Template Syntax

In Vue.js, template syntax is the special way you write HTML, so Vue can connect your data and logic to the UI.

It’s basically HTML + Vue’s magic bindings.

Virtual DOM

The Virtual DOM (VDOM) is a lightweight copy of the real DOM stored in memory. It allows JavaScript frameworks like Vue, React, etc., to efficiently update the UI without touching the actual DOM directly every time.

The real DOM is slow when it comes to frequent changes (like user input, animations, live data updates). Updating the actual DOM element-by-element can be expensive.

So instead of directly making changes:

- Vue updates the Virtual DOM first
- Then, Vue compares the new Virtual DOM with the previous one (called diffing)
- It calculates the minimum number of changes needed
- And finally patches only those specific parts in the real DOM

How Virtual DOM Works :

i. Initial Render : Vue creates a virtual DOM tree based on the component template and mounts it to the real DOM.

ii. Data Changes : A user clicks a button, submits a form, or receives new data.

iii. Re-render Virtual DOM : Vue creates a new virtual DOM tree based on the updated data.

iV. Diffing : Vue compares the old VDOM with the new VDOM to find the differences.

V. Patching : Vue updates only the changed parts in the real DOM.

Types of Template Syntax in Vue

Template Syntax grouped by their purpose

I. Text Interpolation

Text Interpolation in Vue is a way to display dynamic data inside the HTML using double curly braces {{ }}.

```
{{ variableName }}
```

Example

```
```

```
<template>
  <div>
    <h1>Hello, {{ name }}!</h1>
    <p>You have {{ messages }} new messages.</p>
  </div>
</template>

<script setup>
import { ref } from 'vue'

const name = ref('Sourabha')
const messages = ref(3)
</script>
```

`const name = ref('Sourabha')` // creates a reactive variable.
Inside the template, Vue automatically unwraps the ref when you use `{{ name }}` or `{{ messages }}`

II. Raw HTML

v-html is a Vue directive that lets you insert raw HTML content directly into the DOM. It renders Tags also.

```
<div v-html="htmlContent"></div>
```

Example

```
<template>
  <div>
    <h2>Raw HTML Output:</h2>
    <div v-html="htmlMessage"></div>
  </div>
</template>

<script setup>
import { ref } from 'vue'

const htmlMessage = ref('<strong style="color: green;">Hello, Vue!</strong>')
</script>
```

III. Directives

Directives are special tokens in the template marked with a v- prefix.They tell Vue to do something to the DOM element.

These are like instructions to the DOM

- Bind this attribute
- Show/hide this element

- Repeat this block
- Listen to this event

Categories of Core Vue Directives

- v-bind – Bind attributes dynamically
- v-model – Two-way data binding
- v-if, v-else-if, v-else – Conditional rendering
- v-show – Toggle visibility via CSS
- v-for – Looping / list rendering
- v-on – Event handling
- v-html – Render raw HTML
- v-text – Insert plain text
- v-slot – Named and scoped slots (advanced)

a. v-bind

v-bind allows you to bind dynamic values to HTML attributes (like src, href, class, style, id, etc.) using Vue's reactive data Instead of hardcoding attribute values, you bind them to variables that can change over time.

```
v-bind:attributeName="expression"
```

Shorthand

```
:attributeName="expression"
```

Example :- Binding src

:src binds the imageUrl to the src of .

:alt binds description to the alt attribute.

When either value changes, the DOM updates automatically.

```
<template>
  <img :src="imageUrl" :alt="description" width="200" />
</template>

<script setup>
const imageUrl = 'https://vuejs.org/images/logo.png'
const description = 'Vue Logo'
</script>
```

Example :- Binding class

If isActive becomes false, the class is removed from the DOM.

```
<template>
  <button :class="{ active: isActive }">Toggle</button>
</template>

<script setup>
const isActive = true
</script>
```

Example :- Binding style

If isActive becomes false, the class is removed from the DOM.

```
<template>
  <p :style="styleObject">Styled Text</p>
</template>

<script setup>
const styleObject = {
  color: 'red',
  fontSize: '18px'
}
</script>
```

Example : Display item using v-bind

```
<template>
  <div class="product-card">
    <img :src="product.image" :alt="product.name" />
    <h2>{{ product.name }}</h2>
    <p>Price: ₹{{ product.price }}</p>
    <a :href="productLink" target="_blank">Buy Now</a>
  </div>
</template>

<script setup>
const product = {
  name: 'Bluetooth Speaker',
  price: 1999,
  image: 'https://example.com/speaker.jpg'
}
const productLink = 'https://example.com/product/123'
</script>
```

b. v-model

v-model is a Vue directive that creates two-way data binding between form inputs and your component data.

two-way = Update both ways From data to UI and From UI to data

- When the data changes → UI updates
- When the user changes input → data updates

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

is equivalent to

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

Example : with `<input>`

```
<template>
  <input v-model="name" placeholder="Enter your name" />
  <p>Hello, {{ name }}</p>
</template>

<script setup>
import { ref } from 'vue'

const name = ref('')
</script>
```

Example : with `<textarea>`

```
<template>
  <textarea v-model="bio" placeholder="Write something..."></textarea>
  <p>You wrote: {{ bio }}</p>
</template>

<script setup>
import { ref } from 'vue'

const bio = ref('')
</script>
```

Example : with single Checkbox

```
<template>
  <input type="checkbox" v-model="isChecked" />
  <label>Agree</label>
  <p>Status: {{ isChecked ? 'Agreed' : 'Not Agreed' }}</p>
</template>

<script setup>
```

```
import { ref } from 'vue'

const isChecked = ref(false)
</script>
```

Example : with Multiple Checkbox

```
<template>
  <label><input type="checkbox" value="Vue" v-model="selected" /> Vue</label>
  <label><input type="checkbox" value="React" v-model="selected" /> React</label>
  <label><input type="checkbox" value="Angular" v-model="selected" />
Angular</label>

  <p>Selected: {{ selected }}</p>
</template>

<script setup>
import { ref } from 'vue'

const selected = ref([])
</script>
```

Example : with Radio Buttons

```
<template>
  <label><input type="radio" value="Male" v-model="gender" /> Male</label>
  <label><input type="radio" value="Female" v-model="gender" /> Female</label>

  <p>Selected: {{ gender }}</p>
</template>

<script setup>
import { ref } from 'vue'

const gender = ref('')
</script>
```

Example : with Select Dropdown

```
<template>
  <select v-model="country">
    <option disabled value="">Choose a country</option>
    <option>India</option>
    <option>USA</option>
    <option>Canada</option>
  </select>

  <p>Selected country: {{ country }}</p>
</template>

<script setup>
import { ref } from 'vue'

const country = ref('')
</script>
```

c. v-if, v-else-if, v-else

Show something only if a condition is true. Vue provides these directives to control what shows up in the DOM.

Example : Shows a message based on a user's exam score.

```
<template>
  <input type="number" v-model="score" placeholder="Enter your score" />

  <div v-if="score >= 90">Excellent!</div>
  <div v-else-if="score >= 75">Good Job!</div>
  <div v-else-if="score >= 50">Average</div>
  <div v-else> Needs Improvement</div>
</template>

<script setup>
import { ref } from 'vue'

const score = ref(0)
</script>
```

d. v-show

v-show is a directive in Vue that toggles an element’s visibility using CSS. It does not remove the element from the DOM, it just uses display: none to hide it.

Example : toggle a message

```
<template>
  <button @click="showMessage = !showMessage">
    Toggle Message
  </button>

  <p v-show="showMessage">Hello Joshi, I am visible!</p>
```

```
</template>

<script setup>
import { ref } from 'vue'

const showMessage = ref(true)
</script>
```

NOTE :

v-if builds or destroys, v-show hides or reveals

v-show is ideal when:

- You need fast toggle (e.g., dropdowns, tabs)
- You want to preserve component state or DOM

v-if is ideal when:

- You want to avoid rendering until needed
- You don't want it in the DOM at all when hidden

e. v-for

v-for is used to render a list of items by looping over an array, object, or even a range. It's just like a for...of loop in JavaScript, but in the template!

Basic syntax is

```
<div v-for="item in items" :key="item.id">
  {{ item }}
</div>
```

- items: an array (or object/range) you're looping over
- item: current element in the loop
- `:key` : required when rendering lists — helps Vue track items efficiently

Example : List of Users with Their Names and Roles

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

<script setup>
const users = [
  { id: 1, name: "Sourabha", role: "Admin" },
  { id: 2, name: "Ramesh", role: "User" },
  { id: 3, name: "Joshi", role: "Manager" },
]
</script>
```

Example : Render a Table with Indexes

```
<template>
  <table border="1">
    <tr>
      <th>#</th>
      <th>Name</th>
    </tr>
    <tr v-for="(name, index) in names" :key="index">
      <td>{{ index + 1 }}</td>
      <td>{{ name }}</td>
    </tr>
  </table>
</template>

<script setup>
const names = ['Lakshmi', 'Joshi', 'Sourabha']
</script>
```

Example : Loop over objects

```
<template>
  <div v-for="(value, key, index) in student" :key="key">
    {{ index + 1 }}. {{ key }}: {{ value }}
  </div>
</template>

<script setup>
const student = {
  name: 'Sourabha',
  age: 25,
  grade: 'A',
}
</script>
```

Example : Nested v-for – Looping Inside Loop

```
<template>
  <div v-for="(category, index) in categories" :key="index">
```

```
    <h3>{{ category.name }}</h3>
    <ul>
      <li v-for="item in category.items" :key="item.id">
        {{ item.label }}
      </li>
    </ul>
  </div>
</template>

<script setup>
const categories = [
  {
    name: "Fruits",
    items: [
      { id: 1, label: "Apple" },
      { id: 2, label: "Banana" },
    ],
  },
  {
    name: "Vegetables",
    items: [
      { id: 3, label: "Carrot" },
      { id: 4, label: "Broccoli" },
    ],
  },
]
</script>
```

f. v-on

v-on is used to listen to and handle DOM events (like click, input, submit, etc.). It connects user actions (e.g., clicking a button) with your methods or inline JavaScript.

syntax of v-on

```
<!-- Full syntax -->
<button v-on:click="handleClick">Click me</button>

<!-- Shorthand -->
<button @click="handleClick">Click me</button>
```

Example : counter App

```
<template>
  <button @click="count++">Clicked {{ count }} times</button>
</template>

<script setup>
```

```
import { ref } from 'vue'
const count = ref(0)
</script>
```

Example : event object

```
<template>
  <input @input="handleInput($event)">
</template>

<script setup>
function handleInput(e) {
  console.log("User typed:", e.target.value)
}
</script>
```

Example : pass arguments

```
<template>
  <button @click="greet('Sourabha')">Greet</button>
</template>

<script setup>
function greet(name) {
  alert(`Hello, ${name}!`)
}
</script>
```

g. v-html

v-html is a Vue directive used to inject and render raw HTML content into the DOM.

Example : items list

```
<template>
  <div v-for="(card, index) in cards" :key="index" class="card" v-
html="card.content" />
</template>

<script setup>
const cards = [
  {
    content: `
      <h2 style="color: teal;">Vue Tips</h2>
      <p>Use <code>v-model</code> for two-way data binding.</p>
    `
  },
  {
    content: `
      <h2 style="color: purple;">Security Alert</h2>
      <p style="color: red;"><strong>Do not use v-html</strong> with unsanitized
user input!</p>
    `
  },
]
</script>

<style scoped>
.card {
  border: 1px solid #ccc;
  padding: 16px;
  margin-bottom: 12px;
  border-radius: 8px;
  background-color: #f9f9f9;
}
</style>
```

h. v-text

v-text is used to insert plain text into an HTML element. Which similar to using `{{ }}` syntax.

```
<p v-text="message"></p>
```

equivalent to

```
<p>{{ message }}</p>
```

Example : Display greetings

```
<template>
  <div>
    <p v-text="greeting"></p>
  </div>
</template>

<script setup>
const greeting = "Hello from Vue!"
</script>
```

i. v-slot

v-slot is a directive in Vue that lets a parent component inject content into a child component’s slot. Which help us Reusable UI components (cards, modals, tables), Custom layouts and Passing data (scope) from child to parent

There are three types : Default Slot, Named Slot and Scoped Slot

a. Default Slot : Only one unnamed slot

child component : BaseCard.vue

```
<template>
  <div class="card">
    <slot></slot>
  </div>
</template>
```

parent component

```
<BaseCard>
  <p>This is card content</p>
</BaseCard>
```

b. Named Slot : Multiple slots identified by name

child component : BaseCard.vue

```
<template>
  <header><slot name="header"></slot></header>
  <main><slot></slot></main>
  <footer><slot name="footer"></slot></footer>
</template>
```

parent component

```
<BaseLayout>
  <template #header>
    <h1>Welcome</h1>
  </template>

  <p>Main content goes here.</p>

  <template #footer>
    <p>&copy; 2025</p>
  </template>
</BaseLayout>
```

c. Scoped Slot : Child passes data to slot via props; parent can use it

child component : UserList.vue

```
<template>
  <div>
    <slot v-for="user in users" :user="user" :key="user.id"></slot>
  </div>
</template>

<script setup>
const users = [
  { id: 1, name: 'Sourabha' },
  { id: 2, name: 'Ram' }
]
</script>
```

parent component

```
<UserList>
  <template #default="{ user }">
    <p>User: {{ user.name }}</p>
  </template>
</UserList>
```

Exercises

1. Create a text input box using v-model and display the entered value below it live.
2. Make a form where users can write a message in a textarea. Below the textarea, show how many characters they’ve typed.
3. Add a checkbox using v-model. When it’s checked, display "Subscribed!" otherwise show "Not Subscribed".
4. Allow users to select multiple hobbies (e.g., Reading, Gaming, Music) using checkboxes. Show the selected hobbies as a list.
5. Create a small form: Name (input), Age (number input), Gender (radio), Skills (checkboxes). On submit, show the entire form data below in a formatted way.
6. Create a password input field. Use v-model to check the length of the entered password and display: "Weak" for < 5 characters, "Medium" for 5–8 characters and "Strong" for > 8 characters
7. Use .trim, .number, and .lazy modifiers with v-model. Explain and show how each one behaves differently with inputs.
8. Shows content based on user login status(Loggedin/Loggedout)?
9. Handles different UI states when fetching data (loading/Error/Success)?
10. Shows invoice payment status (Oaid/Pending/Overdue)?
11. Dynamic text based on the weather (sunny/rainy/cloudy) ?
12. Build a tab component where only one tab’s content is visible at a time using v-show