Programação Web I

M01T03 - Introduction to Vue.js (Events & Forms)

ESMAD | TSIW | 2022-23

M01 - Introduction to Vue.js

T03 - Events & Forms

1. Event handling
2. Forms binding
3. Template refs

M01 - Introduction to Vue.js

T03 - Events & Forms

1. Event handling
2. Forms binding
3. Template refs

M01 - Introduction to Vue.js

1. Event handling

- Using of the v-on directive to listen for DOM events and execute JavaScript when they fire

```
data: {
    counter: 0
}
```

```
<div id='intro'>
    <button v-on:click='counter +=1'></button>
    <p>The button was clicked {{ counter }} times!</p>
</div>
```

M01 - Introduction to Vue.js

1. Event handling

- Writing JavaScript code in v-on attribute value is not feasible
- v-on also accepts the name of a method to be run when event triggers

```
<div id='intro'>
    <button v-on:click='greet'>Greet</button>
</div>
```

```
data: { name: 'Vue.js' },
// Define methods in the methods object
methods: {
    greet: function (event) {
        // `this` inside methods refers to the Vue instance
        alert('Hello ' + this.name + '!')
        // `event` is the DOM native event
        if (event) {
            alert(event.target.tagName)
        }
    }
}
```

M01 - Introduction to Vue.js

1. Event handling

- Instead of binding directly to a method name, we can also use parameterized methods in an inline JavaScript statement:

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

```
<div id='intro'>
    <button v-on:click='say("hello")'> Say hello</button>
    <button v-on:click='say("good morning")'>Say good morning </button>
</div>
```

M01 - Introduction to Vue.js

1. Event handling

- Sometimes we need to access the original DOM event in an inline instruction handler
- You can pass it in a method using the special $event variable:

```
methods: {
    warn: function (message, event) {
        if (event) event.preventDefault()
        alert(message)
    }
}
```

```
<div id='intro'>
    <button v-on:click='warn("Form não pode ser submetido", $event)'>Submit</button>
</div>
```

M01 - Introduction to Vue.js

1. Event handling

- Event modifiers
  - Another common need is to call event.preventDefault() or event.stopPropagation() in event handlers.
  - Although is easy do this with methods, it's best if the methods were purely data logic rather than dealing with DOM event details
  - To address this issue, Vue provides event modifiers as v-on suffixes:
    - .stop
    - .prevent
    - .capture
    - .self
    - .once
    - .passive

M01 - Introduction to Vue.js

1. Event handling

- Event modifiers
  - Another common need
    event.stopPropagation(
  - Although is easy do this
    purely data logic rather
  - To address this issue, V
    - .stop
    - .prevent
    - .capture
    - .self
    - .once
    - .passive

```
<!-- click event propagation will stop -->
<a v-on:click.stop="doThis"></a>
<!-- submit event will not reload the page -->
<form v-on:submit.prevent="onSubmit"></form>
<!-- modifiers can be chained -->
<a v-on:click.stop.prevent="doThat"></a>
<!-- only the modifier -->
<form v-on:submit.prevent></form>
<!-- Use capture mode when adding event listener ->
<! - That is, an event that targets an inner element is
    handled here before it is handled by that element -->
<div v-on:click.capture="doThis">...</div>
<!-- only triggers the handler if event.target is the
    element itself, not a child element -->
<div v-on:click.self="doThat">...</div>
<!-- modifier works only once -->
<div v-on:click.once="doThat">...</div>
```

M01 - Introduction to Vue.js

1. Event handling

- Key modifiers
  - When listening to keyboard events, we need to check key codes
  - Vue lets you add v-on keyboard modifiers when listening to key events
  - See the completed list of modifier alias:
    - .enter
    - .tab
    - .esc
    - .space
    - .delete (capture bothe keys “Delete” and “Backspace”)
    - .up, .down, .left e .right

```
<!-- Call submit if the key presses is the ENTER -->
<input v-on:keyup.enter='submit'>

<!-- Abreviated form -->
<input @keyup.enter='submit'>
```

M01 - Introduction to Vue.js

1. Event handling

- System modifiers keys
  - You can use the following modifiers to trigger mouse or keyboard event listeners only when the corresponding modifier key is pressed:
    - .ctrl
    - .alt
    - .shift
    - .meta

```
<!-- Alt + C -->
<input @keyup.alt.67='clear'>

<!-- Ctrl + Click -->
<div @click.ctrl='doSomething'>Do something</div>
```

M01 - Introduction to Vue.js

1. Event handling

- System modifiers keys
  - The .exact modifier allows you to control the exact combination of system modifiers needed to trigger an event

```
<!-- fires even if Alt or Shift is pressed -->
<button @click.ctrl="onClick">A</button>

<!--  only fires when Ctrl and no other key is pressed -->
<button @click.ctrl.exact="onCtrlClick">A</button>

<!-- fires when no System modifier is pressed  -->
<button @click.exact="onClick">A</button>
```

M01 - Introduction to Vue.js

1. Event handling

- Mouse modifiers
  - Mouse modifiers restrict the handler to events triggered by a specific mouse button:
    - .left
    - .right
    - .middle

M01 - Introduction to Vue.js

T03 - Events & Forms

1. Event handling
2. Forms binding
3. Template refs

M01 - Introduction to Vue.js

2. Forms binding

- Use v-model directive to create a two-way data binding on form elements
- v-model will ignore value, checked, or selected attributes on any element
- Real value should be assign with the data function of the Vue component

```
data() {
  return {
    message: "",
```

```
<div>
  <input v-model="message" />
  <p>Message is {{ message }}</p>
</div>
```

Binding of input element to the component's data

O Vue é uma framework porreira!

Message is: O Vue é uma framework porreira!

M01 - Introduction to Vue.js

2. Forms binding

Checkbox

- Associate value to the label element for better user experience

```
data() {
  return {
    checked: false,
```

```
<div>
  <input id="idChecked" type="checkbox" v-model="checked">
  <label for="idChecked"> {{ message }}</label>
</div>
```

☐ false ☑ true

M01 - Introduction to Vue.js

2. Forms binding

Checkbox

- Multiple checkboxes, linked to the same array

```
data() {
  return {
    checkedNames: [],
```

```
<div>
  <input type="checkbox" id="idR" value="Ricardo" v-model="checkedNames" />
  <label for="idR">Ricardo</label>
  <input type="checkbox" id="idM" value="Maria" v-model="checkedNames" />
  <label for="idM">Maria</label>
  <input type="checkbox" id="idJ" value="João" v-model="checkedNames" />
  <label for="idJ">João</label>
  <br />
  <span>Checked names: {{ checkedNames }}</span>
</div>
```

☑ Ricardo ☐ Maria ☑ João
Checked names: ["Ricardo", "João"]

M01 - Introduction to Vue.js

2. Forms binding

Radiobox

- Use of radio buttons

```
data() {
  return {
    picked: "One",
```

```
<div>
  <input type="radio" id="one" value="One" v-model="picked" />
  <label for="one">One</label>
  <br />
  <input type="radio" id="two" value="Two" v-model="picked" />
  <label for="two">Two</label>
  <br />
  <span>Picked: {{ picked }}</span>
</div>
```

One
Two
Picked: One

M01 - Introduction to Vue.js

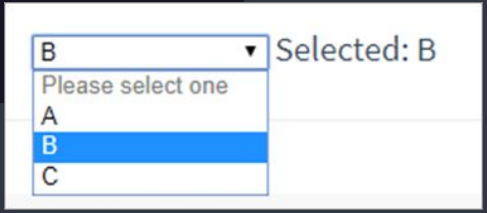
2. Forms binding

Select

- Use of listbox

```
data() {
  return {
    selected: "",
```

```
<div>
  <select v-model="selected">
    <option disabled value="">Please select one</option>
    <option>A</option>
    <option>B</option>
    <option>C</option>
  </select>
  <span>Selected: {{ selected }}</span>
</div>
```



M01 - Introduction to Vue.js

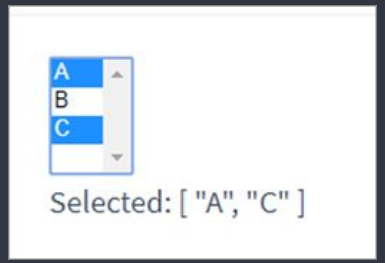
2. Forms binding

Multi-select

- Use of listbox multi-select (use CRTL to select more than one option)

```
data() {
  return {
    selected: [],
```

```
<div>
  <select v-model="selected" multiple>
    <option>A</option>
    <option>B</option>
    <option>C</option>
  </select>
  <span>Selected: {{ selected }}</span>
</div>
```



M01 - Introduction to Vue.js

2. Forms binding

Select with dynamic data

- Dynamic options processed with v-for

```
data() {
  return {
    selected: '',
    options: [
        { text: 'One', value: 'A' },
        { text: 'Two', value: 'B' },
        { text: 'Three', value: 'C' }
      ],
```

```
<div>
  <select v-model="selected">
    <option v-for="option in options" v-bind:key="option.value">
      {{ option.text }}
    </option>
  </select>
  <span>Selected: {{ selected }}</span>
</div>
```

Two ▾ Selected: B

M01 - Introduction to Vue.js

2. Forms binding

Dynamic values binding

- For radio, checkbox and listboxes, v-model binding values are usually static (or boolean to checkbox) strings

```
<div>
  <!-- `picked` is a string "a" when selected -->
  <input type="radio" v-model="picked" value="a" />

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

  <!-- `selected` is a string "abc" when first option is selected -->
  <select v-model="selected">
    <option value="abc">ABC</option>
  </select>
</div>
```

M01 - Introduction to Vue.js

2. Forms binding

Modifiers

- Modify the behaviour of forms binding
- Some examples: lazy, number and trim

M01 - Introduction to Vue.js

2. Forms binding

Modifiers > lazy

- By default, v-model sync data after each input event
- Add lazy modifier to sync only after change event

```
<div>
  <input v-model.lazy="msg" />
  <span>{{ msg }}</span>
</div>
```

M01 - Introduction to Vue.js

2. Forms binding

Modifiers > number

- If you want entered data to be automatically converted to number, add the number modifier

```
<div>
  <input v-model.number="age" type="number" />
</div>
```

- This is useful since even type="number" the input value returns a string
- If the value cannot be parsed with parseFloat(), the original value returns

M01 - Introduction to Vue.js

2. Forms binding

Modifiers > trim

- If you want the input data to be adjusted automatically, you can include the trim modifier

```
<div>
  <input v-model.trim="msg" />
  <span>{{ msg }}</span>
</div>
```

M01 - Introduction to Vue.js

T03 - Events & Forms

1. Event handling
2. Forms binding
3. Template refs

M01 - Introduction to Vue.js

3. Template refs

- There may still be cases where we need direct access to the underlying DOM elements
- To achieve this, we can use the special ref attribute

```
<input ref="input">
```

- Allows us to obtain a direct reference to a specific DOM element or child component instance after it's mounted
- This may be useful when you want to, for example, programmatically focus an input on component mount, or initialize a 3rd party library on an element

M01 - Introduction to Vue.js

3. Template refs

Accessing the Refs

- The resulting ref is exposed on this.$refs

- Note that you can only access the ref after the component is mounted
- If you try to access $refs.input in a template expression, it will be null on the first render. This is because the element doesn't exist until after the first render!

```
<script>
export default {
  mounted() {
    this.$refs.input.focus()
  }
}
</script>

<template>
  <input ref="input" />
</template>
```

M01 - Introduction to Vue.js

3. Template refs

Refs inside v-for

- When ref is used inside v-for, the resulting ref value will be an array containing the corresponding elements:

- It should be noted that the ref array does not guarantee the same order as the source array.

```
<script>
export default {
  data() {
    return {
      list: [
        /* ... */
      ]
    }
  },
  mounted() {
    console.log(this.$refs.items)
  }
}
</script>

<template>
  <ul>
    <li v-for="item in list" ref="items">
      {{ item }}
    </li>
  </ul>
</template>
```