**Section 2 – Basics And Core Concepts – DOM Interaction**

Chapter 14 – Interpolation And Data Binding

1. Interpolation means {{ courseGoal }} in HTML file to print the return value from Vue. It also means you can reference properties, that are part of that object, return data, and then the value of the property will be output.

Chapter 15 – Binding Attributes With The “V-Bind” Directive

1. {{ }} only can be used inside HTML tag
2. v-bind is used to dynamically set a value of an HTML element attribute, in this case of the ref attribute, such as v-bind:href = ”vueLink”, can be used to bind to set the values of attributes.

Chapter 18 – Outputting RAW HTML Content with V-HTML

1. v-html set the content between opening and closing text and tell Vue that the content should be interpreted as HTML.

Summary

1. v-model

<input type="text" id="goal" v-model="enteredValue" />

v-model is used to print the return value (enteredValue) or get the input value (enteredValue)

or

<input type="text" v-bind:value="name" />

v-bind:value=”name” is used to print return value from variable name, the result is same when using v-model.

1. v-on:click”addGoal”

<button v-on:click="addGoal">Add Goal</button>

v-on:click”addGoal” is used to give event onClick name addGoal

1. v-for

<li v-for="goal in goals">{{ goal }}</li>

v-for is used to print the array value goals (like foreach)

1. v-bind

<img v-bind:src="imageLink" />

v-bind is used to give link value such as hyperlink or image

1. v-html

<p v-html="outputGoal()"></p>

v-html is used to print return value which has html tag inside the return value

Chapter 20 – Understanding Event Binding

1. In Vue, we just declare where we you want to have EventListeners and where you want to output the values, and which values you should be aware of. Vue will manage those listeners, updating the counter behind the scenes (adding and reducing), detecting when the counter changes, and updating the parts of the real rendered page.

Chapter 21 – Events and Methods

1. Write v-on:click=”add” or v-on:click=”add()” for method will give same result.

<button v-on:click="add">Add</button>

<button v-on:click="add()">Add</button>

Chapter 22 – Working With Event Arguments

1. Add parameter on the method in Vue

methods: {

    add (num) {

      this.counter += num;

    },

    reduce (num) {

      this.counter -= num;

    }

In HTML, the function on click will be

<button v-on:click="add(5)">Add 5</button>

<button v-on:click="reduce(5)">Reduce 5</button>

Chapter 23 – Using the Native Event Object

1. We can add parameter from input text in HTML like below.

methods: {

    setName (event, lastName) {

      this.name = event.target.value + ' ' + lastName;

    }

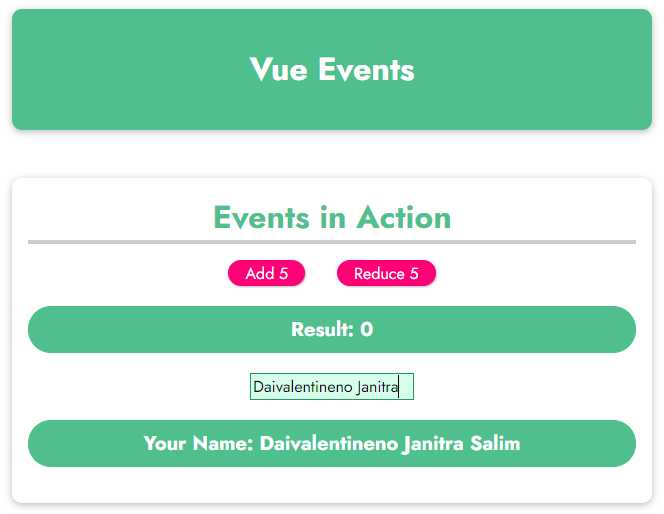
  }

In HTML, we write like below.

<input type="text" v-on:input="setName($event, 'Salim')">

<p>Your Name: {{ name }}</p>

The result is shown in Figure below.



Chapter 24 – Exploring Event Modifiers

1. If we don’t want to submit the form (page will not reload), when we click submit button, we can add this in the JavaScript code.

methods: {

    submitForm (event) {

      event.preventDefault();

      alert('Submitted !')

    }

The HTML code will be like this.

<form v-on:submit="submitForm">

Or we can add .prevent on HTML code like below. It will have same result.

<form v-on:submit.prevent="submitForm">

And on the JavaScript code will be like this.

methods: {

    submitForm () {

      alert('Submitted !')

    }

1. Right, Left, and Middle Click

<button v-on:click.right="reduce(5)">Reduce 5</button>

1. Using key up for get keyboard event listener

<input

     type="text"

     v-on:input="setName($event, 'Salim')"

     v-on:keyup.enter="confirmInput"

   />

After we input the name and press Enter, the confirmed name will be printed. The Vue code can be seen below.

methods: {

    confirmInput() {

      this.confirmedName = this.name;

    }

Chapter 25 – Locking Content with v-once

1. v-once is used to print the return value only once and it will not changed even the value will be changed. The HTML code is like below.

<p v-once>Starting Counter: {{ counter }}</p>

Chapter 26 – Data Binding + Event Binding = Two Way Binding

1. We may change v-bind:value + v-on:input = v-model

<input type="text" v-bind:value='name' v-on:input="setName($event, 'Salim')">

The result will be like this. Note: v-model is two way binding.

<input type="text" v-model="name">

1. Add last name and when reset it will be erased all the input, but this code is not good enough, so the best strategy will be discussed on the next Chapter.

methods: {

    outputFullname() {

      if (this.name === ''){

        return '';

      }

      return this.name + ' ' + 'Salim';

    }

Chapter 28 – Introducing Computed Properties

1. To give the best strategy for adding last name into the full name, the strategy like below.

computed: {

    fullname() {

      if (this.name === ''){

        return '';

      }

      return this.name + ' ' + 'Salim';

    },

  }

And the HTML will be like below.

<p>Your Name: {{ fullname }}</p>

Note: computed: only recalculate and reevaluate if one of the dependencies in this case, the only dependency we have the name property changed.

Chapter 29 – Working with Watchers

1. Watcher is basically a function you can tell Vue to execute, when one of its dependencies changed.
2. Watcher will update or reexecute automatically every property name is changed. The code will be like below. Value means the last value of the property.

watch: {

    name(value) {

      if (value === '') {

        this.fullname = '';

      } else {

        this.fullname = value + ' ' + this.lastName;

      }

    },

    lastName(value) {

      if (value === '') {

        this.lastName = '';

      } else {

        this.fullname = this.name + ' ' + value;

      }

    }

  },

1. Using computed

computed: {

    fullname() {

      if (this.name === '' || this.lastName === ''){

        return '';

      }

      return this.name + ' ' + this.lastName;

    },

  },

1. Watcher vs Computed

Watchers are powerful, if you have a different kind of intent in mind, such as if the counter more than 50, we need to reset it, Watcher may do better job than Computed. The code will be like below.

Chapter 30 – Methods vs Computed Properties vs Watchers

1. Methods

* Use with event binding OR data binding
* Data binding: Method is executed for every “re-ender” cycle of the component
* Use for events or data that really need to be re-evaluated all the time.

1. Computed

* Use with data binding
* Computed properties are only re-evaluated if one of their “used values” changed
* Use for data that depends on other data

1. Watch

* Not used directly in template
* Allows you to run any code in reaction to some changed data (e.g. send HTTP request etc.)
* Use for any non-data update you want to make

Chapter 30 – v-bind and v-on Shorthands

We can change v-on:click=”resetInput” into @click=”resetInput”

We can change v-bind:value=”name” into :value=”name”

Chapter 32 – Dynamic Styling With Inline Styles

1. Write CSS in HTML using Vue (and IF)

<div class="demo"

        v-bind:style="{borderColor: boxASelected ? 'red' : '#ccc'}"

        v-on:click="boxSelected('A')"></div>

The Vue Code will be like below.

methods: {

        boxSelected(box) {

            if (box === 'A') {

                if (this.boxASelected === false)

                    this.boxASelected = true;

                else

                    this.boxASelected = false;

            }

        }

    }

Chapter 33 – Adding CSS Classes Dynamically

1. We can make CSS dynamically with making a new CSS class.

.active {

  border-color: red;

  background-color: salmon;

}

And edit a little on the HTML code like below.

<div class="demo"

        :class="{active: boxASelected}"

        @click="boxSelected('A')"

      ></div>

Chapter 34 – Classes And Computed Properties

1. We can make a new class in computed using Vue, so it will return CSS Style.

computed: {

        boxAClasses() {

            return { active: this.boxASelected }

        }

    }

And the HTML code will be like below.

<div class="demo"

        :class="boxAClasses"

        @click="boxSelected('A')"

      ></div>

Chapter 35 – Dynamic Classes: Array Syntax

1. We can add two classes in v-bind using Array, the HTML code will be like below.

<div

        :class="['demo', { active: this.boxASelected }]"

        @click="boxSelected('A')"

      ></div>

Summary

1. DOM and Templates

* Vue can be used to define the goal instead of the steps (declarative approach), we can define how our page should look like and which part on the page might be dynamic and we let them Vue figure out how to get there. We need to connect our Vue app, our Vue instance, and the actual HTML code with the mound method which we saw.
* Connect Vue to HTML via “mount”: Vue then renders the real DOM based on the connected template.

1. Data and Event Binding

* You can bind data via interpolation ({{}}) or the v-bind (“:”) directive.
* You listen for events via v-on (“@”)

1. Reactivity

* Vue updates the real DOM for you when bound data changes. BINDE our double curly braces changes and you learned that you can register data which might change in the data object of your Vue app, and that you can also work with computed properties and watchers.
* Computed properties and watchers allow you to react to data changes to calculate results dynamically or to run some code.

1. Styling

* Achieving dynamic styling because we can make CSS classes or inline styles dynamically with the bind. Dynamic CSS class and inline style bindings are supported by Vue.
* Vue offers multiple special syntaxes (object-based, array-based) for efficient bindings.