# Event Handling in Vue

Event Handling is the way we listen for user actions like clicking a button, typing in a form, or hovering over an element, and then respond to them with some logic (e.g., method execution, updating state).

In Vue, we use the v-on directive to handle events. It's often shortened as @.

Syntax

```
<!-- Long form -->
<button v-on:click="handleClick">Click Me</button>

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

Example : simple counter app

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

// Define reactive state
const count = ref(0)

// Define the event handler function
const increment = () => {
  count.value++
}
</script>

<template>
  <p>Count: {{ count }}</p>
  <button @click="increment">Increment</button>
</template>
```

Example : Todo List with Input Validation and Keyboard Shortcuts

```
<script setup>
import { ref, computed } from 'vue'

// Reactive state for tasks and input
const tasks = ref([])
const newTask = ref('')
const error = ref('')

// Add task handler
const addTask = () => {
```

```
  if (!newTask.value.trim()) {
    error.value = 'Task cannot be empty'
    return
  }

  tasks.value.push({
    id: Date.now(),
    title: newTask.value,
    completed: false
  })

  newTask.value = ''
  error.value = ''
}

// Delete a task
const deleteTask = (id) => {
  tasks.value = tasks.value.filter(task => task.id !== id)
}

// Toggle completed state
const toggleDone = (task) => {
  task.completed = !task.completed
}

// Computed: Count completed tasks
const completedCount = computed(() => {
  return tasks.value.filter(task => task.completed).length
})

// Clear all tasks (with double click)
const clearAll = () => {
  if (confirm('Are you sure you want to clear all tasks?')) {
    tasks.value = []
  }
}
</script>

<template>
  <h2>Task Manager</h2>

  <!-- Input Field with Enter Key Support -->
  <input
    v-model="newTask"
    @keyup.enter="addTask"
    placeholder="Enter a new task"
  />
  <button @click="addTask">Add Task</button>

  <!-- Error message if input is empty -->
  <p v-if="error" style="color: red;">{{ error }}</p>

  <!-- Task List -->
  <ul>
```

```
    <li v-for="task in tasks" :key="task.id">
      <input type="checkbox" :checked="task.completed" @change="toggleDone(task)"
/>
      <span :style="{ textDecoration: task.completed ? 'line-through' : 'none' }">
        {{ task.title }}
      </span>
      <button @click="deleteTask(task.id)">Delete</button>
    </li>
  </ul>

  <!-- Completed count -->
  <p>Completed: {{ completedCount }} / {{ tasks.length }}</p>

  <!-- Double-click to clear all -->
  <button @dblclick="clearAll">Double-click to Clear All</button>
</template>

<style scoped>
input {
  padding: 5px;
  margin-right: 10px;
}
button {
  margin-left: 10px;
}
li {
  margin: 5px 0;
}
</style>
```

**Types of event handlers in Vue**

In Vue 3, event handlers come in several types based on how you attach them, how you use parameters, modifiers, and the lifecycle they're involved in.

1. Inline Event Handlers

You directly write the logic in the template using a small expression.

When Logic is very short (like toggling a value, incrementing a counter) and no need to reuse the function than inline event handler is used.

```
<template>
  <button @click="count++">Click Me</button>
</template>

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

2. Method-based Event Handler

The logic is placed inside a function in `<script setup>` and referenced in the template.

When the logic is more than a line. You will reuse the logic or want cleaner separation of concerns. You want better readability and testability method based event handler usful.

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

const count = ref(0)

function increment() {
  count.value++
}
</script>

<template>
  <button @click="increment">Increment</button>
</template>
```

3. Event Handler with Parameters

Pass arguments to event handlers using an arrow function.

When need to pass specific data (e.g., an item ID, index, or label) to your handler even handler with parameter is useful.

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

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

4. Accessing the Event Object

Vue automatically passes the native event object to the handler. You can access it via $event or directly as a parameter.

When you need DOM-level details like event.target, event.clientX, event.key, etc. Example: Track mouse position, get input field value directly from the event.

```
<script setup>
const showCoords = (event) => {
```

```
  alert(`X: ${event.clientX}, Y: ${event.clientY}`)
}
</script>

<template>
  <button @click="showCoords">Click to show coordinates</button>
</template>
```