FPT POLYTECHNIC

# XÂY DỰNG GIAO DIỆN TƯƠNG TÁC BACKEND

## BÀI 7:

## EVENT HANDLING VÀ FORM BINDING TRONG VUEJS

# MỤC TIÊU

- Kết thúc bài học này bạn có khả năng
  - Hiểu về Listen to Events (Lắng nghe sự kiện)
  - Tìm hiểu Trình xử lý phương thức
  - Các sự kiện về chuột, bàn phím…
  - Nắm được cách thức liên kết dữ liệu trong Form binding
  - Sử dụng v-model kết hợp các thuộc tính, giá trị
  - Sử dụng v-model với component tùy chỉnh

# NỘI DUNG

- Phần I: Event Handling
  - Listen to Events (Lắng nghe sự kiện)
  - Method Handlers (Trình xử lý phương thức)
  - Event modifiers, Key modifiers, Mouse Button Modifiers
- Phần II: Form Binding
  - Binding đơn giản
  - Ràng buộc giá trị
  - v-model với component

FPT POLYTECHNIC

# BÀI 7:
# EVENT HANDLING VÀ FORM BINDING TRONG VUEJS

## PHẦN I: EVENT HANGLING

www.poly.edu.vn

## **Event Handling** (Xử lý sự kiện)

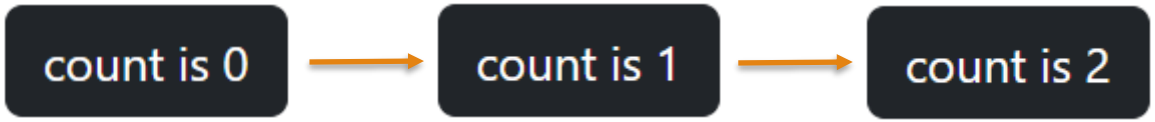
## **Listen to Events** (Lắng nghe sự kiện)

- Để lắng nghe các sự kiện DOM, chúng ta có thể dùng directive **v-on** (hoặc viết tắt là **@**) và thực thi JavaScript khi những sự kiện này được kích hoạt.

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

const count = ref(0)
</script>

<template>
  <div>
    <button class="btn btn-dark" @click="count++">count is {{ count }}</button>
  </div>
</template>
```



# CƠ CHẾ PHÁT RA SỰ KIỆN

- **Emit**: là một cơ chế dùng để phát ra các sự kiện từ một component con để báo cho component cha biết rằng một hành động đã xảy ra (như thay đổi dữ liệu hoặc gửi thông tin ngược lên). Đây là cách mà component con và component cha có thể giao tiếp với nhau.
- Cách dùng: Sử dụng **emit** với Composition API và script setup

Với script setup, chúng ta dùng **defineEmits** (một hàm trong Composition API) để khai báo các sự kiện mà component con có thể phát. Cú pháp:

```
<script setup>
const emit = defineEmits(['my-event'])
</script>
```

- **defineEmits** ở trên đang định nghĩa sự kiện **my-event** mà component con phát ra để báo cho component cha.

# CƠ CHẾ PHÁT RA SỰ KIỆN

Ví dụ về cơ chế giao tiếp giữa component cha và component con:

**Component con (Phát ra sự kiện)**

components > ChildComponent.vue > ...

```vue
<template>
    <button @click="sendEvent">Click me</button>
</template>

<script setup>
const emit = defineEmits(['my-event']);

const sendEvent = () => {
    emit('my-event', 'dữ liệu từ phía component con')
}
</script>
```

**Component Cha (Lắng nghe sự kiện)**

components > ParentComponent.vue > ...

```vue
<template>
    <ChildComponent @my-event="handleEvent" />
</template>

<script setup>
import ChildComponent from './ChildComponent.vue';

const handleEvent = (data) => {
    alert(data);
}
</script>
```

## **Method Handlers** (Trình xử lý phương thức)

- Thông thường ở các bài toán thực tế thì code xử lý sự kiện sẽ không đơn giản là một dòng, mà nó sẽ rất phức tạp. Chính vì thế nên Vue.js cũng hỗ trợ chúng ta tách phần code xử lý vào hàm và gọi hàm đó ở trên sự kiện như javascript thuần.
- Ví dụ: Chuyển đoạn code xử lý sự kiện ở trên vào hàm

# METHOD HANDLERS

**Thực hiện ví dụ**: Phương thức addNumber sẽ xử lý tăng Số lượng khi click vào nút Thêm

```
<template>
  <div>
    <button class="btn btn-dark m-5" @click="addNumber">Thêm</button>
    <p>Số lượng: <span class="text-danger">{{ count }}</span></p>
  </div>
</template>

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

const count = ref(0)

const addNumber = () => {
  count.value += 1;
}
</script>
```

## Truyền Tham Số trong Event Handlers

Có thể truyền tham số vào event handler bằng cách sử dụng cú pháp inline:

```
<script setup>
   const handleClick = (message) => {
   alert(message);
};
</script>

<template>
  <div>
    <button @click="() => handleClick('Hello')">Click me</button>
  </div>
</template>
```


Tái hiện ví dụ ở demo trên

## Event modifiers

- Trong javascript thuần chúng ta thường sử dụng các event modifiles để tác động đến cách xử lý của sự kiện như: preventDefault(), stopPropagation(),...

```
formHandler(event) {
    event.preventDefault();
    // form handling logic
}
```

- Những cách trên vẫn có thể sử dụng được đối với Vue.js, nhưng ngoài cách đó thì Vue.js còn hỗ trợ chúng ta khai báo ở directive bằng cách thêm chúng vào đằng sau directive và ngăn cách giữa chúng bằng dấu .

```
<form @submit.prevent="formHandler"></form>
```

## Event modifiers

- Vue cung cấp một số modifier sự kiện khác nhau hữu ích trong các tình huống xử lý sự kiện phổ biến:

| Modifier | Mô tả |
|---|---|
| **.stop** | Ngăn chặn sự kiện không tiếp tục truyền lên các phần tử cha trong DOM |
| **.prevent** | Gọi event.preventDefault(), sử dụng để ngăn chặn hành vi mặc định của sự kiện |
| **.capture** | Lắng nghe sự kiện ở giai đoạn capturing |
| **.self** | Chỉ kích hoạt sự kiện với chính phần tử đang gán sự kiện đó |
| **.once** | Chỉ kích hoạt sự kiện một lần |

## Key modifiers

- Vue.js hỗ trợ khai báo sự kiện nhấn phím bằng cách thêm mã code của phím đó vào sau directive **v-on:keyup** và ngăn cách giữa chúng bởi dấu **.**
- Cấu trúc sử dụng:

```
v-on:*event*.*keyCode*="method" //eg: v-on:keyup.13="submit"
v-on:*event*.*alias*="method" //eg: v-on:keyup.enter="submit"
```

- *Note: Phím Enter có mã là 13*

## Key modifiers

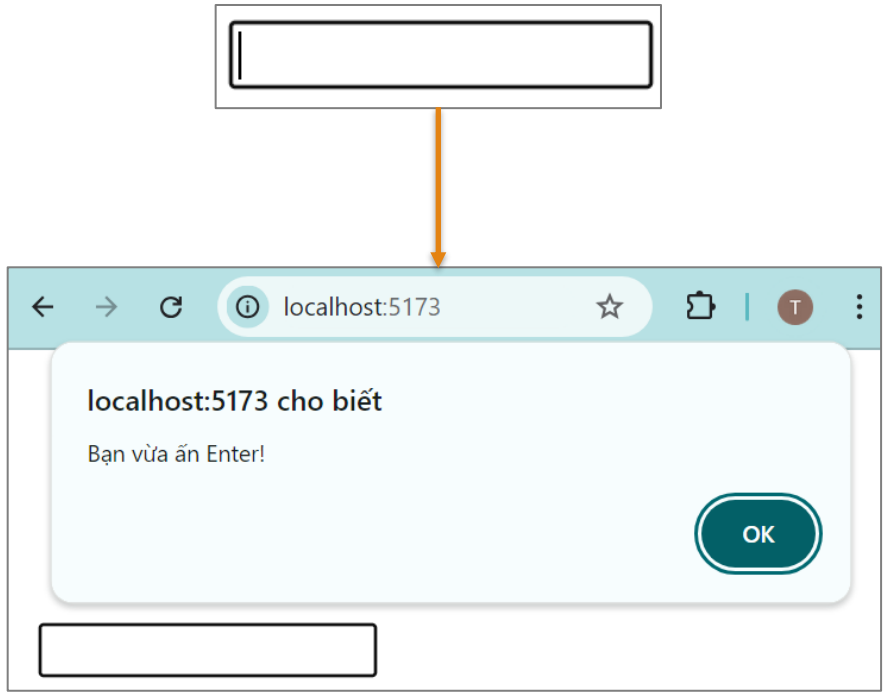
- Vue đã cung cấp **alias** (tên phím mà con người hiểu được) cho một số phím thông dụng:
  - .enter
  - .delete
  - .tab
  - .esc
  - .space
  - .up
  - .down
  - .left
  - .right

# Key modifiers

- **Ví dụ:** Bắt sự kiện nhấn phím enter.

```
<script setup>
const warn = () => {
  alert('Bạn vừa ấn Enter!');
};
</script>

<template>
  <div>
    <input type="text" v-on:keyup.enter="warn" />
  </div>
</template>
```



## System Modifier Keys

- Trường hợp một số phím chuyên biệt (phím hệ thống Ctrl, Alt, Shift, Meta và các nút ở trên chuột) sẽ có các modifier riêng.

- .ctrl
- .alt
- .shift
- .meta

- Những phím hệ thống này thường không được ấn một mình mà sẽ kết hợp thành một tổ hợp phím (đó là lí do vì sao nó trở nên đặc biệt). Ví dụ:

```
<!-- Alt + C --> <input @keyup.alt.67="method">
<!-- Ctrl + Click --> <div @click.ctrl="method">Ctrl click để chạy</div>
```

## .exact Modifier

- Sử dụng trong trường hợp cho phép kiểm soát sự kết hợp chính xác của các công cụ sửa đổi hệ thống cần thiết để kích hoạt một sự kiện.

```
<!-- Sẽ kích hoạt ngay cả khi nhấn Alt hoặc Shift -->
<button @click.ctrl="onClick">A</button>

<!-- Sẽ kích hoạt khi nhấn Ctrl và không có phím nào khác được nhấn-->
<button @click.ctrl.exact="onCtrlClick">A</button>

<!-- Sẽ kích hoạt khi không có System Modifier nào được nhấn-->
<button @click.exact="onClick">A</button>
```

## Mouse Button Modifiers

- .left
- .right
- .middle

- Những công cụ sửa đổi này hạn chế trình xử lý đối với các sự kiện được kích hoạt bởi một nút chuột cụ thể.

FPT POLYTECHNIC

# BÀI 7:

# EVENT HANDLING VÀ FORM BINDING TRONG VUEJS

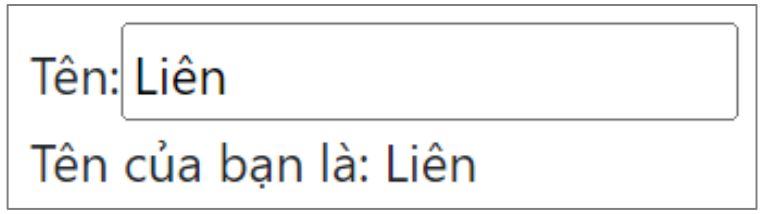
## PHẦN II: FORM BINDING VUEJS

# TỔNG QUAN VỀ FORM BINDING

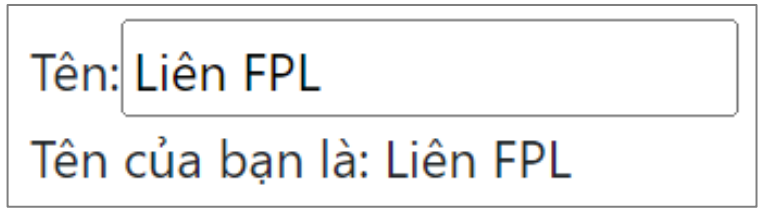
**Form binding** trong VueJS là cách thức liên kết dữ liệu của một mẫu (form) HTML với dữ liệu trong component. VueJS cung cấp các chỉ thị (directives) như **v-model** để giúp việc liên kết này trở nên đơn giản và hiệu quả.

# BINDING ĐƠN GIẢN

**Text input:** Binding đơn giản với input type text sử dụng v-model




Thay đổi nội dung trong <input>:




```
<template>
  <div>
    <label for="name">Tên:</label>
    <input id="name" v-model="name" type="text">
    <p>Tên của bạn là: {{ name }}</p>
  </div>
</template>

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

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

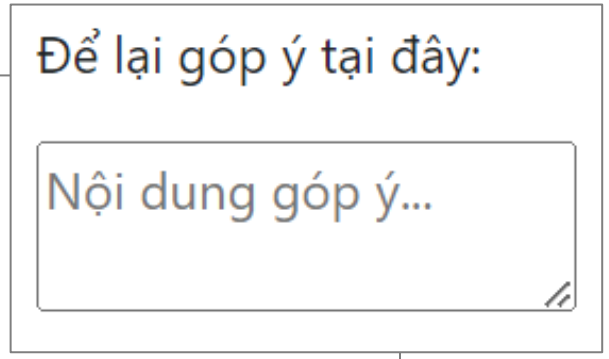
# BINDING ĐƠN GIẢN

## **Multiline text:** Văn bản nhiều dòng

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

const message = ref('')
</script>

<template>
    <span>Để lại góp ý tại đây:</span>
    <p>{{ message }}</p>
    <textarea v-model="message" placeholder="Nội dung góp ý..."></textarea>
</template>
```



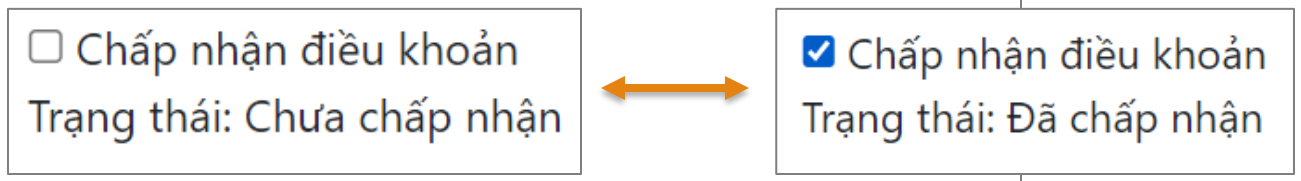
# BINDING ĐƠN GIẢN

## **Checkbox:** Binding với checkbox sử dụng v-model

```
<template>
  <div>
    <label>
      <input type="checkbox" v-model="isChecked">
      Chấp nhận điều khoản
    </label>
    <p>Trạng thái: {{ isChecked ? 'Đã chấp nhận' : 'Chưa chấp nhận' }}</p>
  </div>
</template>

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

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



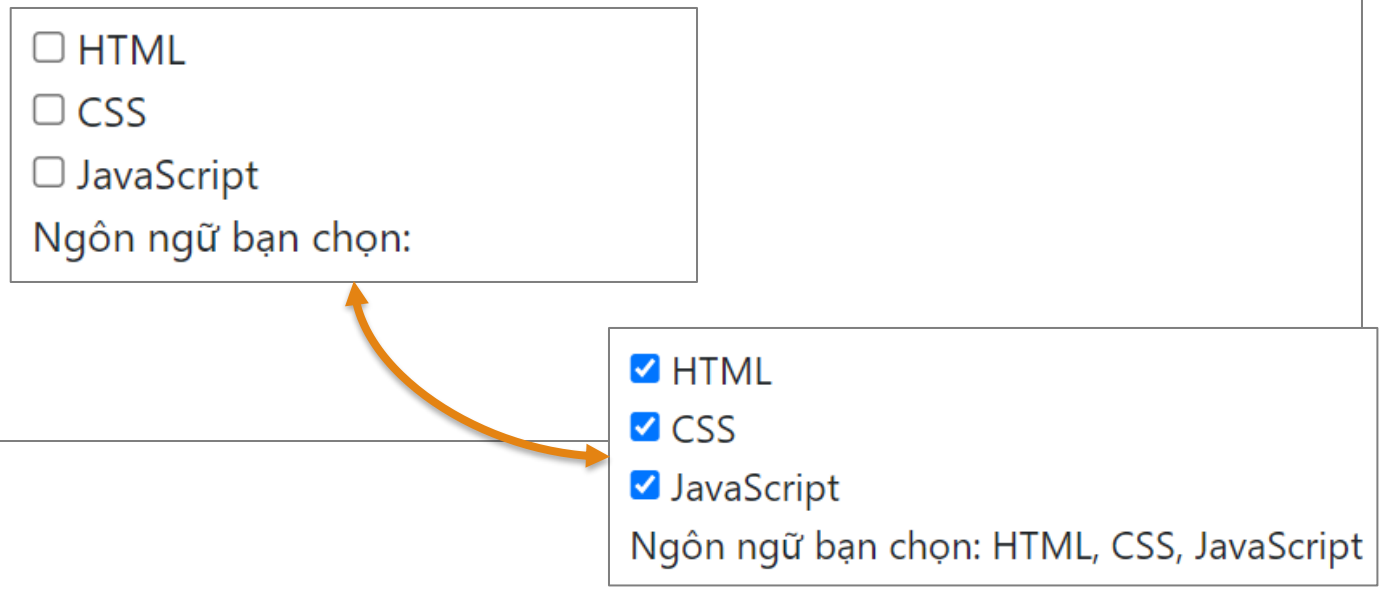
# BINDING ĐƠN GIẢN

## Multiple Checkboxes:

```
<template>
  <div>
    <input type="checkbox" value="HTML" v-model="checkedNames">HTML <br>
    <input type="checkbox" value="CSS" v-model="checkedNames">CSS <br>
    <input type="checkbox" value="JavaScript" v-model="checkedNames">JavaScript <br>
    <p>Ngôn ngữ bạn chọn: {{ checkedNames.join(', ') }}</p>
  </div>
</template>

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

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



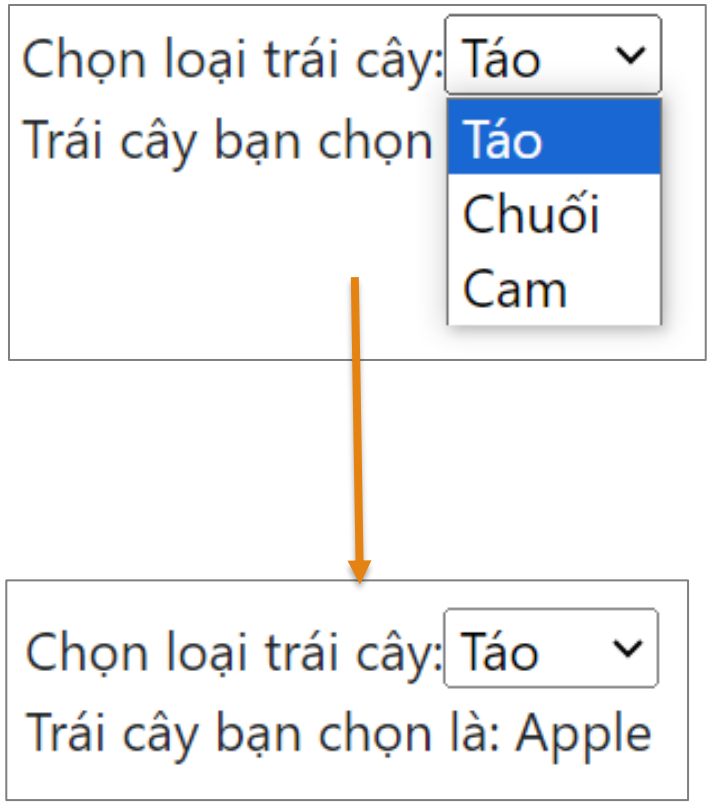
# BINDING ĐƠN GIẢN

## Select/Option:

```
<template>
  <div>
    <label for="fruit">Chọn loại trái cây: </label>
    <select id="fruit" v-model="selectedFruit">
      <option value="Apple">Táo</option>
      <option value="Banana">Chuối</option>
      <option value="Orange">Cam</option>
    </select>
    <p>Trái cây bạn chọn là: {{ selectedFruit }}</p>
  </div>
</template>

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

const selectedFruit = ref('Apple');
</script>
```



## **Value Bindings:** Ràng buộc giá trị

Đối với các tùy chọn radio, checkbox và select option, các giá trị liên kết v-model thường là các chuỗi tĩnh (hoặc boolean cho checkbox):

```
<!-- `picked` là một chuỗi "a" khi được chọn -->
<input type="radio" v-model="picked" value="a" />

<!-- `toggle` là true hoặc false -->
<input type="checkbox" v-model="toggle" />

<!-- `selected` là một chuỗi "abc" khi tùy chọn đầu tiên được chọn -->
<select v-model="selected">
  <option value="abc">ABC</option>
</select>
```

# V-MODEL VỚI CÁC THUỘC TÍNH KHÁC

Trong VueJS, có thể tùy chỉnh các thuộc tính khác ngoài value, ví dụ như checked, selected, innerHTML, v.v., bằng cách sử dụng v-bind cùng với v-model.

```
<template>
   <input
      type="text"
      v-model="message"
      :placeholder="placeholder"
    />
</template>

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

const message = ref('');
const placeholder = ref('Type something here...');
</script>
```

# V-MODEL VỚI CÁC THUỘC TÍNH KHÁC

## Sử dụng **.lazy**, **.number**, và **.trim** modifiers

- **.lazy**: Chỉ cập nhật giá trị sau khi sự kiện change được kích hoạt thay vì input.
- **.number**: Chuyển đổi giá trị input thành kiểu số.
- **.trim**: Tự động loại bỏ khoảng trắng ở đầu và cuối của chuỗi.

```
<template>
  <input v-model.lazy="name" />
  <input v-model.number="age" type="number" />
  <input v-model.trim="text" />
</template>
```

# V-MODEL VỚI COMPONENT

Khi sử dụng v-model với component tùy chỉnh, VueJS tự động thực hiện 2 bước:

- **Bước 1**: Sử dụng '**props**' để nhận giá trị ('**modelValue**')

Trong component con, cần khai báo một '**props**' để nhận giá trị từ v-model. Theo mặc định, v-model sẽ liên kết với một '**props**' có tên là '**modelValue**'.

- **Bước 2**: Phát sự kiện '**update:modelValue**' để cập nhật giá trị

Khi giá trị trong component con thay đổi (ví dụ khi người dùng nhập dữ liệu vào input), cần phát ra sự kiện '**update:modelValue**' để thông báo cho component cha biết rằng giá trị đã thay đổi. Sự kiện này sẽ cập nhật giá trị liên kết trong component cha.

# V-MODEL VỚI COMPONENT

```vue
<template>
  <input :value="modelValue" @input="updateValue" />
</template>

<script>
export default {
  props: ['modelValue'], // Nhận giá trị từ v-model
  methods: {
    updateValue(event) {
        // Phát sự kiện để cập nhật giá trị
      this.$emit('update:modelValue', event.target.value);
    }
  }
}
</script>
```

CustomInput.vue

- Component cha `ParentComponent.vue` sử dụng v-model để liên kết một biến với giá trị từ component con.
- Component con `CustomInput.vue` nhận giá trị từ v-model thông qua **props** và phát sự kiện **update:modelValue** khi giá trị thay đổi.

```vue
<template>
  <div>
    <CustomInput v-model="inputValue" />
    <p>Input Value: {{ inputValue }}</p>
  </div>
</template>

<script>
import { ref } from 'vue';
import CustomInput from './CustomInput.vue';
export default {
  components: { CustomInput },
  setup() {
    const inputValue = ref('');

    return {
      inputValue,
    };
  },
};
</script>
```

ParentComponent.vue


Tái hiện demo sử dụng v-model với component tùy chỉnh

# TỔNG KẾT NỘI DUNG BÀI HỌC

- ☑ Dùng directive v-on để lắng nghe sự kiện và thực thi JavaScript khi những sự kiện này được kích hoạt
- ☑ Trình xử lý phương thức với cách tách phần code xử lý vào hàm và gọi hàm đó ở trên sự kiện như javascript thuần.
- ☑ Vue cung cấp một số modifier sự kiện khác nhau hữu ích trong các tình huống xử lý sự kiện phổ biến
- ☑ Form binding trong VueJS là cách thức liên kết dữ liệu của một mẫu (form) HTML với dữ liệu trong component.


Thank
You