Vue JS 2.0

# Vue Instance

Composed by:

1. el: String selector to connect with DOM
2. data: Object containing stored data values
3. methods: Object containing functions that execute code whenever render.
4. computed: Object containing functions that execute code when reactive dependency is updated.
5. watch: Object containing functions that execute code upon data change
6. mounted: Function for executing code after rendering

el: '#app',  
 data: {  
 message: 'Hello Vue!',  
 counter: 0  
 },  
 methods: {  
 changeTitle: **function**(){  
 **this**.message = "hola";  
 },  
 changeTitleAndNotify: **function**(){  
 **this**.counter++;  
 **this**.message = "hola";  
 }  
 },  
 computed: {  
 titleIsChanged: **function**(){  
 console.log("Message has been changed " + counter + " times")  
 }  
 }  
})

# Declarative rendering

Rendering data to the DOM using template syntax:

## HTML

<div id="app">

<p>{{ message }}</p>

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

</div>

**new** Vue({

el: '#app',  
 data: {  
 message: 'Hello Vue!'  
 },  
 methods: {  
 getMessage: **function** () {  
 **return** 'Hello Vue!'  
 }  
 }

});

# Directives

Directives are used to indicate they are special attributes provided by Vue.

## [v-text](https://vuejs.org/v2/api/#v-text) (print text)

Updates the element’s textContent. If you need to update the part of textContent, you should use {{ Mustache }} interpolations.

<span v-text="msg"></span>

<!-- same as -->

<span>{{msg}}</span>

## [v-html](https://vuejs.org/v2/api/#v-html) (escape HTML)

Updates the element’s innerHTML

## [v-show](https://vuejs.org/v2/api/#v-show) (display none if false)

Toggle’s the element’s display CSS property based on the truthy-ness of the expression value.

## [v-if](https://vuejs.org/v2/api/#v-if) (render if true)

Conditionally render the element based on the truthy-ness of the expression value. The element and its contained directives / components are destroyed and re-constructed during toggles.

|  |
| --- |
| <div id="app-3">  <p v-if="seen">Now you see me</p>  </div> |

|  |
| --- |
| var app3 = new Vue({  el: '#app-3',  data: {  seen: true  }  }) |

## [v-else](https://vuejs.org/v2/api/#v-else) / v-else-if

enote the “else block” for v-if or a v-if/v-else-if chain.

## [v-for](https://vuejs.org/v2/api/#v-for) (Loop)

v-for directive can be used for displaying a list of items using the data from an Array:

<div id="loop">  
 <ol>  
 <li v-for="(person, index) in persons">  
 <p>{{ index }}</p>  
 <p v-for="(val, key, i) in person">{{ i }} - {{ key }}: {{ val }}</p>  
 </li>  
 </ol>  
 <ul>  
 <li v-for="n in 10">  
 {{ n }}  
 </li>  
 </ul>  
</div>  
<script>  
 **var** app4 = **new** Vue({  
 el: '#loop',  
 data: {  
 persons: [  
 {name: 'Juan', age: 15},  
 {name: 'Antonio', age: 30}  
 ]  
 }  
 })  
</script>

* Assigning key to every item in the list we could avoid errors when filtering or sorting list:

…

<ol>  
 <li v-for="(person, index) in persons" ***:key="person.nif"***>  
 <p>{{ index }}</p>  
 <p v-for="(val, key, i) in person">{{ i }} - {{ key }}: {{ val }}</p>  
 </li>  
</ol>

…

## [v-on](https://vuejs.org/v2/api/#v-on) [@] (one way binding)

To let users interact with your app, we can use the v-on directive to attach event listeners that invoke methods on our Vue instances:

|  |
| --- |
| <div id="app-5">  <p>{{ message }}</p>  <button v-on:click="reverseMessage">Reverse Message</button>  </div> |

|  |
| --- |
| var app5 = new Vue({  el: '#app-5',  data: {  message: 'Hello Vue.js!'  },  methods: {  reverseMessage: function () {  this.message = this.message.split('').reverse().join('')  }  }  }) |

### Modifiers

* .stop - call event.stopPropagation().
* .prevent - call event.preventDefault().
* .capture - add event listener in capture mode.
* .self - only trigger handler if event was dispatched from this element.
* .{keyCode | keyAlias} - only trigger handler on certain keys.
* .native - listen for a native event on the root element of component.
* .once - trigger handler at most once.
* .left - (2.2.0+) only trigger handler for left button mouse events.
* .right - (2.2.0+) only trigger handler for right button mouse events.
* .middle - (2.2.0+) only trigger handler for middle button mouse events.
* .passive - (2.3.0+) attaches a DOM event with { passive: true }.

### Argument $event - receiving event as argument

For fetching the event as argument we must use protected name “$event”:

|  |
| --- |
| <div id="app-5">  <p v-on:mousemove="counter()">Reverse Message</p>{{count}} –  <p v-on:mousemove="stop($event)">Reverse Message</p>{{count}}  </div> |

|  |
| --- |
| var app5 = new Vue({  el: '#app-5',  data: {  count: 1  },  methods: {  counter: function () {  this.count += 1;  }  Stop: function(event){  Event.stopPropagation();  }  }  }) |

## [v-bind](https://vuejs.org/v2/api/#v-bind) [:] (update value)

Keep and update value.

<div id="app-2">

<span v-bind:title="message">

Hover your mouse over me for a few seconds

to see my dynamically bound title!

</span>

</div>

var app2 = new Vue({

el: '#app-2',

data: {

message: 'You loaded this page on ' + new Date()

}})

Message is bind with Vue so by running app2.message = ‘Hola’ attribute title will change.

### binding an object of attributes

<div v-bind="{ id: someProp, 'other-attr': otherProp }"></div>

## [v-model](https://vuejs.org/v2/api/#v-model) (two ways binding)

v-model directive that makes two-way binding between form input and app state a breeze:

|  |
| --- |
| <div id="app-6">  <p>{{ message }}</p>  <input v-model="message">  </div> |

var app6 = new Vue({

el: '#app-6',

data: {

message: 'Hello Vue!'

}

})

### [.lazy](https://vuejs.org/v2/guide/forms.html" \l "lazy)

By default, v-model syncs the input with the data after each input event (with the exception of IME composition as [**stated above**](https://vuejs.org/v2/guide/forms.html#vmodel-ime-tip)). You can add the lazy modifier to instead sync after change events:

|  |
| --- |
| <!-- synced after "change" instead of "input" -->  <input v-model.lazy="msg" > |

### .number

If you want user input to be automatically typecast as a number, you can add the number modifier to your v-model managed inputs:

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

This is often useful, because even with type="number", the value of HTML input elements always returns a string.

### .trim

If you want user input to be trimmed automatically, you can add the trim modifier to your v-model managed inputs:

## [v-pre](https://vuejs.org/v2/api/#v-pre) (skip compilation)

## [v-cloak](https://vuejs.org/v2/api/#v-cloak) (hide un-compiled bindings)

This directive will remain on the element until the associated Vue instance finishes compilation. Combined with CSS rules such as [v-cloak] { display: none }, this directive can be used to hide un-compiled mustache bindings until the Vue instance is ready.

## [v-once](https://vuejs.org/v2/api/#v-once) (compile once only )

|  |
| --- |
| Initialize (v-once) For rendering the first value and don’t changing any more, use v-once. In this example first paragraph shows Hello and last shows Adios.  <div id="app-6">  <p>{{ message }}</p>  <p>{{ setMessage() }}</p> </div> <script> **var** app6 = **new** Vue({  el: '#app-6',  data: {  message: 'Hello Vue!'  },  methods: {  setMessage: **function**(){  **this**.message = 'Adios!'  }  } }); |

# AJAX Loading data

<script src="https://cdn.jsdelivr.net/npm/vue-resource@1.3.4"></script>

**this**.$http.get(**this**.controller).then(**function** (response) {  
 // Response to JSON  
 **try** {  
 **var** jRes = JSON.parse(response);  
  
 // Check if error  
 **if** (!jRes.error) {  
  
 // If validate then update  
 **this**.update(jRes);  
 } **else** {  
 console.error("JSON Response", jRes.errorMsg, jRes.errorTrace);  
 }  
 } **catch** (err) {  
 console.error('loading data', ctxt, response, err);  
 } **finally** {  
 **this**.loading = **false**;  
 }  
}, **function** (err) {  
 console.error("POST controller", ctxt, err);  
  
 // Hide loading and show results  
 **this**.loading = **false**;  
});

# Filters

## Reverse text filter

<span v-text="message | reverse"></span>

<script>  
Vue.filter('reverse', **function** (value) {  
**return** value.split('').reverse().join('')  
})  
</script>

# Multiple instances

## Interact between two VUE JS instances

<div id="app1">  
 <h1>{{ title }}</h1>  
 <button @click="changeTitle2">Change Title 2</button>  
</div>  
<div id="app2">  
 <h1>{{ title }}</h1>  
 <button @click="changeTitle1">Change Title 1</button>  
</div>  
  
<script>  
 **var** vm1 = **new** Vue({  
 el: '#app1',  
 data: {  
 title: "Hola"  
 },  
 methods: {  
 changeTitle2: **function**(){  
 vm1.title = "Changed by vm1";  
 }  
 }  
 });  
  
 **var** vm2 = **new** Vue({  
 el: '#app2',  
 data: {  
 title: "Adios"  
 },  
 methods: {  
 changeTitle1: **function**(){  
 vm1.title = "Changed by vm2";  
 }  
 }  
 });  
</script>

## Call VUE JS instance from outside

<div>  
 <h1 id="app1">{{ title }}</h1>  
 <button id="btn">Change Title 2</button>  
</div>  
  
<script>  
 **var** vm1 = **new** Vue({  
 el: '#app1',  
 data: {  
 title: "Hola"  
 }  
 });  
  
 document.getElementById("btn").onclick(**new function**() {  
 vm1.title = "Changed from outside";  
 });  
</script>

# Special methods

## $refs

Accessing directly de DOM HTML Objects, by $refs. It do not change template

<div>  
 <h1 id="app1">{{ title }}</h1>  
 <button id="btn" ref="myButton">Change</button>  
</div>  
  
<script>  
 **var** vm1 = **new** Vue({  
 el: '#app1',  
 data: {  
 title: "Hola"  
 },  
 methods: {  
 showBtn: **function**(){  
 console.log(**this**.$refs.myButton);  
 }  
 }  
 });  
  
 document.getElementById("btn").onclick(**new function**() {  
 vm1.title = "Changed from outside";  
 vm1.showBtn;  
 });  
</script>

## $mount (=el)

References to the HTML element where VUE instance is rendered

<div>  
 <h1 id="app1">{{ title }}</h1>  
 <button id="btn" ref="myButton" @click="showBtn">Change</button>  
</div>  
  
<script>  
 **var** vm1 = **new** Vue({  
 data: {  
 title: "Hola"  
 },  
 methods: {  
 showBtn: **function**(){  
 console.log(**this**.$refs.myButton);  
 }  
 }  
 });  
  
 vm1.$mount("#app1");  
</script>

# Component

Allows to create reusable VUE JS elements.

<div>  
 <hello></hello>  
 <hello></hello>  
</div>  
  
<script>  
 Vue.component("hello",{  
 template: "<h1>Hello</h1>"  
 });  
</script>

# Properties

Values can be set by outside the current component

<div id="app">  
 <test name="hola"></test>  
</div>  
  
<script>  
Vue.Component("test", {  
 props: ["name"],  
 template: "<h1>{{ name }}</h1>"  
});  
  
**new** Vue ({  
 el: 'app'   
});  
</script>

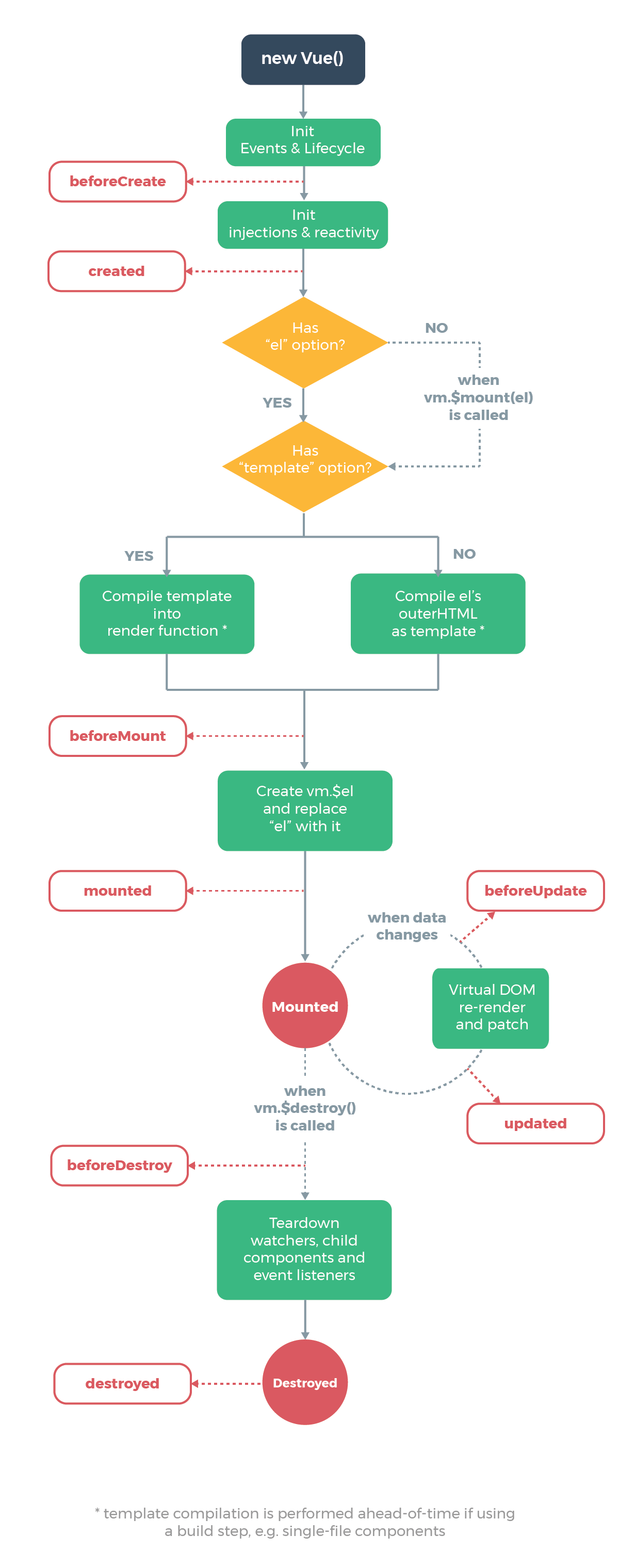
## Type of properties

Properties are String by default. They could be set to different type.

<div id="app">  
 <test name="hola"></test>  
</div>  
  
<script>  
Vue.Component("test", {  
 props: {  
 name:{  
 type:String (Array, Object, Number…)  
 }

},  
 template: "<h1>{{ name }}</h1>"  
});  
  
**new** Vue ({  
 el: 'app'  
});  
</script>

# Lifecycle



## Methods

* beforeCreate
* created
* beforeMount
* mounted
* beforeUpdate
* updated
* beforeDestroy
* destroyed
* activated (keep-alive)
* deactivated (keep-alive)

<div id="app">  
 <h1>{{ title }}</h1>  
 <button @click="title = 'Changed'"></button>  
 <button @click="destroy"></button>  
</div>  
  
<script>  
 **var** vm = **new** Vue({  
 el: "#app",  
 data: {  
 title: "Hola"  
 },  
 beforeCreate: **function**(){  
 console.log("beforeCreate");  
 },  
 created: **function**(){  
 console.log("created");  
 },  
 beforeMount: **function**(){  
 console.log("beforeMount");  
 },  
 mounted: **function**(){  
 console.log("mounted");  
 },  
 beforeUpdate: **function**(){  
 console.log("beforeUpdate");  
 },  
 updated: **function**(){  
 console.log("updated");  
 },  
 beforeDestroy: **function**(){  
 console.log("beforeDestroy");  
 },  
 destroyed: **function**(){  
 console.log("destroyed");  
 },  
 methods: {  
 destroy: **function**(){  
 **this**.$destroy();  
 }  
 }  
 });  
</script>

# Slots

The content is passing between component tag and render with <slot>. It is useful for the case we render the same content changing html and styles. Content is controlled by the main component and styles by the child.

App.vue

<template>

<app-quote>  
 <h2 slot=”title”>The Quote</h2>  
 <p slot=”content”>A wonderful Quote</p>  
</app-quote>  
</template>

Quote.vue

<template>  
 <div>

<slot name=”title”></slot>

<hr>  
 <slot name=”content”></slot>  
 </div>  
</template>

# Dynamic Component

Load a component using tag “component” identified by “is” attribute

<template>  
 <div >  
 <button @click="selectedComponent = 'app-quote'">Quote</button>  
 <button @click="selectedComponent = 'app-author'">Author</button>  
 <button @click="selectedComponent = 'app-new'">New</button>  
 <component :is="selectedComponent"></component>  
 </div>  
</template>

<script>  
 **import** Quote **from** './components/Quote.vue';  
 **import** Author **from** './components/Author.vue';  
 **import** New **from** './components/New.vue';  
  
 **export default** {  
 data: **function**(){  
 **return** {  
 selectedComponent: 'app-quote'  
 }  
 },  
 components: {  
 "app-quote": Quote,  
 "app-author": Author,  
 "app-new": New  
 }  
  
 }  
</script>

## Reuse

By default when changing dynamic component create and destroy component. We can overwrite this behavior by surrounding “component” tag by “keep-alive”

<template>  
 <div >  
 <button @click="selectedComponent = 'app-quote'">Quote</button>  
 <button @click="selectedComponent = 'app-author'">Author</button>  
 <button @click="selectedComponent = 'app-new'">New</button>

<keep-alive>  
 <component :is="selectedComponent"></component>

</keep-alive>  
 </div>  
</template>

<script>  
 **import** Quote **from** './components/Quote.vue';  
 **import** Author **from** './components/Author.vue';  
 **import** New **from** './components/New.vue';  
  
 **export default** {  
 data: **function**(){  
 **return** {  
 selectedComponent: 'app-quote'  
 }  
 },  
 components: {  
 "app-quote": Quote,  
 "app-author": Author,  
 "app-new": New  
 }  
  
 }  
</script>

# Form

## Basic example

<template>  
 <div class="container">  
 <form>  
 <div class="row">  
 <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">  
 <h1>File a Complaint</h1>  
 <hr>  
 <div class="form-group">  
 <label for="email">Mail</label>  
 <input  
 type="text"  
 id="email"  
 class="form-control"  
 v-model="user.email">  
 </div>  
 <div class="form-group">  
 <label for="password">Password</label>  
 <input  
 type="password"  
 id="password"  
 class="form-control"  
 v-model="user.password">  
 </div>  
 <div class="form-group">  
 <label for="age">Age</label>  
 <input  
 type="number"  
 id="age"  
 class="form-control"  
 v-model="user.age">  
 </div>  
  
 </div>  
 </div>  
 <div class="row">  
 <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3 form-group">  
 <label for="message">Message</label><br>  
 <!-- Interpolation between <textarea>{{ test }}</textarea> doesn't work!-->  
 <textarea  
 id="message"  
 rows="5"  
 class="form-control"  
 v-model="message"></textarea>  
 </div>  
 </div>  
 <div class="row">  
 <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">  
 <div class="form-group">  
 <label for="sendmail">  
 <input  
 type="checkbox"  
 id="sendmail"  
 value="SendMail"  
 v-model="sendMail"> Send Mail  
 </label>  
 <label for="sendInfomail">  
 <input  
 type="checkbox"  
 id="sendInfomail"  
 value="SendInfoMail"  
 v-model="sendMail"> Send Infomail  
 </label>  
 </div>  
  
 </div>  
 </div>  
 <div class="row">  
 <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3 form-group">  
 <label for="male">  
 <input  
 type="radio"  
 id="male"  
 value="Male"  
 v-model="gender"> Male  
 </label>  
 <label for="female">  
 <input  
 type="radio"  
 id="female"  
 value="Female"  
 v-model="gender"> Female  
 </label>  
 </div>  
 </div>  
 <div class="row">  
 <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3 from-group">  
 <label for="priority">Priority</label>  
 <select  
 id="priority"  
 class="form-control"  
 v-model="selectedPriority">  
 <option  
 v-for="priority in priorities"  
 :id="priority">{{ priority }}  
 </option>  
 </select>  
 </div>  
 </div>  
 <hr>  
 <div class="row">  
 <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">  
 <button  
 class="btn btn-primary">Submit!  
 </button>  
 </div>  
 </div>  
 </form>  
 <hr>  
 <div class="row">  
 <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">  
 <div class="panel panel-default">  
 <div class="panel-heading">  
 <h4>Your Data</h4>  
 </div>  
 <div class="panel-body">  
 <p>Mail: {{ user.email }}</p>  
 <p>Password: {{ user.password }}</p>  
 <p>Age: {{ user.age }}</p>  
 <p style="white-space: pre">Message: {{ message }}</p>  
 <p><strong>Send Mail?</strong></p>  
 <ul>  
 <li v-for="item in sendMail">{{ item }}</li>  
 </ul>  
 <p>Gender: {{ gender }}</p>  
 <p>Priority:</p>  
 <p>Switched:</p>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
</template>  
  
<script>  
 **export default** {  
 data: **function** () {  
 **return** {  
 user: {  
 email: '',  
 password: '',  
 age: 20  
 },  
 message: '',  
 sendMail: [],  
 gender: 'Male',  
 priorities: [  
 'Low',  
 'Medium',  
 'High',  
 ],  
 selectedPriority: 'Medium'  
 }  
 }  
 }  
</script>

## Custom input

Directive v-model is equal to v-on and v-bind working together. We can create a component which use value property and emit input event:

<!--v-model-->  
<input type="text" id="email" class="form-control" v-model="user.email">  
<!--v-on and v-bind-->  
<input type="text" id="email2" class="form-control"   
 :value="user.email" @input="user.email = $event.target.value">

Parent component

</template>

<switch v-model="switch"></switch>  
</template>  
  
<script>  
 **export default** {  
 data: **function** () {  
 **return** {  
 **switch**: **true** }  
 }  
 }  
</script>

Child component

<template>  
 <div>  
 <div id="on" @click="changed(true)"></div>  
 <div id="off" @click="changed(false)"></div>  
 </div>  
</template>  
  
<script>  
 **export default** {  
 props: ['value'],  
 methods:{  
 changed(value){  
 **this**.$emit('input', value)  
 }  
 }  
 }  
</script>

# VUE Application

## VUE CLI

Ref: <https://github.com/vuejs/vue-cli>

1. Require NodeJS
2. $ sudo npm install –g vue-cli
3. $ vue init {{ template: webpack-simple }} {{ projectName: vue-cli}}
4. $ cd {{ project folder: vue-cli }}
5. $ npm install
6. $ npm run dev

* En caso de que no se pueda accede por localhost (si la máquina de desarrollo es virtual) añadir en package.json

"dev": "cross-env NODE\_ENV=development webpack-dev-server --open --hot"

## Styles in components

Styles are applied locally using scrope attribute

App.vue

<style scoped>  
 div{  
 border: 1px solid red;  
 }  
</style>

## Communication between components

### Properties to show

Array of Values can be set by outside the current component

1. Parent component

<template>

<button @click="changeName">Change my name</button>  
<hr>  
<div class="col-xs-12 col-sm-6">  
 <app-user-detail :name="name"></app-user-detail>  
</div>

</template>  
  
<script>  
 **import** UserDetail **from** './UserDetail.vue';   
  
 **export default** {  
 data: **function** () {  
 **return** {  
 name: 'Max'  
 }  
 },  
 components: {  
 appUserDetail: UserDetail   
 },  
 methods:{  
 changeName(){  
 **this**.name = 'Anna'  
 }  
 }  
 }  
</script>

1. Child component

<template>  
 <div class="component">  
 <p>User name: {{ name }}</p>  
 </div>  
</template>  
  
<script>  
 **export default** {  
 props: ['name']  
 }  
</script>

### Events to set from child

1. Parent

<template>  
 <p>I'm an awesome User! ({{name}})</p>  
 <button @click="changeName">Change my name</button>  
 <hr>  
 <div class="col-xs-12 col-sm-6">  
 <app-user-detail :name="name" @nameWasReset="nameWasResetByEvent($event)"></app-user-detail>  
 </div>  
</template>  
  
<script>  
 **import** UserDetail **from** './UserDetail.vue';   
  
 **export default** {  
 data: **function** () {  
 **return** {  
 name: 'Max'  
 }  
 },  
 components: {  
 appUserDetail: UserDetail   
 },  
 methods:{  
 changeName(){  
 **this**.name = 'Anna'  
 },   
 nameWasResetByEvent(eventName){  
 **this**.name = eventName;  
 }  
 },  
  
 }  
</script>

1. Child

<template>  
 <div class="component">  
 <p>User name: {{ switchName() }}</p>  
 <button @click="resetName">resetName</button>  
 </div>  
</template>  
  
<script>  
 **export default** {  
 props: {  
 name:{  
 type:String  
 }  
 },  
 methods: {  
 switchName(){  
 **return this**.name.split("").reverse().join("");  
 },  
 resetName(){  
 **this**.name="Max";  
 **this**.$emit('nameWasReset', **this**.name);  
 }  
 }  
 }  
</script>

### Callback function as property

Set propery as Function type and execute it in the child component

1. Parent

<template>  
 <div class="component">  
 <p>I'm an awesome User! ({{name}})</p>  
 <app-user-detail

:name="name"  
 @nameWasReset="nameWasResetByEvent($event)"

:resetFn="resetName"></app-user-detail>  
 </div>  
</template>  
  
<script>  
 **import** UserDetail **from** './UserDetail.vue';  
  
 **export default** {  
 data: **function** () {  
 **return** {  
 name: 'Max'  
 }  
 },  
 components: {  
 appUserDetail: UserDetail  
 }

}  
</script>

1. Child

<template>  
 <div class="component">  
 <h3>You may view the User Details here</h3>  
 <p>Many Details</p>  
 <p>User name: {{ switchName() }}</p>  
 <button @click="resetFn()">resetName</button>  
 </div>  
</template>  
  
<script>  
 **export default** {  
 props: {  
 name:{  
 type: String  
 },  
 resetFn:{  
 type: Function  
 }  
 }  
 }  
</script>

### Event Bus

Create a new Vue instance in main.js which will manage events.

1. Main.js

**[…]**

**export const** eventBus = **new** Vue();  
  
**new** Vue({

[…]

1. Child listening event

<template>  
 <div class="component">  
 <p>User age: {{ age }}</p>  
 </div>  
</template>  
  
<script>  
 **import** {eventBus} **from** '../main.js'  
  
 **export default** {  
 props: {  
 age:{  
 type: Number  
 }  
 },  
 created(){  
 eventBus.$on('ageWasEdited', (age) => {  
 **this**.age = age;  
 })  
 }  
 }  
</script>

1. Child throwing event

<template>  
 <div class="component">

<p>Age: {{age}}</p>  
 <button @click="editAge">Edit age</button>  
 </div>  
</template>  
  
<script>  
 **import** {eventBus} **from** '../main.js'  
  
 **export default** {  
 props:{  
 age:{  
 type: Number  
 }  
 },  
 methods:{  
 editAge(){  
 **this**.age = 30;   
 eventBus.$emit('ageWasEdited', **this**.age);  
 }  
 }  
  
 }  
</script>

# Vue Resource

1. $ npm install vue-resource –save
2. **import** Vue **from** 'vue'  
   **import** VueResource **from** 'vue-resource';  
   **import** App **from** './App.vue'  
     
   Vue.use(VueResource);