

Rendering and Styles



SoftUni Team

Technical Trainers



SoftUni



Software University

<https://about.softuni.bg>

Table of Contents

1. Conditional and list rendering
2. Event handling
3. Computed properties
4. Watchers



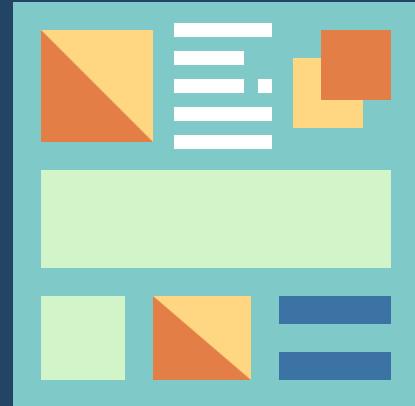
Have a Question?



sli.do

#vue-js

The text is overlaid on a faint, light-gray network diagram consisting of several circular nodes connected by thin lines, suggesting a social or technical network structure.



Rendering Elements

Conditional and list rendering

Conditional Rendering

- **v-if**
- used to conditionally render a block
- will only be rendered if the directive's expression returns a truthy value
- **v-else**
- You can use the **v-else** directive to indicate an "else block" for **v-if**
- Element must **immediately** follow a **v-if**

```
<h1 v-if="isVisible">  
  This will be conditionaly  
  visible! Vue is awesome!  
</h1>
```

```
<button @click="isDay =  
  !isDay">Toggle</button>  
  
<h1 v-if="isDay">☀ It's sunny  
  outside!</h1>  
<h1 v-else>🌙 It's dark outside!  
</h1>
```

Conditional Rendering

■ **v-else-if**

- as the name suggests, serves as an "else if block" for **v-if**
- can also be chained multiple times
- a **v-else-if** element must immediately follow a **v-if** or a **v-else-if**

```
<div v-if="weather === 'cloudy'>
  ☁️ Cloudy
</div>

<div v-else-if="weather ===
  'rainy'>
  ⛈ Rainy
</div>

<div v-else-if="weather ===
  'stormy'>
  ⚡️ Stormy
</div>

<div v-else>
  ☀️ Sunny
</div>
```

Conditional rendering with <template>

■ <template>

- Used as a placeholder when we want to use a built-in directive without rendering an element in the DOM
- The special handling is only triggered if it is used with one of these directives: v-if / v-else-if / v-else / v-slot / v-for
- If none of those directives are present, then it will be rendered as a native <template> element instead

```
<template v-if="showContent">
  <h1>
    🎉 Welcome to My Awesome Page 🎉
  </h1>
  <p>
    🤝 Hello there! This is a cool Vue.js
    example.
  </p>
  <p>
    🚀 Let's explore some awesome features
    together!
  </p>
</template>

<p v-else>
  Click the button above to reveal the
  exciting content! 😊
</p>
```

Conditional Rendering – v-show

■ v-show

- Another option for conditionally displaying an element
- An element with **v-show** will always be rendered and remain in the DOM
- **v-show** only toggles the display CSS property of the element
- **v-show** doesn't support the **<template>** element, nor does it work with **v-else**

```
<h1 v-show="isVisible">I'm visible!</h1>
```

v-if vs. v-show

- **v-if**
 - "real" conditional rendering because it ensures that event listeners and child components inside the conditional block are properly destroyed and re-created during toggles
- **v-show**
 - Is much simpler - the element is always rendered regardless of initial condition, with CSS-based toggling

**Generally speaking, v-if has higher toggle costs,
while v-show has higher initial render costs**

List Rendering

v-for

- render a multiple (list) items
- works with an array, object or a range

```
<li v-for="product in cartProducts">  
  {{ product.name }}  
</li>
```

```
data() {  
  return {  
    cartProducts: [  
      { id: 1, name: 'Smartphone', price:  
499.99, quantity: 2 },  
      { id: 2, name: 'Laptop', price: 999.99,  
quantity: 1 },  
      { id: 3, name: 'Headphones', price:  
79.99, quantity: 3 },  
      { id: 4, name: 'Tablet', price: 299.99,  
quantity: 2 },  
    ]  
  }  
}
```

// We can also get and use the index parameter in a `v-for` as an optional second argument

```
<li v-for="(product, idx) in cartProducts">  
  {{ idx }} - {{ product.name }}  
</li>
```

List Rendering - keys

- **v-for** and **key**
 - give Vue a hint so that it can track each rendered element identity
 - should be unique and "constant" value
- When using **<template v-for>**, the **key** should be placed on the **<template>** container

```
<div v-for="product in cartProducts"
  :key="product.id">
  <!-- content -->
</div>
```

```
<template v-for="product in cartProducts"
  :key="product.id">
  <li>{{ product.id }}</li>
</template>
```

Nested v-for

- For nested **v-for**, scoping also works similar to nested functions. Each **v-for** scope has access to parent scopes

```
<li v-for="category in inventory"
:key="category.id">
  <h3>{{ category.name }}</h3>
  <ul>
    <li v-for="product in category.products"
:key="product.id">
      <span>{{ product.name }}</span>
      <span>Price: ${{ product.price }}</span>
      <span>Quantity: {{ product.quantity }}</span>
    </li>
  </ul>
</li>
```

```
data() {
  return {
    inventory: [
      {
        id: 1,
        name: 'Electronics',
        products: [
          { id: 1, name: 'Smartphone', price: 499.99, quantity: 2 },
          { id: 2, name: 'Laptop', price: 999.99, quantity: 1 },
        ],
      },
      {
        id: 2,
        name: 'Audio',
        products: [
          { id: 3, name: 'Headphones', price: 79.99, quantity: 3 },
        ],
      },
    ];
  },
}
```

v-for with an Object

- With an object, the second optional argument/alias will be the property's name (key), and a third one for the index

```
data() {  
  return {  
    bookInfo: {  
      title: 'The Magical Adventure',  
      author: 'John Smith',  
      genre: 'Fantasy',  
      publishedAt: '2023-07-15',  
      pages: 320,  
      rating: '★★★★★',  
    },  
  };  
},
```

```
<ul>  
  <li v-for="(value, key) in  
  bookInfo" :key="key">  
    <strong>{{ key }}:</strong> {{  
      value }}  
  </li>  
</ul>
```

v-for with a Range

- v-for can also take an integer
- In this case it will repeat the template that many times, based on a range of **1...n**

```
<span v-for="n in 10">{{ n }}</span>
```

v-for with v-if

- It's **not** recommended to use **v-if** and **v-for** on the same element due to implicit precedence
- When **v-if** and **v-for** are both used on the same element, **v-if will be evaluated first**



```
<ul>
  <li
    v-for="user in users"
    v-if="user.isActive"
    :key="user.id"
  >
    {{ user.name }}
  </li>
</ul>
```



```
<ul>
  <template v-for="user in users" :key="user.id">
    <li v-if="user.isActive">
      {{ user.name }}
    </li>
  </template>
</ul>
```





Event Handling

Listening, calling and modifying events

Listening to Events - Inline handlers

- **v-on** directive
 - listen to DOM events and run some JavaScript when they're triggered
 - we typically shorten to the **@** symbol
 - usage would be **v-on:click="handler"** or with the shortcut, **@click="handler"**

```
data() {  
  return {  
    count: 0  
  }  
}
```

```
<button @click="count++">Add 1</button>  
<p>Count is: {{ count }}</p>
```

Listening to Events - Method handlers

- The logic for many event handlers will be more complex
- "Cleaner" and easier to debug
- A method handler automatically receives the native DOM Event object that triggers it

```
data() {  
  return {  
    name: 'Vue.js'  
  }  
},  
methods: {  
  greet() {  
    // `this` inside methods points to the  
    current active instance  
    alert(`Hello ${this.name}!`)  
  }  
}
```

<!-- `greet` is the name of the method defined above -->

```
<button @click="greet">Greet</button>
```

What to listen to?

- **v-on/@** directive can be used for any element's event
- Arguably the most common ones are
 - @click
 - @change
 - @input
- Read more
 - [HTML DOM Events](#)
 - [Event reference](#)



Calling Methods in Inline Handlers

- Instead of binding directly to a method name, we can also call methods in an inline handler. This allows us to pass the method custom arguments.

```
methods: {  
  say(message) {  
    alert(message)  
  }  
}
```

```
<button @click="say('hello')">Say hello</button>  
<button @click="say('bye ')">Say bye</button>
```

Accessing Event Argument - \$event

- Sometimes we need to have access and pass the event argument, but also pass a custom argument. This is possible with **\$event**

```
<!-- using $event special variable -->
<button @click="warn('Form cannot be submitted yet.', $event)">
  Submit
</button>

<!-- using inline arrow function -->
<button @click="(event) => warn('Form cannot be submitted yet.',
  event)">
  Submit
</button>
```

```
warn(message, event) {
  console.warn(message, event.target.tagName)
}
```

Event Modifiers

- **.stop**
 - stop the propagation of an event through the DOM tree (`stopPropagation()`)

```
<!-- the click event's propagation will be stopped -->
<a @click.stop="doThis"></a>
```
- **.prevent**
 - method is used to prevent the default behavior of an event (`preventDefault()`)

```
<!-- the submit event will no longer reload the page -->
<form @submit.prevent="onSubmit"></form>
```
- **.self / .capture / .once / .passive**
 - Find the rest of the modifiers in the Documentation page [Event Modifiers](#)

Key Modifiers

- When listening for keyboard events, we often need to check for specific keys
- Vue allows adding key modifiers when listening for key events
- See all key modifiers [Key Modifiers](#)
- **.exact** Modifier
 - allows control of the exact combination of system modifiers needed to trigger an event

```
<!-- only call `submit` when the `key` is  
`Enter` -->  
<input @keyup.enter="submit" />
```

```
<!-- this will fire even if Alt or Shift is also  
pressed -->  
<button @click.ctrl="onClick">A</button>  
  
<!-- this will only fire when Ctrl and no other  
keys are pressed -->  
<button  
@click.ctrl.exact="onCtrlClick">A</button>  
  
<!-- this will only fire when no system  
modifiers are pressed -->  
<button @click.exact="onClick">A</button>
```



Reactivity

Computed and Watchers

Computed properties

- **computed:{}** properties allow us to declaratively **compute derived values**
 - Keep template cleaner
 - Cache computation
 - Reactive - If any of the data properties they depend on change, the computed property will automatically update

```
<p>Has published books:</p>
<span>{{ author.books.length > 0 ? 'Yes' : 'No' }}</span>

// Or use the computed
<span>{{ publishedBooksMessage }}</span>
```

```
export default {
  data() {
    return {
      author: {
        name: 'John Doe',
        books: [
          'Vue 2 - Advanced Guide',
          'Vue 3 - Basic Guide',
          'Vue 4 - The Mystery'
        ]
      }
    },
    computed: {
      publishedBooksMessage() {
        return this.author.books.length > 0
? 'Yes' : 'No'
      }
    }
}
```

Example - Computed properties

```
data() {  
  return {  
    cartProducts: [  
      { id: 1, name: 'Smartphone', price:  
499.99, quantity: 2 },  
      { id: 2, name: 'Laptop', price: 999.99,  
quantity: 1 },  
      { id: 3, name: 'Headphones', price:  
79.99, quantity: 3 },  
      { id: 4, name: 'Tablet', price: 299.99,  
quantity: 2 },  
    ],  
  };  
},
```

```
computed: {  
  totalCartValue() {  
    return  
this.cartProducts.reduce((total, product)  
=> {  
      return total + (product.price *  
product.quantity);  
    }, 0);  
  },  
  totalProducts() {  
    return  
this.cartProducts.reduce((total, product)  
=> {  
      return total + product.quantity;  
    }, 0);  
  },  
}
```

Watchers

- **watch{}** property
 - When we need to perform "side effects" in reaction to state changes
 - Separation of Concerns
 - separate logic for reacting to data changes from the rest of your code

```
data() {
  return {
    counter: 0
  };
},
methods: {
  incrementCounter() {
    this.counter++;
  }
},
watch: {
  counter(newValue, oldValue) {
    console.log(`Counter changed from ${oldValue} to ${newValue}`);
  }
}
```

Deep Watchers

- **watch{}** is shallow by default
- Will only trigger when the watched property has been assigned a new value - it won't trigger on nested property changes
- For tracking nested mutations enable the **deep** argument

```
export default {
  watch: {
    stateVariable : {
      handler(newValue, oldValue) {
        // Note: `newValue` will be equal
        // to `oldValue` here
        // on nested mutations as long as
        // the object itself
        // hasn't been replaced.
      },
      deep: true
    }
  }
}
```

Eager Watchers

- `watch{}` is **lazy** by default
- Won't be triggered until the watched source has changed
- In some cases, we may want the same callback logic to be run **eagerly** (run on creation)
- Enable with the **immediate** argument

```
export default {  
  // ...  
  watch: {  
    stateVariable: {  
      handler(newQuestion) {  
        // this will be run immediately on  
        // component creation.  
      },  
      // force eager callback execution  
      immediate: true  
    }  
  }  
  // ...  
}
```

Exercise – Timer App

■ Create a simple Timer App

- An input to accept text in h:m:s – "00:01:30"
- Show the selected time and update the remaining time in the UI
- Create Start/Pause/Reset buttons
- Use a watcher to indicate to the user that 20% of the time is left
- Think about how you can use computed() property





Practice

Live Exercise in Class (Lab)

Summary

- **v-if** and **v-for** directives for better control when rendering components and elements
- **v-on/@** to listen and handle events
- Computed properties help us write cleaner and maintainable code
- Watchers to react to changes in our state



Questions?



SoftUni



Software
University



SoftUni
Creative



SoftUni
Digital



SoftUni
Foundation



SoftUni
Kids



Finance
Academy

SoftUni Diamond Partners



**SUPER
HOSTING
.BG**

INDEAVR
Serving the high achievers

 **SOFTWARE
GROUP**

 **PHAR
VISION**



**Coca-Cola HBC
Bulgaria**

 **AMBITIONED**



BOSCH

 **SmartIT**

 **DXC
TECHNOLOGY**

 **Flutter
International**™

 **DRAFT
KINGS**

 **Postbank**
Решения за твоето утре

 **createX**

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg/>
- © Software University – <https://softuni.bg>



Trainings @ Software University (SoftUni)



- Software University – High-Quality Education, Profession and Job for Software Developers
 - softuni.bg, about.softuni.bg
- Software University Foundation
 - softuni.foundation
- Software University @ Facebook
 - facebook.com/SoftwareUniversity
- Software University Forums
 - forum.softuni.bg



Software
University

