Two way data binding using v-model

Example 01: The data floats both ways and updates its values.

<div id="app">

<input type="text" v-model="name">

<p><strong>Name: </strong>{{name}}</p>

</div>

<script>

new Vue({

el:'#app',

data:{

name: "",

age:28,

},

});

</script>

N.B, if we set default value for name: "Bablu Ahmed" and in the input element like value=”Test” then Vue will ignore the default value.

Example 02: Modifiers for v-model directive.

Trim, Number (parse.int in js),

<div id="app">

<input type="text" v-model.number="age">

<p><strong>Name: </strong>{{name}}</p>

<p><strong>Type: </strong> {{ typeof age }}</p>

</div>

<script>

new Vue({

el:'#app',

data:{

name: "Bablu Ahmed",

age:28,

},

});

</script>

Example 03: The lazy modifier update data value after the whole thing is written like jquery change event

<div id="app">

<input type="text" v-model.number.lazy="age">

<p><strong>Name: </strong>{{age}}</p>

<p><strong>Type: </strong> {{ typeof age }}</p>

</div>

<script>

new Vue({

el:'#app',

data:{

name: "Bablu Ahmed",

age:28,

},

});

</script>

Outputting HTML Elements

For xss (Cross Site Security) and loading time consuming we need to render HTML by default it doesn’t encoded if include v-html then it shows as usual.

For example 01:

<div id="app">

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

</div>

<script>

new Vue({

el:'#app',

data:{

html: "<h1>Bablu Ahmed</h1>",

},

});

</script>

N.B, Below code will not work

<div id="app">

{{ html }}

</div>

Example 02: Outputting HTML elements only one time or just it will show initial data and never be changed anymore. If we use @click.once then it will only be able one time second time click will not acceptable.

<div id="app">

<h1 v-once>{{ title }}</h1>

<button v-on:click="title = 'Amar sonar Bangla'">Update once </button>

</div>

<script>

new Vue({

el:'#app',

data:{

title: "What is Vue JS?",

},

});

</script>

**Conditionally Rendering Elements**

if, else if, and else statement

Example 01:

<div id="app">

<p v-if="itemsInStock > 10">{{itemsInStock}} in stock</p>

<p v-else-if="itemsInStock > 0">Hurry up, there are just a few items left!</p>

<p v-else>Too bad, we're all out!</p>

</div>

<script>

new Vue({

el:'#app',

data:{

itemsInStock: 0,

},

});

</script>

Example 02: We can use <template></template> markup for multiple elements and the markup will not render in html.

<div id="app">

<p v-if="itemsInStock > 10">{{itemsInStock}} in stock</p>

<p v-else-if="itemsInStock > 0">Hurry up, there are just a few items left!</p>

<p v-else>Too bad, we're all out!</p>

<template v-if="itemsInStock > 10">

<p>special Offer!</p>

<p>Looks like we have tons of products in stock...save 20% if you order now!</p>

</template>

</div>

<script>

new Vue({

el:'#app',

data:{

itemsInStock: 15,

},

});

</script>

Showing and Hiding Elements (Toggle)

Example 01:

<div id="app">

<p v-show="isBablu">Yes, I am Bablu</p>

<p v-show="!isBablu">No, I am not Bablu</p>

<button v-on:click="isBablu=!isBablu">Toggle</button>

</div>

<script>

new Vue({

el:'#app',

data:{

isBablu: true,

},

});

</script>

Hiding Elements until until the Vue instance is Ready

<style>

[v-cloak]{

display: none;

}

</style>

<div id="app">

<p v-cloak>{{ message }}</p>

</div>

<script>

setTimeout(function() {

new Vue({

el:'#app',

data:{

message: "Hi, I am Bablu Ahmed",

},

});

}, 5000);

</script>

Looping through Elements

**Arrays:**

<div id="app">

<h2>Songs</h2>

<ul>

<li v-for="title in songTitles">{{title}}</li>

</ul>

</div>

<script>

new Vue({

el:'#app',

data:{

songTitles: ['Jhoom','Jhor','Jol'],

employees: [

{"name":"Bablu Ahmed", "title":"Programmer"},

{"name":"Abdul Kuddus", "title":"Vue JS Expert"},

{"name":"Sakibur Rahman", "title":"Marketing Manager"},

],

companyName: "w3public.com",

},

});

</script>

**Objects:**

<div id="app">

<h2>Empployee Details</h2>

<table>

<tr>

<th>Name</th>

<th>Title</th>

<th>Company</th>

</tr>

<tr v-for="employee in employees">

<td>{{employee.name}}</td>

<td>{{employee.title}}</td>

<td>{{companyName}}</td>

</tr>

</table>

</div>

Accessing Loop index:

<tr v-for="(employee, index) in employees">

<td>{{employee.name}}</td>

<td>{{employee.title}}</td>

<td>{{companyName}}</td>

<td>{{index}}</td>

</tr>

Looping through object properties:

<div id="app">

<h2>Empployee Details</h2>

<ul>

<li v-for="(value, property, index) in person">{{property}}: {{value}}({{index}})</li>

</ul>

</div>

<script>

new Vue({

el:'#app',

data:{

songTitles: ['Jhoom','Jhor','Jol'],

person: {"name":"Bablu Ahmed", "title":"Programmer"},

companyName: "w3public.com",

},

});

</script>

Looping through number ranges:

<ul>

<li v-for="n in 10">{{n}}</li>

</ul>

With index:

<li v-for="(n, index) in 10">

{{n}}\*{{index}} = {{n\*index}}

</li>

Output:

1\*0 = 0

2\*1 = 2

3\*2 = 6

4\*3 = 12

5\*4 = 20

6\*5 = 30

7\*6 = 42

8\*7 = 56

9\*8 = 72

10\*9 = 90

Array: By default Vue doesn’t change any array value, if needed we can change using Vue.set() method.

<ul>

<li v-for="(array, index) in numbers">

{{array}}

</li>

</ul>

<button v-on:click="changeNumber">Change Number</button>

<script>

new Vue({

el:'#app',

data:{

numbers: [1,2,3,4],

},

methods:{

changeNumber: function () {

Vue.set(this.numbers, 1, 10);

alert(this.numbers[1]);

}

},

});

</script>

**Computed Properties**: Computed properties are also function but when we will use it doesn’t need parenthesis. We can optimize performance using computed properties. This is because it doesn’t change a value of the dada until change it with new value.

For example:

<p>{{getFullname()}}</p>

<p>{{counter}}</p>

<button v-on:click="counter++">Full Name</button>

<script>

new Vue({

el:'#app',

data:{

counter:1,

firstName:"Bablu",

lastName:"Ahmed",

},

methods:{

getFullname: function () {

alert("Assembling fullname...");

return this.firstName+" "+this.lastName;

}

},

});

</script>

In above code, we can see the same code is executing again and again in every single click although the value is not change of the data property.

We can solve it by using computed property as follows:

<p>{{getFullName}}</p>

<p>{{counter}}</p>

<button v-on:click="counter++">Full Name</button>

<script>

new Vue({

el:'#app',

data:{

counter:1,

firstName:"Bablu",

lastName:"Ahmed",

},

computed:{

getFullName: function () {

alert("Assembling fullname...");

return this.firstName+" "+this.lastName;

}

},

});

</script>

Here, we can see that the value of farstName and lastName is not been changed and hence the alert is not showing and the of counter is been changed and it is showing.

Character Length length checking Application

<div id="app">

<!--two-way data binding to update the value of data-->

<input v-model="password" type="password">

<p v-if="password.length > 6">Awesome! Your password is longer than 6 characters</p>

<p v-else-if="password.length > 0">Please enter a longer password</p>

<p v-else>Enter a password</p>

</div>

<script>

new Vue({

el:'#app',

data:{

password:'',

},

});

</script>

**Categories and Sub Categories Displaying Project**

<div id="app">

<ol>

<li v-for="proLanguage in proLanguages">

{{proLanguage.category}}

<ul>

<li v-for="sub in proLanguage.subCat">

{{sub}}

</li>

</ul>

</li>

</ol>

</div>

<script>

new Vue({

el:'#app',

data:{

proLanguages:[

{category: "Javascript", subCat: ["Vue JS","React JS", "Angular2 JS"]},

{category: "Databases", subCat: ["MySQL","PostgreSQL", "MariaDB"]}

]

},

});

</script>

Adding Getter and Setter to Computed properties:

<div id="app">

<p>{{fullName}}</p>

<button v-on:click="getFullName()">Full Name</button>

</div>

<script>

new Vue({

el:'#app',

data:{

firstName:"Bablu",

lastName:"Ahmed",

},

methods:{

getFullName : function () {

this.fullName = "Faruk,Ahmed";

}

},

computed:{

fullName: {

// getter

get: function () {

return this.firstName + ' ' + this.lastName

},

// setter

set: function (newValue) {

var names = newValue.split(',');

this.firstName = names[0];

this.lastName = names[names.length - 1];

alert("Setting new name: "+this.firstName+" "+this.lastName);

}

}

},

});

</script>

**Watchers**: To see the changes the value of data property (like data:{property:value}). If we want to see the changes of a data property then we have to add a watch object. In watcher we have to add the property name as function as follows:

Example 01:

<div id="app">

<input v-model="searchQuery" type="text">

<p>{{searchQuery}}</p>

</div>

<script>

new Vue({

el:'#app',

data:{

searchQuery: '',

},

watch: {

searchQuery :function(query) {

console.log(query);

}

}

});

</script>

Example 02:

<div id="app">

<input v-model="searchQuery" type="text">

<p>{{searchQuery}}</p>

</div>

<script>

new Vue({

el:'#app',

data:{

searchQuery: '',

},

watch: {

searchQuery :function(query) {

console.log(query);

}

}

});

</script>

Example 02:

<div id="app">

<input v-model="searchQuery" type="text">

<p v-if="isSearching">Searching...</p>

<div v-else>

<ol>

<li v-for="result in results">{{result}}</li>

</ol>

</div>

</div>

<script>

new Vue({

el:'#app',

data:{

searchQuery: '',

results: [],

isSearching: false,

},

watch: {

searchQuery :function(query) {

//console.log(query);

this.isSearching = true;

//console.log(this);

let vm = this;

let myQuery = query;

setTimeout(function() {

//console.log(this);

//console.log(vm);

let regex = /^(?!\s)(php|mysql|Javascript)/ig;

let isFound = myQuery.match(regex);

console.log(isFound);

if(isFound === null){

vm.results = [];

}else{

vm.results = ['PHP','MySQL','Javascript'];

}

vm.isSearching = false;

},500);

}

}

});

</script>

**Exercise**: Movie name withe release year and add new move to the move list and watch it with watcher.

<div id="app">

<!-- Movie name withe release year -->

<ol>

<li v-for="movie in formattedMovies">{{ movie }}</li>

</ol>

<!-- Add new move to the move list and watch it with watcher -->

<button v-on:click="addMovie">Add Movie</button>

</div>

<script>

new Vue({

el:'#app',

data:{

movies: [

{name: "Aynabaji", year:"2016"},

{name: "Dhaka Attack", year:"2017"},

{name: "Doob", year:"2018"},

],

},

computed: {

formattedMovies: function () {

return this.movies.map(function (movie) { //will return a new array i.e, ["Aynabaji (2016)", "Dhaka Attack (2017)", "Doob (2018)"]

return movie.name + ' ('+ movie.year +')';

});

}

},

methods:{

addMovie: function() {

this.movies.push({

name: "Pora Mon 2", year: "2018"

});

}

},

watch: {

movies: function(movies) {

var newMovie = movies[movies.length - 1];

alert(newMovie.name + ' from ' + newMovie.year + ' was just added!');

}

}

});

</script>

////Filters are functions. They can use both string interpolation (text transformation) and v-bind directive expressions.

**Example 01:**

<div id="app">

<h1>{{ message | uppercase }}</h1>

</div>

new Vue({

el:'#app',

data:{

message: 'hello world!',

},

filters: {

uppercase: function(value) {

if (!value) {

return '';

}

value = value.toString().toUpperCase();

return value;

}

}

});

</script>

In the above template, message is the parameter of uppercase.

**Example 02**:

Template:

<div id="app">

<h1>{{ message | uppercase(true) }}</h1>

</div>

Vue Instance:

new Vue({

el:'#app',

data:{

message: 'hello world!',

},

filters: {

uppercase: function(value, onlyFirstCharacter) {

if (!value) {

return '';

}

value = value.toString();

if (onlyFirstCharacter) {

return value.charAt(0).toUpperCase() + value.slice(0);

}else{

return value.toUpperCase();

}

}

},

});

</script>

Here, in Template we can see the **message** property is the first argument of uppercase and true is the second.

Example 03: Firstly make capitalized every words and secondly remove spaces form the words.

<div id="app">

<h1 v-bind:title="message | uwfirst">{{ message }}</h1>

<h1>{{ message | uwfirst | removeSpaces }}</h1>

</div>

new Vue({

el:'#app',

data:{

message: 'hello world!',

},

filters: {

uwfirst: function(value) {

if (!value) {

return '';

}

var parts = value.toString().split(' ');

var uppercasedWords = parts.map(function(word) {

return word.charAt(0).toUpperCase() + word.slice(1);

});

return uppercasedWords.join(' ');

},

removeSpaces: function (value) {

if (!value) {

return '';

}

return value.toString().replace(/ /g, '');

}

}

});

</script>

Here, message property is argument both uwfirst and removeSpaces filter.

**Styling inline css:**

Example 01:

//Static

<div id="app">

<div style="width: 200px; height: 200px;" v-bind:style="{ 'background-color': 'red' }"></div>

</div>

//Dynamic

<div id="app">

<div style="width: 200px; height: 200px;" v-bind:style="{ 'background-color': color }"></div>

</div>

<script>

new Vue({

el:'#app',

data:{

color: 'green',

},

});

</script>

Example 02:

<div id="app">

<div style="width: 200px; height: 200px;" v-bind:style="{ 'background-color': color }"></div>

<button v-on:click="changeColor">Change Color</button>

</div>

<script>

new Vue({

el:'#app',

data:{

color: 'green'

},

methods: {

changeColor: function () {

if (this.color == 'green'){

this.color = 'red';

//alert('hi');

}else{

this.color = 'green';

}

}

}

});

</script>

Example 03: We can use style object

<div id="app">

<div v-bind:style="styles"></div>

</div>

<script>

new Vue({

el:'#app',

data:{

color: 'green',

styles: {

'background-color': 'blue',

width: '200px',

height: '200px'

}

}

});

</script>

Example 04: We can also use style object in computed methods

<div id="app">

<div v-bind:style="[styles, moreStyles]"></div>

</div>

<script>

new Vue({

el:'#app',

data:{

color: 'green',

moreStyles: {

'border-radius': '5px'

}

},

computed: {

styles : function () {

var height = 200;

return {

'background-color': 'green',

width: (height/2) + 'px',

height: height + 'px'

}

}

}

});

</script>

//Style with class

Example 01:

<style>

.shape{

width: 150px;

height: 150px;

float: left;

margin: 10px;

}

.circle{

background-color: red;

border-radius: 50%;

}

.square{

background-color: blue;

}

</style>

<div id="app">

<div class="shape" v-for="shape in shapes" v-bind:class="{ circle: shape.isRound, square: !shape.isRound }">

</div>

</div>

<script>

new Vue({

el:'#app',

data:{

shapes: [

{isRound: true},

{isRound: false}

],

}

});

</script>

Example 02:

<style>

.shape{

width: 150px;

height: 150px;

float: left;

margin: 10px;

}

.circle{

background-color: red;

border-radius: 50%;

}

.square{

background-color: blue;

}

.triangle {

border-color: transparent transparent green transparent;

border-style: solid;

border-width: 0px 150px 150px 150px;

height: 0px;

width: 0px;

}

/\* Animations \*/

.shape.animate{

animation-name: stretch;

animation-duration: 1.0s;

animation-timing-function: ease-out;

}

@keyframes stretch {

0%{

transform: scale(.3);

}

100%{

transform: scale(1.0);

}

}

</style>

<div id="app">

<div class="shape" v-for="shape in shapes" v-bind:class="[shape.shape, shape.direction ? shape.direction : '', {animate: shape.animate }]">

</div>

</div>

<script>

new Vue({

el:'#app',

data:{

shapes: [

{ shape: 'circle' },

{ shape: 'square', animate: true},

{ shape: 'triangle', direction:'up' },

{ shape: 'triangle', direction:'right', animate: true},

{ shape: 'triangle', direction:'down' },

{ shape: 'triangle', direction:'left', animate: true}

],

}

});

</script>

In above code, we have used an array of class with if statement and another if statement of object.

**Example 03**:

<style>

.highlighted{

background-color: #ddd;

}

.shape{

width: 150px;

height: 150px;

float: left;

margin: 10px;

background-color: blue;

}

</style>

<div id="app">

<div class="shape" v-bind:style="shapeStyles" v-on:click="changeShape"></div>

<div style="clear: both;"></div>

<h1>Employees</h1>

<table>

<thead>

<tr>

<th>Name</th>

<th>Title</th>

<th>Company</th>

</tr>

</thead>

<tbody>

<tr v-for="(employee, loopIndex) in employees" v-bind:class="{ highlighted: loopIndex % 2 }">

<td>{{employee.name}}</td>

<td>{{employee.title}}</td>

<td>{{companyName}}</td>

</tr>

</tbody>

</table>

</div>

<script>

new Vue({

el:'#app',

data: {

shapeStyles: {},

employees: [

{ name: "Bablu", title: "Engr" },

{ name: "Sa", title: "Engr" }

],

companyName: 'W3 Public'

},

methods: {

changeShape: function () {

for (var i = 0; i < this.employees.length; i++) {

console.log(this.employees[i].name);

}

if (typeof this.shapeStyles['border-radius'] !== 'undefined') {

Vue.set(this.shapeStyles, 'border-radius', undefined);

} else {

Vue.set(this.shapeStyles, 'border-radius', '50%');

}

}

}

});

</script>

//Shorthands for events and bindings

<button v-on:click=”expression”>Click Me</button>

<button @click=”expression”>Click Me</button>

<p v-bind:title=”expression”>Hover Me</p>

<p :title=”expression”>Hover Me</p>

**Events**

//Form submitting prevent

<form v-on:submit.prevent>

//or

<button type=”submit” @click.submit.prevent>

</form>

//Form submitting prevent but when we will click on button with crtl+alt key then it will fire

<button type=”submit” @click.submit.prevent.alt=”function\_name”>

**Components**

Components are Vue instances that are used inside a root Vue instance as a custom HTML element.

**Why are these things needed?**

If we want in a single page all pages of my project will be shown then the problem will arise aside, header, footer, and article will be multiple times and we can easily solve this problem by using components

**Example 01:**

<div class="container">

<div id="app">

<div class="row">

<sidebar></sidebar>

<sidebar></sidebar>

</div>

</div>

</div>

<script>

Vue.component('sidebar',{

template:`

<div class='col-md-4 col-md-offset-4'>

<ul>

<li><a href="#">Home</a></li>

<li><a href="#">About</a></li>

<li><a href="#">Services</a></li>

</ul>

</div>

`

});

new Vue ({

el:'#app',

});

</script>

Since components are reusable Vue instances, they accept the same options as **new Vue**, such as computed, watch, methods, and lifecycle hooks. The only exceptions are a few root-specific options like **el**.

For example: data must be a function and must return an object like this

data: function () {

return {

count: 0

}

}

**Example 02:**

<div class="container">

<div id="app">

<div class="row">

<sidebar></sidebar>

<sidebar></sidebar>

</div>

</div>

</div>

<script>

Vue.component('sidebar',{

template:`

<div class='col-md-4 col-md-offset-4'>

<ul>

<li v-for="navItem in navItems"><a href="#">{{navItem}}</a></li>

</ul>

</div>

`,

data(){

return {

navItems: ['Home','About', 'Services']

}

}

});

new Vue ({

el:'#app',

});

</script>

**Props**

Props are custom attributes we can register on a component. When a value is passed to a prop attribute, it becomes a property on that component instance. By default, any value can be passed to any prop.

**Passing Data to Child Components with Props**

<div class="container">

<div id="app">

<div class="row">

<sidebar title="My journey with Vue"></sidebar>

<sidebar title="Blogging with Vue"></sidebar>

</div>

</div>

</div>

<script>

Vue.component('sidebar',{

props: ['title'],

template:`

<div class='col-md-4 col-md-offset-4'>

<h3>{{ title }}</h3>

</div>

`

});

new Vue ({

el:'#app',

});

</script>

In the above code, <sidebar></sidebar> is called parent and template inside the Vue component is called child and we have registered a title by probs attribute and it becomes a property on that component instance.

**Passing Data from Root to component instance with Props**

<div class="container">

<div id="app">

<div class="row">

<sidebar innumber="value" title="My journey with Vue"></sidebar>

<sidebar :innumber="value" title="Blogging with Vue"></sidebar>

<button class="btn btn-success" @click="increment()">Increment</button>

<p><b>In Root Instance: </b> {{ value }}</p>

</div>

</div>

</div>

<script>

Vue.component('sidebar',{

props: ['title', 'innumber'],

template:`

<div class='col-md-4 col-md-offset-4'>

<h3>{{ title }}</h3>

<p><b>Inside the component instance: </b> {{ innumber }}</p>

</div>

`

});

new Vue ({

el:'#app',

data:{

value: 0

},

methods:{

increment: function () {

this.value +=1;

}

}

});

</script>

**N.B**, If we want to pass variable as data from parent component to child component we have to use v-bind attribute except string value.

**how to emit a value with a click event in Vue JS**

**Example**: 01

<div id="app">

<div class="container">

<div class="col-lg-offset-4 col-lg-4">

<sidebar v-on:increment-btn="increment += $event"></sidebar>

<p>{{ increment }}</p>

</div>

</div>

</div>

<script>

Vue.component('sidebar',{

template:`

<div>

<button class="btn btn-info" v-on:click="$emit('increment-btn', 1)">Increment on click</button>

</div>

`

});

new Vue ({

el:'#app',

data:{

increment:0

}

});

</script>

**Example 02**: if the event handler is a method

<div id="app">

<div class="container">

<div class="col-lg-offset-4 col-lg-4">

<sidebar v-on:increment-btn="incMthod"></sidebar>

<p>{{ increment }}</p>

</div>

</div>

</div>

<script>

Vue.component('sidebar',{

template:`

<div>

<button class="btn btn-info" v-on:click="$emit('increment-btn', 1)">Increment on click</button>

</div>

`

});

new Vue ({

el:'#app',

data:{

increment:0

},

methods:{

incMthod: function (ValFromComponent) {

this.increment+=ValFromComponent;

}

}

});

</script>

**N.B,** incMthod($event) or may be blank

**Example 03**: custom input event

<div id="app">

<div class="container">

<div class="col-lg-offset-4 col-lg-4">

<sidebar v-on:custom-input="searchText = $event"></sidebar>

<p>{{ searchText }}</p>

<label for="">Root Input Box</label>

<input v-model="searchText">

</div>

</div>

</div>

<script>

Vue.component('sidebar',{

template:`

<div>

<label for="">Component Input Box</label>

<input v-on:input="$emit('custom-input', $event.target.value)">

</div>

`

});

new Vue ({

el:'#app',

data:{

searchText:''

}

});

</script>

**Example 04**: Using v-model on Components does the same as example 03

<div id="app">

<div class="container">

<div class="col-lg-offset-4 col-lg-4">

<sidebar v-model="searchText"></sidebar>

<p>{{ searchText }}</p>

<label for="">Root Input Box</label>

<input v-model="searchText">

</div>

</div>

</div>

<script>

Vue.component('sidebar',{

template:`

<div>

<label for="">Component Input Box</label>

<input v-on:input="$emit('input',searchTextCom)" v-model="searchTextCom">

</div>

`,

data(){

return{

searchTextCom:''

}

}

});

new Vue ({

el:'#app',

data:{

searchText:''

}

});

</script>

**Example 05**: Using methods on Components

<div id="app">

<div class="container">

<div class="col-lg-offset-4 col-lg-4">

<sidebar v-model="searchText"></sidebar>

<p>{{ searchText }}</p>

<label for="">Root Input Box</label>

<input v-model="searchText">

</div>

</div>

</div>

<script>

Vue.component('sidebar',{

template:`

<div>

<label for="">Component Input Box</label>

<input v-on:input="changeText" v-model="searchTextCom">

</div>

`,

data(){

return{

searchTextCom:''

}

},

methods:{

changeText: function(){

this.$emit('input',this.searchTextCom);

}

}

});

new Vue ({

el:'#app',

data:{

searchText:''

}

});

</script>

**Slots and Named Slots**

Example 01:

<div id="app">

<div class="container">

<div class="col-lg-offset-4 col-lg-4">

<btn><slot></slot></btn>

<modal>

<template slot="modal\_header">

<button type="button" class="close" data-dismiss="modal">&times;</button>

<h4 class="modal-title">Modal Header</h4>

</template>

<template slot="modal\_body">

<p>Modal Body</p>

</template>

</modal>

</div>

</div>

</div>

Vue.component('btn',{

template:`

<button type="button" class="btn btn-info btn-lg">Open Modal</button>

`

});

Vue.component('modal',{

template:`

<div id="myModal" class="modalf" role="dialog">

<div class="modal-dialog">

<!-- Modal content-->

<div class="modal-content">

<div class="modal-header">

<slot name="modal\_header"></slot>

</div>

<div class="modal-body">

<slot name="modal\_body"></slot>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-default" data-dismiss="modal">Close</button>

</div>

</div>

</div>

</div>

`

});

new Vue ({

el:'#app',

data:{

searchText:''

}

});

Example 02: slot and named slot with modal hide/show

<div id="app">

<div class="container">

<div class="col-lg-offset-4 col-lg-4">

<btn @modal-btn-clicked="modalLaunched"><slot></slot></btn>

<modal :class="activeModal" @inner-close-btn="closeModal">

<template slot="modal\_header">

<button type="button" @click="closeModal" class="close">&times;</button>

<h4 class="modal-title">Modal Header</h4>

</template>

<template slot="modal\_body">

<p>Modal Body</p>

</template>

</modal>

</div>

</div>

</div>

Vue.component('btn',{

template:`

<button @click="launchModal" type="button" class="btn btn-info btn-lg">Open Modal</button>

`,

methods:{

launchModal: function(){

this.$emit('modal-btn-clicked'); //custom event

}

}

});

Vue.component('modal',{

template:`

<div id="myModal" :class="activeModal" role="dialog">

<div class="modal-dialog">

<!-- Modal content-->

<div class="modal-content">

<div class="modal-header">

<slot name="modal\_header"></slot>

</div>

<div class="modal-body">

<slot name="modal\_body"></slot>

</div>

<div class="modal-footer">

<button type="button" @click="innerCloseBtn" class="btn btn-default" data-dismiss="modal">Close</button>

</div>

</div>

</div>

</div>

`,

props:['activeModal'],

methods:{

innerCloseBtn: function(){

this.$emit('inner-close-btn'); //custom event

}

}

});

new Vue ({

el:'#app',

data:{

activeModal:'modal'

},

methods:{

modalLaunched: function(){

this.activeModal='modalopen';

},

closeModal: function(){

this.activeModal='modal';

}

}

});

**Dynamic Components**: Below two components are exactly the same effect.

<component **is**="manage-posts"></component> and <manage-posts></manage-posts>

But using the component tag we can make dynamic with the **is** special attribute dynamically bind like this:

<!-- render the currently active component/page here -->

<component **:is**="currentView"></component>

new Vue({

el: '#app',

data: {

currentView: 'manage-posts'

}

});

**For example 01**: Let’s see first static

<div id="app">

<div class="container">

<!-- render the currently active component/page here -->

<component is="manage-posts"></component>

</div>

</div>

<template id="manage-template">

<div>

<h1>Manage Posts</h1>

<div class="list-group">

<a v-for="post in posts" href="#" class="list-group-item clearfix">

{{ post }}

<span class="pull-right">

<button class="btn btn-xs btn-info">

<span class="glyphicon glyphicon-edit"></span>

</button>

<button class="btn btn-xs btn-warning">

<span class="glyphicon glyphicon-trash"></span>

</button>

</span>

</a>

</div>

</div>

</template>

new Vue({

el: '#app'

});

**Example 02**: dynamically bind **is** attribute

<div id="app">

<div class="container">

<br>

<a href="#" @click="currentView='manage-posts'">Manage Posts</a> |

<a href="#" @click="currentView='create-post'">Create Post</a>

<br>

<!-- render the currently active component/page here -->

<component :is="currentView"></component>

</div>

</div>

<template id="manage-template">

<div>

<h1>Manage Posts</h1>

<div class="list-group">

<a v-for="post in posts" href="#" class="list-group-item clearfix">

{{ post }}

<span class="pull-right">

<button class="btn btn-xs btn-info">

<span class="glyphicon glyphicon-edit"></span>

</button>

<button class="btn btn-xs btn-warning">

<span class="glyphicon glyphicon-trash"></span>

</button>

</span>

</a>

</div>

</div>

</template>

<template id="create-template">

<div>

<h1>Create Post</h1>

<form class="form-horizontal">

<div class="form-group">

<label class="col-sm-2 control-label">Post title</label>

<div class="col-sm-10">

<input type="text" class="form-control" placeholder="Post title">

</div>

</div>

<div class="form-group">

<label class="col-sm-2 control-label">Post body</label>

<div class="col-sm-10">

<textarea class="form-control" rows="5"></textarea>

</div>

</div>

<div class="form-group">

<div class="col-sm-offset-2 col-sm-10">

<button type="submit" class="btn btn-primary">Create</button>

</div>

</div>

</form>

</div>

</template>

Vue.component('manage-posts', {

template: '#manage-template',

data: function() {

return {

posts: [

'This is my first post about Vue JS',

'This is my Second post about Vue JS',

'This is my third post about Vue JS'

]

}

}

});

Vue.component('create-post', {

template: '#create-template'

});

new Vue({

el: '#app',

data: {

currentView: 'manage-posts'

},

watch:{

currentView: function(query){

console.log(query);

}

}

});