  .  .   <div class="logo-container">
4  .  .  .     
5  .  .  .   <div v-else class="logo">{{ firstLetter }}</div>
6  .  .  </div>
7  .  .  <div class="content">
8  .  .  . <div>{{ cardData.userName }}</div>
9  .  .  . <div>{{ cardData.message }}</div>
10 .  .  <div>
11 .  .  .   <button @click="$emit('like', cardData.id)">
12 .  .  .     Like
13 .  .  .   </button>
14 .  .  .   <span>{{ cardData.likes }}</span>
15 .  .  .   <button @click="$emit('dislike', cardData.id)">
16 .  .  .     DisLike
17 .  .  .   </button>
18 .  .  </div>
19 .  .  </div>
20 .  </div>
21 </template>
22
23 <script>
24 > export default { ...
25
26 }
27 </script>
```

1. Дочірній компонент генерує подію і передає дані

... ▼ PostsPanel.vue components U X Батьківський компонент

src > components > ▼ PostsPanel.vue > {} "PostsPan...

```
1  <template>
2  .  .  . <post-card
3  .  .  .   v-for="card in postsList"
4  .  .  .   :key="card.id"
5  .  .  .   :card-data="card"
6  .  .  .   @like="onLike"
7  .  .  .   @dislike="onDislike"
8  .  .  >
9  .  </template>
10 . <script>
11 import PostCard from './PostCard.vue'
12 export default {
13   components: { PostCard },
14   name: 'PostsPanel',
15   props: { ... },
16   methods: {
17     onLike(cardId) { ... },
18     onDislike(cardId) { ... },
19   },
20 }
21 </script>
```

2. На дочірній компонент призначаємо обробника

PostCard.vue components U

src > components > PostCard.vue > {} "PostCard.vue"

```
1  <template>
2    <div class="container">
3      <div class="logo-container">
4        
5        <div v-else class="logo">{{ firstLetter }}</div>
6      </div>
7      <div class="content">
8        <div>{{ cardData.userName }}</div>
9        <div>{{ cardData.message }}</div>
10       <div>
11         <button @click="$emit('like', cardData.id)">
12           Like
13         </button>
14         <span>{{ cardData.likes }}</span>
15         <button @click="$emit('dislike', cardData.id)">
16           DisLike
17         </button>
18       </div>
19     </div>
20   </div>
21 </template>
22
23 <script>
24 > export default { ...
25 }
26 </script>
```

PostsPanel.vue components U

src > components > PostsPanel.vue > {} "PostsPanel.vue"

```
1  <template>
2    <post-card
3      v-for="card in postsList"
4      :key="card.id"
5      :card-data="card"
6      @like="onLike"
7      @dislike="onDislike"
8    />
9  </template>
10
11 <script>
12 import PostCard from './PostCard.vue'
13 export default {
14   components: { PostCard },
15   name: 'PostsPanel',
16
17   props: { ... },
18
19   methods: {
20     onLike(cardId) { ... },
21     onDislike(cardId) { ... },
22   },
23 }
24 </script>
```

The diagram illustrates the flow of an event from a child component (`PostCard.vue`) to a parent component (`PostsPanel.vue`) in a Vue.js application.

PostCard.vue (Child Component)

```
src > components > ▼ PostCard.vue > {} "PostCard.vue"
1  <template>
2  ...
3  <div class="container">
4  ...
5  <div v-if="cardData.logo" :src="cardData.logo" class="logo" />
6  ...
7  <div class="content">
8  ...
9  <div>{{ cardData.userName }}</div>
10 ...
11 <button @click="$emit('like', cardData.id)">
12   Like
13 </button>
14 <span>{{ cardData.likes }}</span>
15 <button @click="$emit('dislike', cardData.id)">
16   DisLike
17 </button>
18 ...
19 ...
20 ...
21 </div>
22 ...
23 <script>
24 > export default { ...
25 ...
26 ...
27 ...
28 ...
29 ...
30 ...
31 ...
32 ...
33 </script>
```

PostsPanel.vue (Parent Component)

```
src > components > ▼ PostsPanel.vue > {} "PostsPanel.vue"
1  <template>
2  ...
3  <post-card
4  ...
5  :card-data="card"
6  ...
7  @like="onLike"
8  ...
9  </template>
10 ...
11 <script>
12 import PostCard from './PostCard.vue'
13 export default {
14   ...
15   components: { PostCard },
16   ...
17   props: { ...
18   },
19   ...
20   methods: {
21     onLike(cardId) { ...
22     },
23     onDislike(cardId) { ...
24     },
25     ...
26   },
27   ...
28   ...
29   ...
30   ...
31   ...
32 ...
33 </script>
```

The flow of the event is as follows:

1. На дочірній компонент призначаємо обробника (Assign an event handler to the child component). The `@like` and `@dislike` directives are highlighted with red and purple boxes respectively.
2. Викликається обробник події (The event handler is called). The `onLike` and `onDislike` methods are highlighted with purple boxes.
3. Обробник події отримує передані дані (The event handler receives passed data). The `cardId` parameter is highlighted with an orange box.

v-model на нативному елементі

Як ми використовуємо **v-model**
для стандартних компонентів ?

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

```
data() {
  return {
    searchText: null
  }
},
```

v-model на нативному елементі

Як ми використовуємо **v-model**
для стандартних компонентів ?

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

```
data() {
  return {
    searchText: null
  },
},
```

Насправді тут
1)визначачається значення властивості **:value**
2)аналізується подія **@input**

```
<input
  :value="searchText"
  @input="searchText = $event.target.value"
/>
```

```
data() {
  return {
    searchText: null
  },
},
```

v-model для власних компонентів у Vue 2

Власний компонент	Батьківський компонент	
<pre><template> . . . </template> <script> export default { name: "...", props:{}, value:{ type: ... , default: ... }, }, methods: { метод () { this.\$emit('input', дані) } }, }</pre>	<pre><template> . . . <дочірній_компонент> :value = " модель_даніх " @input = " модель_даніх = \$event " /> . . . </template> <script> export default { name: "...", data: { модель_даніх : почткове_значення ; }, } </script> <style> ... </style></pre>	При зміні моделі даних дані передаються у властивість value компонента

Власний компонент

Батьківський компонент

value:{

- 1)описуємо власт. **value**
- 2)при зміні генеруємо подію **input**

v-model для власних компонентів у Vue 2

Власний компонент	Батьківський компонент
<pre><template> . . . </template> <script> export default { name: "...", props: { value: { type: ... , default: ... }, }, methods: { [метод] () { this.\$emit('input', <u>дані</u>) } } } </script> <style> ... </style></pre> <p>При необідності зміни значення властивості value компонент ... генерує подію input</p>	<pre><template> . . . <дочірній_компонент :value = " модель_даних " @input = " модель_даних = \$event " /> . . . </template> <script> export default { name: "...", data: { модель_даних : почткове_значення ; }, } </script> <style> ... </style></pre>

v-model для власних компонентів у Vue 2

Власний компонент	Батьківський компонент
<pre><template> . . . </template> <script> export default { name: ".", props:{}, value:{ type: ... , default: ... }, }, methods: { метод () { this.\$emit('input', дані) } }, } </script> <style> ... </style></pre>	<pre><template> . . . <дочірній_компонент v-model = " модель_даніх " /> . . . </template> <script> export default { name: "." data: { модель_даніх : початкове_значення ; }, }, } </script> <style> ... </style></pre> <p>v-model – це спрощений запис для:</p> <p>:value = “ модель_даніх ” @input = “ модель_даніх = \$event ”</p>

Модифікатор .sync (аналог v-model) для власних компонентів у Vue 2

Власний компонент	Батьківський компонент
<pre><template> . . . </template> <script> export default { name: ".", props:{ . . . назва_властивості:{ type: ... , default: ... }, . . . }, methods: { метод () { . . . this.\$emit('update: назва_властивості', дані) . . . } }, . . . } </script> <style> . . . </style></pre>	<pre><template> . . . <дочірній_компонент :назва_властивості.sync = " модель_даних " /> . . . </template> <script> export default { name: "." data: { . . . модель_даних : початкове_значення ; . . . }, . . . }, . . . } </script> <style> . . . </style></pre>

У Vue 3 вже немає !!!

v-model для власних компонентів у Vue 3

Власний компонент у Vue 2	Власний компонент у Vue 3
<pre><template> ... </template> <script> export default { name: "...", props: { ... value: { type: ..., default: ... }, ... }, methods: { метод () { ... this.\$emit('input', дані) ... } }, ... } </script> <style> ... </style></pre>	<pre><template> ... </template> <script> export default { name: "...", props: { ... modelValue: { type: ..., default: ... }, ... }, methods: { метод () { ... this.\$emit('update:modelValue', дані) ... } }, ... } </script> <style> ... </style></pre>

При переході до Vue 3 змінились назви властивості і події

- назва властивості:
`value` → `modelValue`
- подія:
`input` → `update:modelValue`

v-model для власних компонентів у Vue 3

Власний компонент	Батьківський компонент
<pre><template> . . . </template> <script> export default { name: "...", props: { modelValue:{ type: ... , default: ... }, }, methods: { метод () { this.\$emit('update:modelValue', дані) } }, } </script> <style> ... </style></pre>	<pre><template> . . . <дочірній_компонент v-model = " модель_даніх "/> . . . </template> <script> export default { name: "...", data: { модель_даніх : почткове_значення ; }, }, } </script> <style> ... </style></pre>

The image shows two code editors side-by-side, illustrating the implementation of a custom Vue.js component.

CustomInput.vue

```
1 <template>
2   <label>
3     {{ title }}
4     <input
5       type="text"
6       :value="modelValue"
7       @input="$emit('update:modelValue',
8         $event.target.value)">
9   </label>
10 </template>
11
12 <script>
13 export default {
14   name: 'CustomInput',
15
16   props: {
17     title: {
18       type: String,
19       default: '',
20     },
21     modelValue: {
22       type: String,
23     },
24   },
25 }
26 </script>
```

App.vue

```
1 <template>
2   <custom-input title="User name" v-model="myValue" />
3   <div>User name: {{ myValue }}</div>
4 </template>
5
6 <script>
7 import CustomInput from './components/CustomInput.vue'
8
9 export default [
10   name: 'App',
11
12   components: {
13     CustomInput,
14   },
15
16   data() {
17     return {
18       myValue: null,
19     }
20   },
21 ]
22 </script>
23
24 > <style> ... </style>
25
26
```

A red arrow points from the `v-model="myValue"` attribute in the `App.vue` template to the `modelValue` prop in the `CustomInput.vue` script. Another red arrow points from the `$emit('update:modelValue', $event.target.value)` event handler in `CustomInput.vue` to the `modelValue` prop in the `CustomInput.vue` script. A third red arrow points from the `modelValue` prop in the `CustomInput.vue` script to the `myValue` data property in the `App.vue` script.

Output:

User name User name: Ivan

v-model для власних компонентів у Vue 3

Власний компонент	Батьківський компонент
<pre><template> ... <input type="text" v-model="currentValue" /> ... </template> <script> export default { name: ".....", props: { ... modelValue: { type: ... , default: ... }, ... }, computed: { currentValue : { get(){ return this.modelValue }, set(newValue){ this.\$emit('update:modelValue', newValue) } }, ... } </script> <style> ... </style></pre> <p>Спрощений запис з використанням computed</p>	<pre><template> ... <дочірній_компонент v-model = " модель_даніх " /> ... </template> <script> export default { name: ".....", data: { ... модель_даніх : початкове_значення ; ... }, ... } </script> <style> ... </style></pre>

v-model для власних компонентів у Vue 3

Власний компонент	Батьківський компонент
<pre><template> ... <input type="text" v-model="currentValue" /> ... </template> <script> export default { name: "...", props: { ... modelValue: { type: ..., default: ... }, ... }, computed: { currentValue : { get(){ return this.modelValue }, set(newValue){ this.\$emit('update:modelValue', newValue) } }, ... } </script> <style> ... </style></pre> <p>Спрощений запис з використанням computed</p>	<pre><template> ... <дочірній_компонент v-model = " модель_даніх " /> ... </template> <script> export default { name: "...", data: { ... модель_даніх : початкове_значення ; ... }, ... } </script> <style> ... </style></pre> <p>Передача даних дочірньому компоненту</p>

Власний компонент	Батьківський компонент
<pre> <template> ... <input type="text" v-model="currentValue" /> ... </template> <script> export default { name: ".....", props: { ... modelValue: { type: ... , default: ... }, ... }, computed: { currentValue : { get(){ return this.modelValue }, set(newValue){ this.\$emit('update:modelValue', newValue) } }, ... } </script> <style> ... </style> </pre>	<pre> <template> ... <дочірній_компонент v-model = " модель_даніх " /> ... </template> <script> export default { name: ".....", data: { ... модель_даніх : початкове_значення ; ... }, ... } </script> <style> ... </style> </pre>

Спрощений запис
з використанням
computed

Передача даних батьківському
компоненту

The diagram illustrates the data flow between two Vue components: `CustomInput.vue` and `App.vue`.

CustomInput.vue (Left Panel):

```
src > components > CustomInput.vue > {} "CustomInput.vue" > template
1 <template>
2   <label>
3     {{ title }}
4     <input type="text" v-model="currentValue" />
5   </label>
6 </template>
7
8 <script>
9 export default {
10   name: 'CustomInput',
11
12   props: {
13     title: {
14       type: String,
15       default: '',
16     },
17     modelValue: {
18       type: String,
19     },
20   },
21
22   computed: {
23     currentValue: {
24       get() {
25         return this.modelValue
26       },
27       set(newVal) {
28         this.$emit('update:modelValue', newVal)
29       },
30     },
31   },
32 }
33 </script>
34
35 <style lang="css" scoped></style>
```

A blue arrow points from the `v-model="currentValue"` attribute in the `CustomInput` template to the `currentValue` computed property in the script.

App.vue (Right Panel):

```
src > App.vue > {} "App.vue"
You, 5 seconds ago | 1 author (You)
1 <template>
2   <custom-input title="User name" v-model="myValue" />
3   <div>User name: {{ myValue }}</div>
4 </template>
5
6 <script>
7 import CustomInput from './components/CustomInput.vue'
8
9 export default {
10   name: 'App',
11
12   components: {
13     CustomInput,
14   },
15
16   data() {
17     return {
18       myValue: null,
19     }
20   },
21 }
22 </script>
23
24 > <style> ... </style>
25
26
```

The `myValue` variable in the `App` script corresponds to the `currentValue` prop of the `CustomInput` component, demonstrating the two-way data binding mechanism.

CustomInput.vue components U

src > components > CustomInput.vue > {} "CustomInput.vue" > template

```
1 <template>
2   <label>
3     {{ title }}
4     <input type="text" v-model="currentValue" />
5   </label>
6 </template>
7
8 <script>
9 export default {
10   name: 'CustomInput',
11
12   props: {
13     title: {
14       type: String,
15       default: '',
16     },
17     modelValue: {
18       type: String,
19     },
20   },
21
22   computed: {
23     currentValue: {
24       get() {
25         return this.modelValue
26       },
27       set(newVal) {
28         this.$emit('update:modelValue', newVal)
29       },
30     },
31   },
32 }
33 </script>
34
35 <style lang="css" scoped></style>
```

App.vue src M

You, 5 seconds ago | 1 author (You)

```
1 <template>
2   <custom-input title="User name" v-model="myValue" />
3   <div>User name: {{ myValue }}</div>
4 </template>
5
6 <script>
7   import CustomInput from './components/CustomInput.vue'
8
9 export default {
10   name: 'App',
11
12   components: {
13     CustomInput,
14   },
15
16   data() {
17     return {
18       myValue: null,
19     }
20   },
21 }
22 </script>
23
24 > <style> ... </style>
25
26
27
28
29
30
31
32
33
34
```

CustomInput.vue components U

src > components > CustomInput.vue > {} "CustomInput.vue" > template

```
1 <template>
2   <label>
3     {{ title }}
4     <input type="text" v-model="currentValue" />
5   </label>
6 </template>
7
8 <script>
9 export default {
10   name: 'CustomInput',
11
12   props: {
13     title: {
14       type: String,
15       default: '',
16     },
17     modelValue: {
18       type: String,
19     },
20   },
21
22   computed: {
23     currentValue: {
24       get() {
25         return this.modelValue
26       },
27       set(newVal) {
28         this.$emit('update:modelValue', newVal)
29       },
30     },
31   },
32 }
33 </script>
34
35 <style lang="css" scoped></style>
```

App.vue src M

You, 5 seconds ago | 1 author (You)

```
1 <template>
2   <custom-input title="User name" v-model="myValue" />
3   <div>User name: {{ myValue }}</div>
4 </template>
5
6 <script>
7   import CustomInput from './components/CustomInput.vue'
8
9 export default {
10   name: 'App',
11
12   components: {
13     CustomInput,
14   },
15
16   data() {
17     return {
18       myValue: null,
19     }
20   },
21 }
22 </script>
23
24 > <style> ... </style>
25
```

CustomInput.vue

```
src > components > CustomInput.vue > {} "CustomInput.vue" > template
1 <template>
2   <label>
3     {{ title }}
4     <input type="text" v-model="currentValue" />
5   </label>
6 </template>
7
8 <script>
9 export default {
10   name: 'CustomInput',
11
12   props: {
13     title: {
14       type: String,
15       default: '',
16     },
17     modelValue: {
18       type: String,
19     },
20   },
21
22   computed: {
23     currentValue: {
24       get() {
25         return this.modelValue
26       },
27       set(newVal) {
28         this.$emit('update:modelValue', newVal)
29       },
30     },
31   },
32 }
33 </script>
34
35 <style lang="css" scoped></style>
```

App.vue

```
src > App.vue > {} "App.vue"
You, 5 seconds ago | 1 author (You)
1 <template>
2   <custom-input title="User name" v-model="myValue" />
3   <div>User name: {{ myValue }}</div>
4 </template>
5
6 <script>
7   import CustomInput from './components/CustomInput.vue'
8
9 export default {
10   name: 'App',
11
12   components: {
13     CustomInput,
14   },
15
16   data() {
17     return {
18       myValue: null,
19     }
20   },
21 }
22 </script>
23
24 > <style> ... </style>
25
26
```

The image shows two code editors side-by-side, illustrating the flow of data and component usage in a Vue.js application.

CustomInput.vue

```
src > components > CustomInput.vue > {} "CustomInput.vue" > template
1 <template>
2   <label>
3     {{ title }}
4     <input type="text" v-model="currentValue" />
5   </label>
6 </template>
7
8 <script>
9 export default {
10   name: 'CustomInput',
11
12   props: {
13     title: {
14       type: String,
15       default: '',
16     },
17     modelValue: {
18       type: String,
19     },
20   },
21
22   computed: {
23     currentValue: {
24       get() {
25         return this.modelValue
26       },
27       set(newVal) {
28         this.$emit('update:modelValue', newVal)
29       },
30     },
31   },
32 }
33 </script>
34
35 <style lang="css" scoped></style>
```

App.vue

```
src > App.vue > {} "App.vue"
You, 5 seconds ago | 1 author (You)
1 <template>
2   <custom-input title="User name" v-model="myValue" />
3   <div>User name: {{ myValue }}</div>
4 </template>
5
6 <script>
7 import CustomInput from './components/CustomInput.vue'
8
9 export default {
10   name: 'App',
11
12   components: {
13     CustomInput,
14   },
15
16   data() {
17     return {
18       myValue: null,
19     }
20   },
21 }
22 </script>
23
24 > <style> ...
25
26
27
28
29
30
31
32
33
34
```

A dashed orange arrow points from the `v-model="currentValue"` attribute in the `CustomInput.vue` template to the `currentValue` computed property in the script. Another dashed orange arrow points from the `newVal` parameter in the `set` method of the `currentValue` computed property to the `$emit` call, which emits the `'update:modelValue'` event back to the parent component.

CustomInput.vue

```
src > components > CustomInput.vue > {} "CustomInput.vue" > template
1 <template>
2   <label>
3     {{ title }}
4     <input type="text" v-model="currentValue" />
5   </label>
6 </template>
7
8 <script>
9 export default {
10   name: 'CustomInput',
11
12   props: {
13     title: {
14       type: String,
15       default: '',
16     },
17     modelValue: {
18       type: String,
19     },
20   },
21
22   computed: {
23     currentValue: {
24       get() {
25         return this.modelValue
26       },
27       set(newVal) {
28         this.$emit('update:modelValue', newVal)
29       },
30     },
31   },
32 }
33 </script>
34
35 <style lang="css" scoped></style>
```

App.vue

```
src > App.vue > {} "App.vue"
1 You, 5 seconds ago | 1 author (You)
2 <template>
3   <custom-input title="User name" v-model="myValue" />
4   <div>User name: {{ myValue }}</div>
5 </template>
6
7 <script>
8 import CustomInput from './components/CustomInput.vue'
9
10 export default {
11   name: 'App',
12
13   components: {
14     CustomInput,
15   },
16
17   data() {
18     return {
19       myValue: null,
20     }
21   }
22 </script>
23
24 > <style> ... </style>
25
26
27
28
29
30
31
32
33
34
```

Приклад. Розробити компонент у який передається selectorTitle та список можливих значень для вибору. Компонент повертає виране значення з використанням v-model. З його використанням організувати ввід марки авто та кузова

Виберіть тип кузова:

Виберіть марку авто:

Виберіть тип палива:

```
const carBrands = [  
  { id: 1, title: "Toyota" },  
  { id: 2, title: "Ford" },  
  { id: 3, title: "Honda" },  
  { id: 4, title: "Chevrolet" },  
  { id: 5, title: "BMW" },  
  { id: 6, title: "Audi" },  
  { id: 7, title: "Mercedes-Benz" }  
];
```

```
const carBodyTypes = [  
  { id: 1, title: "Седан" },  
  { id: 2, title: "Хетчбек" },  
  { id: 3, title: "Купе" },  
  { id: 4, title: "Універсал" },  
  { id: 5, title: "Спорткар" },  
  { id: 6, title: "Кросовер" },  
  { id: 7, title: "Мінівен" }  
];
```

```
const fuelTypes = [  
  { id: 1, title: "Бензин" },  
  { id: 2, title: "Дизель" },  
  { id: 3, title: "Гібрид" },  
  { id: 4, title: "Електрика" },  
  { id: 5, title: "Газ" }  
];
```

Декілька v-model для власних компонентів у Vue 3. v-model з аргументами

Власний компонент	Батьківський компонент
<pre><template> . . . </template> <script> export default { name: "", props: { . . . властивість_1: { type: ... , default: ... }, . . . }, methods: { метод_1 () { . . . this.\$emit('update:властивість_1', дані1) . . . }, . . . } </script> <style> . . . </style></pre>	<pre><template> . . . <дочірній_компонент> v-model:властивість_1 = " модель_даних_1 " /> . . . </template> <script> export default { name: "", data: { . . . модель_даних_1 : початкове_значення ; . . . }, . . . } </script> <style> . . . </style></pre>

Декілька v-model для власних компонентів у Vue 3. v-model з аргументами

Власний компонент	Батьківський компонент
<pre><template> . . . </template> <script> export default { name: "", props: { . . . властивість_1: { type: ... , default: ... }, . . . }, methods: { метод_1 () { . . . this.\$emit('update:властивість_1', дані1) . . . }, . . . } </script> <style> . . . </style></pre>	<pre><template> . . . <дочірній_компонент> v-model:властивість_1 = " модель_даних_1 " /> . . . </template> <script> export default { name: "", data: { . . . модель_даних_1 : початкове_значення }, . . . } </script> <style> . . . </style></pre>

Декілька v-model для власних компонентів у Vue 3. v-model з аргументами

Власний компонент	Батьківський компонент
<pre><template> . . . </template> <script> export default { name: ".", props: { . . . властивість_1: { type: ... , default: ... }, властивість_2: { type: ... , default: ... }, . . . }, methods: { метод_1 () { . . . this.\$emit('update:властивість_1', дані1) . . . }, метод_2 () { . . . this.\$emit('update:властивість_2', дані2) . . . }, . . . } </script> <style> . . . </style></pre>	<pre><template> . . . дочірній компонент v-model:властивість_1 = " модель_даніх_1 " v-model:властивість_2 = " модель_даніх_2 " /> . . . </template> <script> export default { name: "." data: { . . . модель_даніх_1 : почткове_значення ; модель_даніх_2 : почткове_значення ; . . . }, . . . } </script> <style> . . . </style></pre>

▼ LoginForm.vue components U ●

```
src > components > ▼ LoginForm.vue > {} "LoginForm.vue" > ⚡ template > ⚡ label
1  <template>
2    <label>
3      Login
4      <input
5        type="text" :value="login"
6        @input="$emit('update:login', $event.target.value)">
7    />
8  </label>
9
10
11
12
13
14
15  </label>
16 </template>
17
18 <script>
19 export default {
20   name: 'LoginForm',
21
22   props: {
23     login: {
24       type: String,
25       default: '',
26     },
27
28
29   },
30
31   },
32 }
</script>
```

▼ App.vue src M ●

```
src > ▼ App.vue > {} "App.vue" > ⚡ template > ⚡ login-form
...
1  <template>
2    <login-form
3      v-model:login="userLogin">
4    />
5    You, 1 second ago • uncommitted changes
6    <hr />
7    <div>User data: {{ userLogin }} - {{ userPassword }}</div>
8  </template>
9
10 <script>
11 import LoginForm from './components/LoginForm.vue'
12
13 export default {
14   name: 'App',
15
16   components: {
17     LoginForm,
18   },
19
20   data() {
21     return {
22       userLogin: null,
23     }
24   },
25 }
26 </script>
27
28
29 > <style>...
```

The screenshot shows a Vue.js application with two files: `LoginForm.vue` and `App.vue`.

`LoginForm.vue` (Left):

- Template: Contains a label with the text "Login" and an input field. The input field has a value of "Ivan".
- Script: Contains a prop named `login` with a type of `String` and a default value of an empty string. An event listener `@input` emits an update to the `login` prop.

`App.vue` (Right):

- Template: Contains a `login-form` component with a `v-model:login` binding. The value of this binding is `"userLogin"`.
- Script: Imports `LoginForm` and defines the `App` component, which includes `LoginForm` in its components and has a `userLogin` data property set to `null`.

At the bottom, there is a preview of the application showing a login form with the input field containing "Ivan" and a message below it stating "User data: Ivan - 123".

▼ LoginForm.vue components U ●

```
src > components > ▼ LoginForm.vue > {} "LoginForm.vue" > ⚡ template > ⚡ label
1  <template>
2    <label>
3      Login
4      <input
5        type="text" :value="login"
6        @input="$emit('update:login', $event.target.value)">
7    />
8    </label>
9    <label>
10      Password
11      <input
12        type="text" :value="password"
13        @input="$emit('update:password', $event.target.value)">
14    />
15    </label>
16  </template>
17
18 <script>
19 export default {
20   name: 'LoginForm',
21
22   props: {
23     login: {
24       type: String,
25       default: '',
26     },
27     password: {
28       type: String,
29       default: '',
30     },
31   },
32 }
33 </script>
```

▼ App.vue src M ●

```
src > ▼ App.vue > {} "App.vue" > ⚡ template > ⚡ login-form
...
1  <template>
2    <login-form
3      v-model:login="userLogin"
4      v-model:password="userPassword">
5    />
6    You, 1 second ago • Uncommitted changes
7    <hr />
8    <div>User data: {{ userLogin }} - {{ userPassword }}</div>
9  </template>
10 <script>
11 import LoginForm from './components/LoginForm.vue'
12
13 export default {
14   name: 'App',
15
16   components: {
17     LoginForm,
18   },
19
20   data() {
21     return {
22       userLogin: null,
23       userPassword: null,
24     }
25   },
26 }
27 </script>
28
29 > <style>...
```

The screenshot shows a Vue.js application with two files: `LoginForm.vue` and `App.vue`.

`LoginForm.vue` (Left):

- Template:
 - Contains two `<label>` elements with `<input>` children.
 - The first `<input>` has a red box around its `@input` event handler.
 - The second `<input>` has a green box around its `@input` event handler.
- Script:
 - Defines a prop `login` with type `String` and default value `''`.
 - Defines a prop `password` with type `String` and default value `''`.

`App.vue` (Right):

- Template:
 - Contains a `<login-form>` component with `v-model:login="userLogin"` and `v-model:password="userPassword"`.
- Script:
 - Imports `LoginForm` from `./components/LoginForm.vue`.
 - Exports a default component `App` with name `'App'` and components `LoginForm`.
 - Data function returns an object with `userLogin: null` and `userPassword: null`.

Bottom: A screenshot of the application interface showing:

- A login form with fields for `Login` (value: Ivan) and `Password` (value: 123).
- The text `User data: Ivan - 123` displayed below the form.

LoginForm.vue components U X

src > components > ▼ LoginForm.vue > {} "LoginForm.vue" > script > default > computed >

```
1 <template>
2   <label>
3     Login
4     <input type="text" v-model="loginValue" />
5   </label>
6   <label>
7     Password
8     <input type="text" v-model="passwordValue" />
9   </label>
10  </template>
11  <script>
12    export default {
13      name: 'LoginForm',
14      props: {
15        login: { type: String },
16        password: { type: String },
17      },
18      computed: {
19        loginValue: {
20          get() {
21            return this.login
22          },
23          set(newVal) {
24            this.$emit('update:login', newVal)
25          },
26        },
27        passwordValue: {
28          get() {
29            return this.password
30          },
31          set(newVal) {
32            this.$emit('update:password', newVal)
33          },
34        },
35      },
36    }
37  </script>
```

App.vue src M X

You, 47 seconds ago | 1 author (You)

```
1 <template>
2   <login-form v-model:login="userLogin"
3     v-model:password="userPassword" />
4   <hr />
5   <div>User data: {{ userLogin }} - {{ userPassword }}</div>
6 </template>
7 <script>
8   import LoginForm from './components/LoginForm.vue'
9
10  export default {
11    name: 'App',
12    components: {
13      LoginForm,
14    },
15    data() {
16      return {
17        userLogin: null,
18        userPassword: null,
19      }
20    },
21  }
22 </script>
23 <style>...</style>
24
25
26
27
28
29
30
31
32
33
34
35
36
```

З використанням
обчислювальних
властивостей

LoginForm.vue components U X

src > components > ▼ LoginForm.vue > {} "LoginForm.vue" > script > default > computed >

```
1 <template>
2   <label>
3     Login
4     <input type="text" v-model="loginValue" />
5   </label>
6   <label>
7     Password
8     <input type="text" v-model="passwordValue" />
9   </label>
10  </template>
11  <script>
12    export default {
13      name: 'LoginForm',
14      props: {
15        login: { type: String },
16        password: { type: String },
17      },
18      computed: {
19        loginValue: {
20          get() {
21            return this.login
22          },
23          set(newVal) {
24            this.$emit('update:login', newVal)
25          },
26        },
27        passwordValue: {
28          get() {
29            return this.password
30          },
31          set(newVal) {
32            this.$emit('update:password', newVal)
33          },
34        },
35      },
36    }
37  </script>
```

App.vue src M X

You, 47 seconds ago | 1 author (You)

```
1 <template>
2   <login-form v-model:login="userLogin"
3     v-model:password="userPassword" />
4   <hr />
5   <div>User data: {{ userLogin }} - {{ userPassword }}</div>
6 </template>
7 <script>
8   import LoginForm from './components/LoginForm.vue'
9
10  export default {
11    name: 'App',
12    components: {
13      LoginForm,
14    },
15    data() {
16      return {
17        userLogin: null,
18        userPassword: null,
19      }
20    },
21  }
22 </script>
23 <style>...</style>
24
25
26
27
28
29
30
31
32
33
34
35
36
```

З використанням
обчислювальних
властивостей

LoginForm.vue components U X

src > components > ▼ LoginForm.vue > {} "LoginForm.vue" > script > default > computed >

```
1 <template>
2   <label>
3     Login
4     <input type="text" v-model="loginValue" />
5   </label>
6   <label>
7     Password
8     <input type="text" v-model="passwordValue" />
9   </label>
10  </template>
11  <script>
12    export default {
13      name: 'LoginForm',
14      props: {
15        login: { type: String },
16        password: { type: String },
17      },
18      computed: {
19        loginValue: {
20          get() {
21            return this.login
22          },
23          set(newVal) {
24            this.$emit('update:login', newVal)
25          },
26        },
27        passwordValue: {
28          get() {
29            return this.password
30          },
31          set(newVal) {
32            this.$emit('update:password', newVal)
33          },
34        },
35      },
36    }
37  </script>
```

App.vue src M X

You, 47 seconds ago | 1 author (You)

```
1 <template>
2   <login-form v-model:login="userLogin"
3     v-model:password="userPassword" />
4   <hr />
5   <div>User data: {{ userLogin }} - {{ userPassword }}</div>
6 </template>
7 <script>
8   import LoginForm from './components/LoginForm.vue'
9
10  export default {
11    name: 'App',
12    components: {
13      LoginForm,
14    },
15    data() {
16      return {
17        userLogin: null,
18        userPassword: null,
19      }
20    },
21  }
22 </script>
23 <style>...</style>
24
25
26
27
28
29
30
31
32
33
34
35
36
```

З використанням обчислювальних властивостей

LoginForm.vue components U X

src > components > ▼ LoginForm.vue > {} "LoginForm.vue" > ⚡ script > default > 🌐 computed >

```
1 <template>
2   <label>
3     Login
4     <input type="text" v-model="loginValue" />
5   </label>
6   <label>
7     Password
8     <input type="text" v-model="passwordValue" />
9   </label>
10  </template>
11  <script>
12    export default {
13      name: 'LoginForm',
14      props: {
15        login: { type: String },
16        password: { type: String },
17      },
18      computed: {
19        loginValue: {
20          get() {
21            return this.login
22          },
23          set(newVal) {
24            this.$emit('update:login', newVal)
25          },
26        },
27        passwordValue: {
28          get() {
29            return this.password
30          },
31          set(newVal) {
32            this.$emit('update:password', newVal)
33          },
34        },
35      },
36    }
37  </script>
```

App.vue src M X

You, 47 seconds ago | 1 author (You)

```
1 <template>
2   <login-form v-model:login="userLogin"
3     v-model:password="userPassword" />
4   <hr />
5   <div>User data: {{ userLogin }} - {{ userPassword }}</div>
6 </template>
7 <script>
8   import LoginForm from './components/LoginForm.vue'
9
10  export default {
11    name: 'App',
12    components: {
13      LoginForm,
14    },
15    data() {
16      return {
17        userLogin: null,
18        userPassword: null,
19      }
20    },
21  }
22 </script>
23 <style>...</style>
24
25
26
27
28
29
30
31
32
33
34
35
36
```

З використанням обчислювальних властивостей

LoginForm.vue components U X

src > components > ▼ LoginForm.vue > {} "LoginForm.vue" > ⚡ script > default > 🌐 computed >

```
1 <template>
2   <label>
3     Login
4     <input type="text" v-model="loginValue" />
5   </label>
6   <label>
7     Password
8     <input type="text" v-model="passwordValue" />
9   </label>
10  </template>
11  <script>
12    export default {
13      name: 'LoginForm',
14      props: {
15        login: { type: String },
16        password: { type: String },
17      },
18      computed: {
19        loginValue: {
20          get() {
21            return this.login
22          },
23          set(newVal) {
24            this.$emit('update:login', newVal)
25          },
26        },
27        passwordValue: {
28          get() {
29            return this.password
30          },
31          set(newVal) {
32            this.$emit('update:password', newVal)
33          },
34        },
35      },
36    }
37  </script>
```

App.vue src M X

You, 47 seconds ago | 1 author (You)

```
1 <template>
2   <login-form v-model:login="userLogin"
3     v-model:password="userPassword" />
4   <hr />
5   <div>User data: {{ userLogin }} {{ userPassword }}</div>
6  </template>
7  <script>
8    import LoginForm from './components/LoginForm.vue'
9
10   export default {
11     name: 'App',
12
13     components: {
14       LoginForm,
15     },
16
17     data() {
18       return {
19         userLogin: null,
20         userPassword: null,
21       }
22     },
23   }
24  </script>
25
26  > <style>...
36
```

З використанням
обчислювальних
властивостей

Приклад. Компонент фільтр дозволяє вводити параметри фільтрації і повертає їх з використанням v-model

Фільтри : Назва Рік видання Автор

Список книг

Назва книги	Рік видання	Автор
"Тисяча чудес"	2003	Марк Левін
"Шлях до багатства"	1937	Наполеон Хілл
"Гаррі Поттер і філософський камінь"	1997	Джоан Роулінг
"Таємниця старого мосту"	1982	Ерл Стенлі Гарднер
"Маленький принц"	1943	Антуан де Сент-Экзюпери

Використання модифікаторів

Модифікатор	Призначення	Приклад
.lazy	для синхронізації після подій change	<code><input v-model.lazy="msg" /></code>
.number	дані автоматично перетворювалися на числові	<code><input v-model.number="age" /></code>
.trim	видаляються початкові і кінцеві пробіли	<code><input v-model.trim="msg" /></code>

Власний компонент

```
<template>
  . . .
</template>
<script>
export default {
  name: ". . . . .",
  props:{},
  .....
  modelValue:{
    type: ... ,
    default: ...
  },
  modelModifiers: {
    default: () => ({})
  }
  .....
},
methods: {
  метод () {
    .....
    . . . . ВИКОРИСТАННЯ . . .
    this.modelModifiers.назва_модифікатора
    .....
    this.$emit('update:modelValue', дані )
    .....
  }
},
.....
}
</script>
<style> ... </style>
```

Батьківський компонент

```
<template>
  . . .
  <дочірній_компонент
    v-model.модифікатор1.модифікатор2... = " модель_даніх "
  />
  . . .
</template>
<script>
export default {
  name: ". . . . ."
  data: {
    .....
    модель_даніх : початкове_значення ;
    .....
  },
  .....
  .....
}
</script>
<style> ... </style>
```

Власний компонент

```
<template>
    . . .
</template>
<script>
export default {
    name: "...",
    props:{},
    .....
    modelValue:{
        type: ... ,
        default: ...
    },
    modelModifiers: {
        default: () => ({})
    }
    .....
},
methods: {
    метод () {
        .....
        . . . . ВИКОРИСТАННЯ . . .
        this.modelModifiers.назва_модифікатора
        .....
        this.$emit('update:modelValue', дані )
        .....
    }
},
.....
}
</script>
<style> ... </style>
```

Батьківський компонент

```
<template>
    . . .
    <дочірній_компонент
        v-model.модифікатор1.модифікатор2... = "модель_даних">
    />
    . . .
</template>
<script>
export default {
    name: "...",
    data: {
        .....
        модель_даних : початкове_значення ;
        .....
    },
    .....
}
</script>
<style> ... </style>
```

The image shows a screenshot of a Vue.js application in a code editor, illustrating the flow of data and component usage.

CustomInput.vue (Left Panel):

```
src > components > CustomInput.vue > {} "CustomInput.vue" > script > default > props > title > de
1  <template>
2  |   <label>{{ title }}</label>
3  |   <input type="text" v-model="currentValue" />
4  | </template>
5  <script>
6  export default {
7      name: 'CustomInput',
8      props: {
9          title: { type: String, default: '' },
10         modelValue: {
11             type: String,
12         },
13         modelModifiers: {
14             default: () => ({})
15         },
16     },
17 },
18 computed: {
19     currentValue: {
20         get() {
21             return this.modelValue
22         },
23         set(newVal) {
24             if (this.modelModifiers.trimStart) newVal = newVal.trimStart()
25             if (this.modelModifiers.makeFirstUppercase && newVal) {
26                 newVal = `${newVal[0].toUpperCase()}${newVal.slice(1)}`
27             }
28             this.$emit('update:modelValue', newVal)
29         },
30     },
31 },
32 }
33 </script>
34
35 <style lang="css" scoped></style>
```

A tooltip is displayed over the `modelModifiers` prop, showing its type as an object with a `default` key:

```
{ trimStart: true, makeFirstUppercase: true }
```

Two green arrows point from the `trimStart` and `makeFirstUppercase` keys in the tooltip back to their respective uses in the `modelModifiers` object definition in the `CustomInput.vue` script.

App.vue (Right Panel):

```
src > App.vue > {} "App.vue" > script > default
You, 1 second ago | 1 author (You)
1  <template>
2  |   <custom-input
3  |       title="User name"
4  |       v-model.trimStart.makeFirstUppercase="myValue"
5  |   />
6  |   <div>User name: {{ myValue }}</div>
7  </template>
8
9  <script>
10 import CustomInput from './components/CustomInput.vue'
11
12 export default {
13     name: 'App',
14
15     components: {
16         CustomInput,
17     },
18
19     data() {
20         return {
21             myValue: null,
22         }
23     },
24 }
25 </script>
26
27 > <style>...</style>
28
29
30
31
32
33
34
35
36
37
```

The `v-model.trimStart.makeFirstUppercase` binding in the `App.vue` template is highlighted with a red box, and a green arrow points from it to the corresponding prop definition in the `CustomInput.vue` script.

Приклад. Розробити компонент для введення балансу користувача. При введенні кожні 3 розряди потрібно відділяти пробілом

“1234567” => “1 234 567”

Модифікатори для v-model з аргументами

Власний компонент

```
<template> . . . . .</template>
<script>
export default {
  name: ". . . . .",
  props: {
    Властивість_1: {
      type: ... ,
      default: ...
    },
    Властивість_1Modifiers: {
      default: () => ({})
    }
  },
  .....
  methods: {
    метод_1 () {
      .... використовуємо . . .
      this.Властивість_1Modifiers.назва_модифіктора
      .....
      this.$emit('update:властивість_1', дані1)
    },
    .....
  }
}
</script>
<style> . . . </style>
```

Батьківський компонент

```
<template>
  . . .
  <дочірній компонент>
    v-model:властивість_1.модифіктор1_1.модифікатор1_2. ... = " модель_даних_1 "
    . . .
</template>
<script>
export default {
  name: ". . . . .",
  data: {
    .....
    модель_даних_1 : початкове_значення ;
  },
  .....
}
</script>
<style> . . . </style>
```

Додаємо до назви
властивості закінчення
Modifiers

Модифікатори для v-model з аргументами

Власний компонент

```
<template> . . . . .</template>
<script>
export default {
  name: ". . . . .",
  props: {
    властивість_1: {
      type: ... ,
      default: ...
    },
    властивість_1Modifiers: {
      default: () => ({})
    }
  },
  властивість_2: {
    type: ... ,
    default: ...
  },
  властивість_2Modifiers: {
    default: () => ({})
  }
  .....
},
methods: {
  метод_1 () {
    . . . використовуємо . . .
    this.властивість_1Modifiers.назва_модифіктора
    .....
    this.$emit('update:властивість_1', дані1 )
  },
  метод_2 () {
    . . . використовуємо . . .
    this.властивість_2Modifiers.назва_модифіктора
    .....
    this.$emit('update:властивість_2', дані2 )
  }
  .....
}
</script>
<style> . . . </style>
```

Додаємо до назв
властивостей
закінчення **Modifiers**

Батьківський компонент

```
<template>
  . . .
  <дочірній компонент
    v-model:властивість_1.модифіктор1_1.модифікатор1_2. . . . = " модель_даніх_1 "
    v-model:властивість_2.модифіктор1_1.модифікатор1_2. . . . = " модель_даніх_2 "
  />
  . . .
</template>
<script>
export default {
  name: ". . . . .",
  data: {
    .....
    модель_даніх_1 : почткове_значення ;
    модель_даніх_2 : почткове_значення ;
    .....
  },
  .....
}
</script>
<style> . . . </style>
```

The screenshot shows two code editors side-by-side, illustrating the flow of data and component interaction in a Vue.js application.

LoginForm.vue (Left Editor)

```
src > components > ▼ LoginForm.vue > {} "LoginForm.vue" > ⚡ script > ⚡ default > ⚡ computed > ⚡ loginValue
```

```
1 <template>
2   <label> Login <input type="text" v-model="loginValue" /> </label>
3   <label> Password <input type="text" v-model="cardValue" /> </label>
4 </template>
5 <script>
6 export default {
7   name: 'LoginForm',
8   props: {
9     login: { type: String },
10    loginModifiers: { default: () => {} },
11    cardNumber: { type: String },
12    cardNumberModifiers: { default: () => {} },
13  },
14  computed: {
15    loginValue: {
16      get() { return this.login },
17      set(newVal) {
18        if (this.loginModifiers.uppercase) newVal = newVal.toUpperCase()
19        this.$emit('update:login', newVal)
20      },
21    },
22    cardValue: {
23      get() { return this.cardNumber },
24      set(newVal) {
25        if (this.cardNumberModifiers.separate4Digits) {
26          newVal = newVal.replace(/(\d{4})(?=\S+)/g, '$1 ')
27        }
28        this.$emit('update:cardNumber', newVal)
29      },
30    },
31  },
32}
33 </script>
34
35 <style lang="css" scoped></style>
```

A callout box highlights the `uppercase :true` value being passed from the `loginModifiers` prop. A dashed arrow points from this value to the `newVal` variable in the `set` method of the `loginValue` computed property.

App.vue (Right Editor)

```
src > ▼ App.vue > {} "App.vue" > ⚡ template > ⚡ login-form
```

```
1 <template>
2   <login-form
3     v-model:login uppercase="userLogin"
4     v-model:cardNumber.separate4Digits="userCreditCard"
5   />
6   <hr />
7   <div>User data: {{ userLogin }} : {{ userCreditCard }}</div>
8 </template>
9
10 <script>
11 import LoginForm from './components/LoginForm.vue'
12
13 export default {
14   name: 'App',
15
16   components: {
17     LoginForm,
18   },
19
20   data() {
21     return {
22       userLogin: null,
23       userCreditCard: null,
24     }
25   },
26 }
27 </script>
28
29 > <style>...</style>
30
31
```

A callout box highlights the `uppercase="userLogin"` and `separate4Digits="userCreditCard"` values being used in the `v-model` attributes of the `login-form` component. A curved arrow points from the `userLogin` and `userCreditCard` variables in the `data` method back to their respective `v-model` attributes.

The screenshot shows two code editors side-by-side, illustrating the flow of data and component interaction in a Vue.js application.

LoginForm.vue (Left Editor)

```
src > components > ▼ LoginForm.vue > {} "LoginForm.vue" > ⚡ script > ⚡ default > ⚡ computed > ⚡ loginValue
1 <template>
2   <label> Login <input type="text" v-model="loginValue"/> </label>
3   <label> Password <input type="text" v-model="cardValue"/> </label>
4 </template>
5 <script>
6 export default {
7   name: 'LoginForm',
8   props: {
9     login: { type: String },
10    loginModifiers: { default: () => {} },
11    cardNumber: { type: String },
12    cardNumberModifiers: { default: () => {} },
13  },
14  computed: {
15    loginValue: {
16      get() { return this.login },
17      set(newVal) {
18        if (this.loginModifiers.uppercase) newVal = newVal.toUpperCase()
19        this.$emit('update:login', newVal)
20      },
21    },
22    cardValue: {
23      get() { return this.cardNumber },
24      set(newVal) {
25        if (this.cardNumberModifiers.separate4Digits) {
26          newVal = newVal.replace(/(\d{4})(?=\S+)/g, '$1 ')
27        }
28        this.$emit('update:cardNumber', newVal)
29      },
30    },
31  },
32}
33 </script>
34
35 <style lang="css" scoped></style>
```

A callout box highlights the `separate4Digits :true` property being used in the `cardNumberModifiers` object.

App.vue (Right Editor)

```
src > ▼ App.vue > {} "App.vue" > ⚡ template > ⚡ login-form
You, 27 minutes ago | 1 author (You)
1 <template>
2   <login-form
3     v-model:login.uppercase="userLogin"
4     v-model:cardNumber.separate4Digits="userCreditCard"
5   />
6   <br />
7   <div>User data: {{ userLogin }} : {{ userCreditCard }}</div>
8 </template>
9
10 <script>
11 import LoginForm from './components/LoginForm.vue'
12
13 export default {
14   name: 'App',
15
16   components: {
17     LoginForm,
18   },
19
20   data() {
21     return {
22       userLogin: null,
23       userCreditCard: null,
24     }
25   },
26 }
27 </script>
28
29 > <style>...</style>
30
31
```

A callout box highlights the `v-model:cardNumber.separate4Digits="userCreditCard"` binding in the `login-form` component.

Приклад. Розробити компонент для вводу віку користувача та імені. Робити форматування відповідних іпут у випадку потожнього поля (червони фон) і вік (вік менший за 18 – колір фону червоний, інакше – зелений).