Fuente: tutorialspoint.com // platzi.com

¿Qué es Vue.JS?

Vue.JS es un framework JavaScript progresivo de código abierto que se utiliza para desarrollar interfaces web interactivas. Es uno de los marcos famosos que se utilizan para simplificar el desarrollo web. Vue.JS se centra en la capa de vista. Se puede integrar fácilmente en grandes proyectos para el desarrollo front-end sin ningún problema.

La instalación de Vue.JS es muy fácil. Cualquier desarrollador puede comprender y crear interfaces web interactivas fácilmente en cuestión de tiempo. Vue.JS fue creado por Evan You, un ex empleado de Google. La primera versión de Vue.JS se lanzó en febrero de 2014.

Virtual DOM

Vue.JS hace uso del DOM virtual, que también es utilizado por otros frameworks como React, Ember, etc. Los cambios no se realizan en el DOM, sino que se crea una réplica del DOM que está presente en forma de estructuras de datos JavaScript. . Siempre que se deben realizar cambios, se realizan en las estructuras de datos de JavaScript y esta última se compara con la estructura de datos original. Luego, los cambios finales se actualizan al DOM real, que el usuario verá cambiar. Esto es bueno en términos de optimización, es menos costoso y los cambios se pueden realizar a un ritmo más rápido.

Data Binding

La función de data binding ayuda a manipular o asignar valores a atributos HTML, cambiar el estilo, asignar clases con la ayuda de la directiva de enlace llamada **v-bind** disponible en Vue.JS.

Components

Los componentes son una de las características importantes de Vue.JS que ayudan a crear elementos personalizados, que se pueden reutilizar en HTML.

Event Handling

v-on es el atributo agregado a los elementos DOM para escuchar los eventos en Vue.JS.

Computed Properties

Esta es una de las características más importantes de Vue.JS. Ayuda a escuchar los cambios realizados en los elementos de la interfaz de usuario y realiza los cálculos necesarios. No hay necesidad de codificación adicional para este fin.

Templates

Vue.JS proporciona plantillas basadas en HTML que unen el DOM con los datos de la instancia de Vue. Vue compila las plantillas en funciones virtuales DOM Render. Podemos hacer uso de la plantilla de las funciones de render y para hacerlo tenemos que reemplazar la plantilla con la función de render.

Directives

Vue.JS tiene directivas integradas como **v-if, v-else, v-show, v-on, v-bind** y **v-model**, que se utilizan para realizar varias acciones en la interfaz.

Watchers

Los Watchers se aplican a los datos que cambian. Por ejemplo, elementos de input en formularios. Aquí, no tenemos que agregar ningún evento adicional. Los Watchers se encargan de manejar cualquier cambio de datos haciendo que el código sea simple y rápido.

Liviano

Vue.JS es muy ligero y el rendimiento también es muy rápido.

Hay muchas formas de instalar Vue.JS. Algunas de las formas de cómo realizar la instalación se comentan a continuación.

Usando la etiqueta <script> en el documento HTML.

Descargar desde https://vuejs.org/v2/guide/installation.html la versión necesaria: La versión de desarrollo no está minificada, mientras que la versión de producción si.

```
<html>
  <head>
    <script type = "text/javascript" src = "vue.min.js"></script>
  </head>
  <body></body>
</html>
```

Usando CDN

```HTML
<!-- development version, includes helpful console warnings -->
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

or:

```HTML
<!-- production version, optimized for size and speed -->
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
```

Ejemplo

```
<html>
  <head>
    <title>VueJs Introduction</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "intro">
      <h1>{{ message }}</h1>
    </div>
    <script type = "text/javascript">
      const vue_det = new Vue({
         el: '#intro',
         data: {
            message: 'My first VueJS Task'
         }
      });
    </script>
  </body>
</html>
```

Output

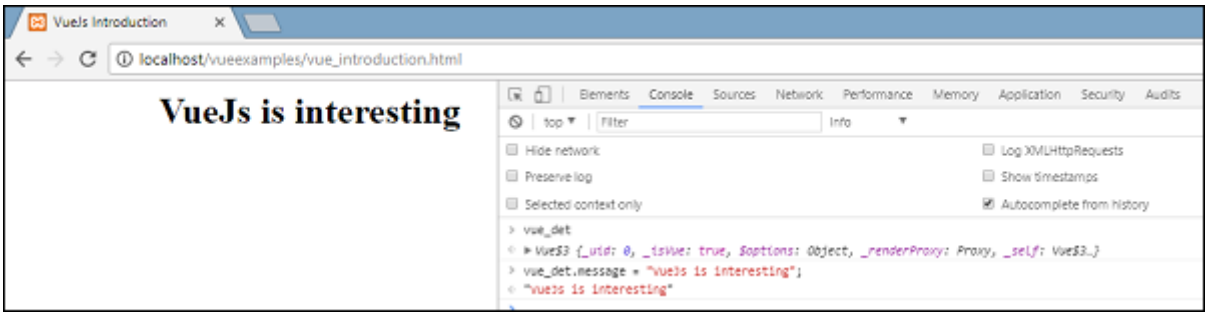
Para este ejemplo se incluyó el import del archivo js de Vue.JS dentro de la etiqueta <head>. El div presente tiene un mensaje en una interpolación {{}}. Esto interactúa con Vue.JS y muestra los datos en el navegador. Para mostrar el valor del mensaje en el DOM, se crea una instancia de Vue.JS.

```
<div id = "intro">
  <h1>{{ message }}</h1>
</div>
```

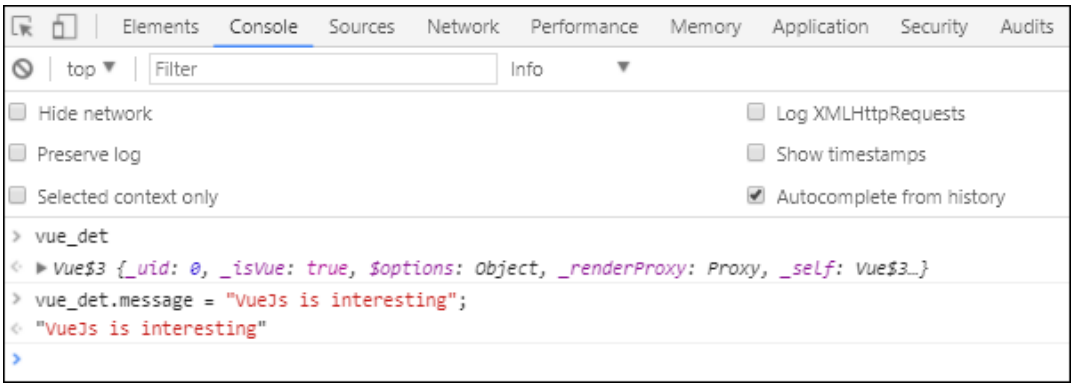
```
const vue_det = new Vue({
  el: '#intro',
  data: {
    message: 'My first VueJS Task'
  }
});
```

Se llama a la instancia de Vue, la cual toma el ID del elemento del DOM, por ejemplo '#intro'. Hay datos dentro de dicha instancia con el mensaje al cual se le asigna el valor 'My first VueJS Task'. Vue.JS interactúa con el DOM y cambia el valor en el DOM {{message}} con el mensaje 'My first VueJS Task'.

También es posible cambiar el valor del mensaje en consola:



Detalles en consola



Para empezar con Vue.JS es necesario crear una instancia de Vue, que es conocida como la instancia Root de Vue.

Sintaxis

```
const app = new Vue({
  // options
})
```

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "vue_det">
      <h1>Firstname : {{firstname}}</h1>
      <h1>Lastname : {{lastname}}</h1>
      <h1>{{mydetails()}}</h1>
    </div>
    <script type = "text/javascript" src = "js/vue_instance.js"></script>
  </body>
</html>
```

vue_instance.js

```
const  vm = new Vue({
```

```
  el: '#vue_det',
  data: {
    firstname : "Ria",
    lastname  : "Singh",
    address    : "Mumbai"
  },
  methods: {
    mydetails : function() {
       return "I am "+this.firstname +" "+ this.lastname;
    }
  }
});
```

Hay un parámetro llamado **el**. Toma el id del elemento a modificar en el DOM.

```
<div id = "vue_det"></div>
```

Lo que escribamos dentro de los {{}} afectará únicamente al div seleccionado y a nada más fuera de él. Posteriormente se definen los datos, nombre, apellido y dirección.

```
<div id = "vue_det">
  <h1>Firstname : {{firstname}}</h1>
  <h1>Lastname : {{lastname}}</h1>
  <h1>Address : {{address}}</h1>
</div>
```

El valor del nombre se reemplazará dentro de la primera interpolación y así sucesivamente con los demás parámetros.

Posteriormente se define una función llamada mydetails() que devuelve un valor que será asignado a la interpolación siguiente:

```
<h1>{{mydetails()}}</h1>
```

Output

Podemos también mostrar en el DOM un template HTML generado dentro de una instancia de Vue.JS

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "vue_det">
      <h1>Firstname : {{firstname}}</h1>
      <h1>Lastname : {{lastname}}</h1>
      <div>{{htmlcontent}}</div>
    </div>
    <script type = "text/javascript" src = "js/vue_template.js"></script>
  </body>
</html>
```

vue_template.js

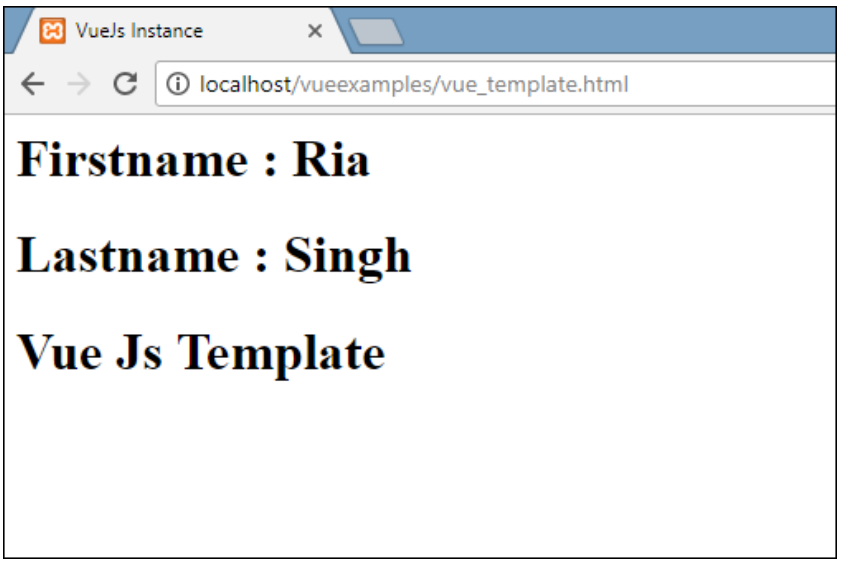
```
const vm = new Vue({
  el: '#vue_det',
  data: {
    firstname : "Ria",
    lastname  : "Singh",
    htmlcontent : "<div><h1>Vue Js Template</h1></div>"
  }
});
```

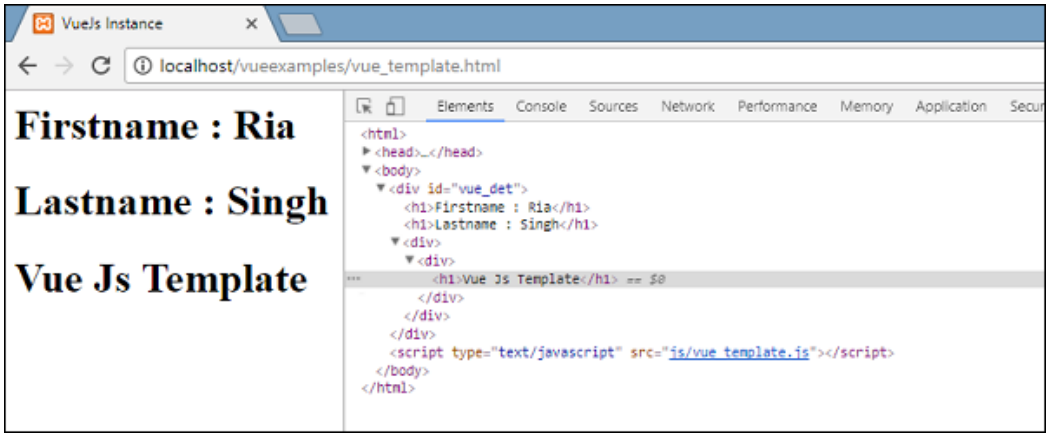
Esto no es correcto, dado que el resultado en el navegador sería el siguiente:

Si necesitamos insertar código HTML en el dom desde Vue.JS se utiliza la directiva **v-html**. Desde el momento que se usa Vue.JS ya sabe que tiene que mostrar dicho contenido como HTML en el navegador.

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "vue_det">
      <h1>Firstname : {{firstname}}</h1>
      <h1>Lastname : {{lastname}}</h1>
      <div v-html = "htmlcontent"></div>
    </div>
    <script type = "text/javascript" src = "js/vue_template.js"></script>
  </body>
</html>
```

```
const vm = new Vue({
  el: '#vue_det',
  data: {
    firstname : "Ria",
    lastname  : "Singh",
    htmlcontent : "<div><h1>Vue Js Template</h1></div>"
  }
});
```



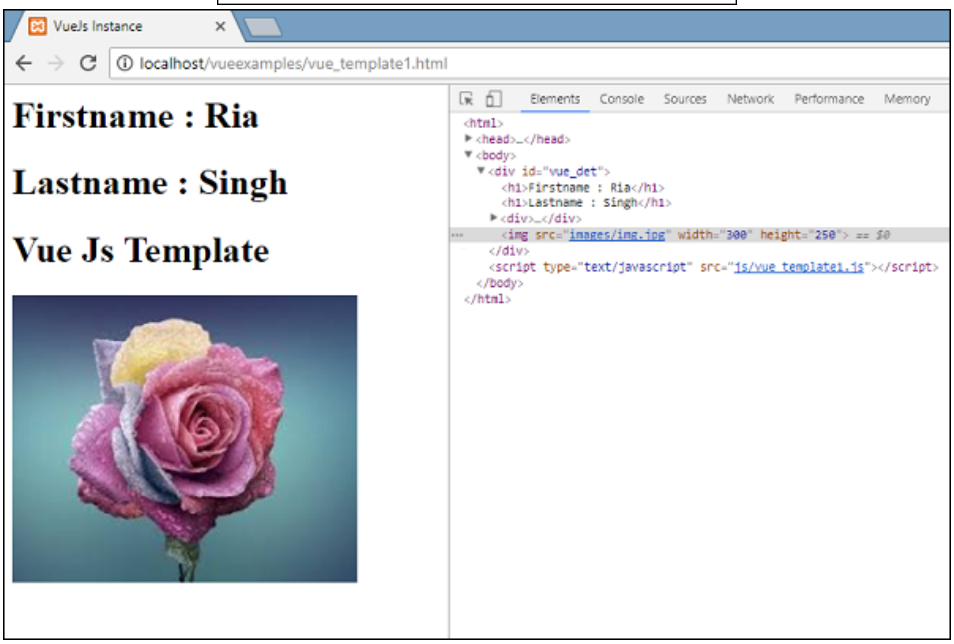


Para añadir **atributos** a los elementos del DOM utilizamos la directiva v-bind, que en este caso se orienta al atributo src.

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "vue_det">
      <h1>Firstname : {{firstname}}</h1>
      <h1>Lastname : {{lastname}}</h1>
      <div v-html = "htmlcontent"></div>
      <img v-bind:src = "imgsrc" width = "300" height = "250" />
    </div>
    <script type = "text/javascript" src = "js/vue_template1.js"></script>
  </body>
</html>
```

En la instancia Vue se agrega en el apartado **data** la siguiente línea

imgsrc : "img/paisaje.jpg"



v-once

Todo elemento que posea la directiva **v-once** será renderizado sólo una vez.

```
<span v-once>Mensaje: {{ msg }}</span>
```

Componentes

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "component_test">
      <testcomponent></testcomponent>
    </div>
    <div id = "component_test1">
      <testcomponent></testcomponent>
    </div>
    <script type = "text/javascript" src = "js/vue_component.js"></script>
  </body>
</html>
```

vue_component.js

```
Vue.component('testcomponent',{
  template : '<div><h1>This is coming from component</h1></div>'
});
const vm = new Vue({
  el: '#component_test'
});
const vm1 = new Vue({
  el: '#component_test1'
});
```

Se crean dos div con id component_test y component_test1, junto con dos instancias Vue que hacen referencia a dichos ids. Se crea un componente en común que será asignado a ambas instancias.

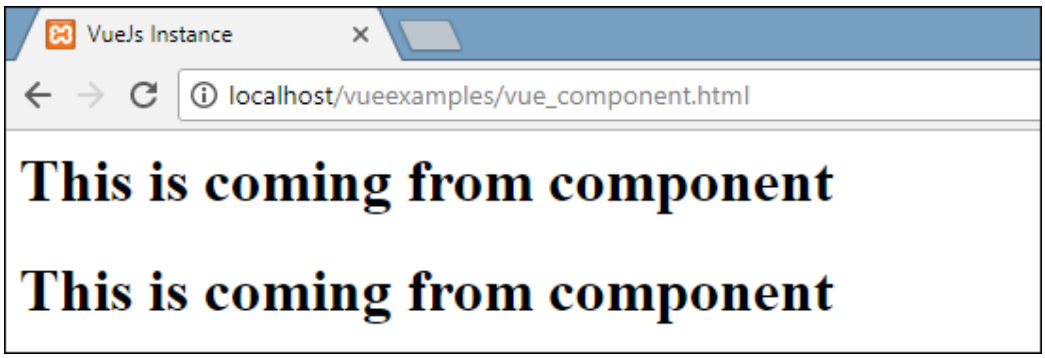
Para crear un componente se utiliza la siguiente sintaxis:

```
Vue.component('nombredelcomponente',{ // options});
```

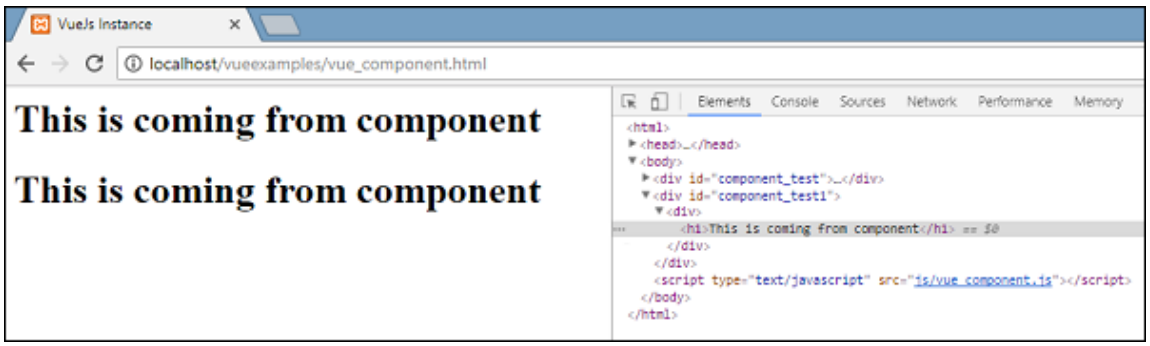
```
<div id = "component_test">
  <testcomponent></testcomponent>
</div>
<div id = "component_test1">
  <testcomponent></testcomponent>
</div>
```

Dentro del componente se agrega un template el cual tiene asignado código HTML. Esta es la manera de registrar globalmente un componente en Vue.JS, que puede ser reutilizado en cualquier instancia Vue.

```
Vue.component('testcomponent',{
  template : '<div><h1>This is coming from component</h1></div>'
});
```



El nombre personalizado de las etiquetas será reemplazado por el código del template en escrito en la creación del componente. En el navegador no existe ningún componente con etiquetas <testcomponent></testcomponent>.



También se puede modificar el comportamiento de un componente global dentro de una instancia Vue.

```
const vm = new Vue({
  el: '#component_test',
  components:{
    'testcomponent': {
       template : '<div><h1>This is coming from component</h1></div>'
    }
  }
});
```

Example

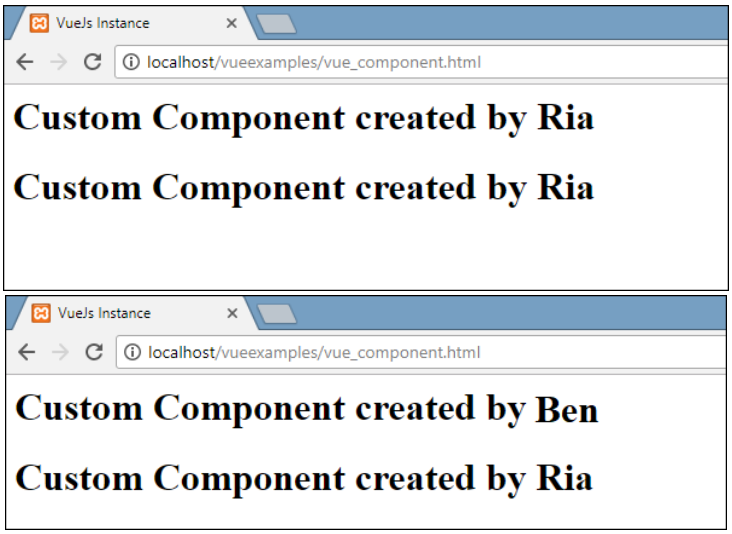
```
<html>
   <head>
      <title>VueJs Instance</title>
      <script type = "text/javascript" src = "js/vue.js"></script>
   </head>
   <body>
      <div id = "component_test">
         <testcomponent></testcomponent>
      </div>
      <div id = "component_test1">
         <testcomponent></testcomponent>
      </div>
      <script type = "text/javascript" src = "js/vue_component.js"></script>
   </body>
</html>
```

vue_component.js

```
Vue.component('testcomponent',{
   template : '<div v-on:mouseover = "changename()" v-on:mouseout =
"originalname();"><h1>Custom Component created by <span id =
"name">{{name}}</span></h1></div>',
   data: function() {
      return {
         name : "Ria"
      }
   },
   methods:{
      changename : function() {
         this.name = "Ben";
      },
      originalname: function() {
         this.name = "Ria";
      }
   }
});
const vm = new Vue({
   el: '#component_test'
});
const vm1 = new Vue({
   el: '#component_test1'
```

```
});
```

En el ejemplo anterior, el template dentro del componente hace que cuando el mouse se posicione encima del mismo ejecute la función changename() y cuando el mismo salga del componente se ejecute la función originalname(). Ambas funciones trabajan sobre la propiedad name del componente, cambiandola.



Componentes dinámicos

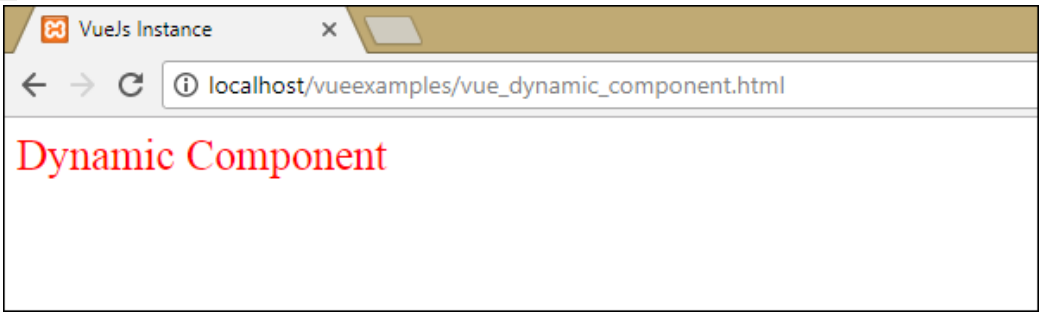
Los componentes dinámicos se crean con la etiqueta <component></component> y se ligan con un elemento del DOM de la siguiente manera.

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>

    <div id = "databinding">
      <component v-bind:is = "view"></component>
    </div>

<script type = "text/javascript">
     const vm = new Vue({
         el: '#databinding',
         data: {
           view: 'component1'
         },
         components: {
           'component1': {
             template: '<div><span style = "font-size:25;color:red;">Dynamic
Component</span></div>'
```

```
            }
         }
      });
   </script>
  </body>
</html>
```



Los componentes dinámicos son creados de la siguiente manera:

```
<component v-bind:is = "view"></component>
```

El componente posee una directiva **v-bind:is**, cuyo valor es “view” y un valor asignado a ese **view**. View está definido en la instancia Vue.

```
var vm = new Vue({
  el: '#databinding',
  data: {
    view: 'component1'
  },
  components: {
    'component1': {
      template: '<div><span style = "font-size:25;color:red;">Dynamic
Component</span></div>'
    }
  }
});
```

Propiedades computadas

Las propiedades computadas son similares a los métodos, pero las mismas no aceptan parámetros y su invocación no lleva () por que se ha vuelto una propiedad de la instancia Vue. Además, los métodos son llamados en todo momento cuando se actualice la vista, mientras que las propiedades computadas analizan el código y solo se actualizan si las propiedades involucradas en este cómputo cambian su valor.

```
<html>

  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "computed_props">
```

```
        FirstName : <input type = "text" v-model = "firstname" /> <br/><br/>
        LastName : <input type = "text" v-model = "lastname"/> <br/><br/>
        <h1>My name is {{firstname}} {{lastname}}</h1>
        <h1>Using computed method : {{getfullname}}</h1>
    </div>
    <script type = "text/javascript" src = "js/vue_computedprops.js"></script>
  </body>
</html>
```

vue_computeprops.js

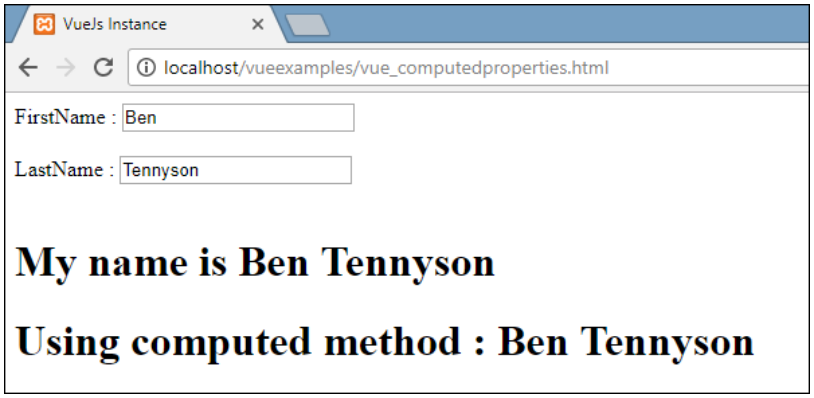
```
const vm = new Vue({
  el: '#computed_props',
  data: {
    firstname :"",
    lastname :"",
    birthyear : ""
  },
  computed :{
    getfullname : function(){
      return this.firstname +" "+ this.lastname;
    }
  }
});
```

Se crea el documento con dos elementos input tipo caja de texto, los cuales están ligados a la instancia Vue con las directivas **v-model**.

Posteriormente tenemos la propiedad computada getfullname, que devuelve el firstname y el lastname que ingresó el usuario concatenados.

```
computed :{
  getfullname : function(){
    return this.firstname +" "+ this.lastname;
  }
}
```

Lo que se escribe en los textboxes es lo mismo que devolverá la función, cuando los valores de los textboxes cambien así también lo hará lo devuelto por la función, gracias a utilizar las propiedades computadas que se llaman automáticamente cuando los textboxes cambien.



Acceder a los elementos del DOM utilizando $refs

Todo el tiempo necesitamos acceder a elementos del DOM utilizando JavaScript.
Vue.JS nos permite hacerlo de una manera supremamente sencilla.
Las instancias de Vue.JS cuentan con diversas propiedades, una de ellas es $refs.
Visto en código sería algo como:

app.$refs
vm.$refs

Donde “app” o “vm” representan la instancia misma de Vue (por convención se utiliza “app” o “vm” para nombrar a la instancia de Vue, pero puedes utilizar el nombre que desees, también puedes utilizar la palabra reservada “this” para referirte a la instancia. En éste ejemplo utilizaremos el nombre de la instancia, en cuyo caso es “app”) y $refs sería una propiedad propia de la instancia.

Ahora bien, ¿qué es examente “$refs”?
Es un objeto, dentro de él se van a almacenar todos los elementos del DOM que cuenten con el atributo “ref”. El atributo “ref” vendría a ser algo así como darle un ID al elemento.
<input ref="entrada"></input>

Podemos tener todos los elementos que deseemos con el atributo “ref”, siempre y cuando el valor del atributo sea diferente para cada elemento.
<input ref="entrada"></input>
<input ref="entrada2"></input>

Para acceder al objeto que almacena estos elementos bastaría con llamarlo de la siguiente manera:
app.$refs

Y para acceder al elemento dentro del objeto:
app.$refs.entrada

Vamos a aplicar lo anterior con una aplicación sencilla
Lo que hace la aplicación es añadir a un párrafo el texto que escribamos en una entrada:

Lo primero es crear el HTML y la instancia de Vue.
HTML:

```
<div id="app">
	<h1>Accediendo a Elementos del DOM utilizando vm.$refs</h1>
	<h2>Añade texto a la página</h2>
	<input type="text">
	<br>
	<br>
	<button type="button">Guardar</button>
	<button type="button">Borrar</button>
	<p></p>
</div>
```

Instancia de Vue:

```
const app = new Vue({
	el : '#app'
});
```

Vamos colocarle los atributos “ref” a los elementos cuyas propiedades queremos acceder, que en este caso son el input y el párrafo.

```
<div id="app">
	<h1>Accediendo a Elementos del DOM utilizando vm.$refs</h1>
	<h2>Añade texto a la página</h2>
	<input type="text" ref="text">
	<br>
	<br>
	<button type="button">Guardar</button>
	<button type="button">Borrar</button>
	<p ref="textField"></p>
</div>
```

Ahora vamos a escribir los métodos **addText()** y **deleteText()** de los botones en la instancia de Vue.

```
const app = new Vue({
	el : '#app',
	methods : {
```

```
        addText () {
            const text = app.$refs.text.value
            const textField = app.$refs.textField
            textField.innerHTML = textField.innerHTML + '<br />' + text
        },
        deleteText () {
            const textField = app.$refs.textField
            textField.innerHTML = ''
        }
    }
});
```

Y por último ponemos los botones a la escucha de los métodos.

```
<div id="app">
    <h1>Accediendo a Elementos del DOM utilizando vm.$refs</h1>
    <h2>Añade texto a la página</h2>
    <input type="text" ref="text">
    <br>
    <br>
    <button type="button" @click="addText">Guardar</button>
    <button type="button" @click="deleteText">Borrar</button>
    <p ref="textField"></p>
</div>
```

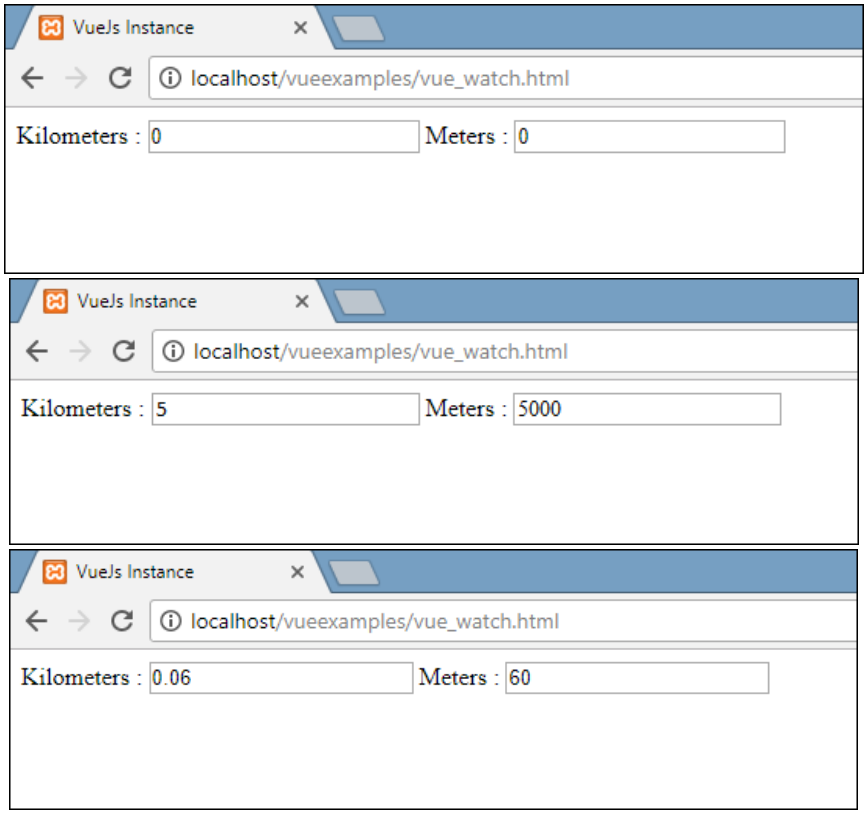
Fuente:tutorialspoint.com

Watchers

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "computed_props">
      Kilometers : <input type = "text" v-model = "kilometers">
      Meters : <input type = "text" v-model = "meters">
    </div>
    <script type = "text/javascript">
      const vm = new Vue({
        el: '#computed_props',
        data: {
          kilometers : 0,
          meters:0
        },
        methods: {
        },
        computed :{
        },
        watch : {
          kilometers:function(val) {
            this.kilometers = val;
            this.meters = val * 1000;
          },
          meters : function (val) {
            this.kilometers = val/ 1000;
            this.meters = val;
          }
        }
      });
    </script>
  </body>
</html>
```

Se crean dos textboxes, uno con kilómetros y otro con metros. En **data** ambas propiedades son inicializadas en cero. Existe dentro de la instancia Vue un objeto **watch** que se crea con dos funciones, cuyo objetivo es convertir de kilómetros a metros y viceversa.

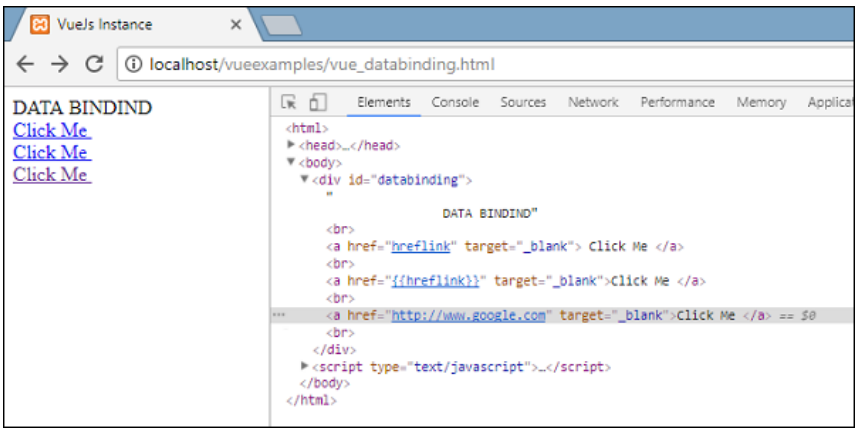
Cada vez que se ingrese algún valor en los textboxes, **watch** se encarga de actualizarlos calculando lo que está declarado en las funciones.



Más sobre manipulación de atributos

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "databinding">
      {{title}}<br/>
      <a href = "hreflink" target = "_blank"> Click Me </a> <br/>
      <a href = "{{hreflink}}" target = "_blank">Click Me </a>  <br/>
      <a v-bind:href = "hreflink" target = "_blank">Click Me </a>   <br/>
    </div>
    <script type = "text/javascript">
      const vm = new Vue({
        el: '#databinding',
        data: {
          title : "DATA BINDING",
          hreflink : "http://www.google.com"
        }
      });
    </script>
  </body>
</html>
```

En el ejemplo anterior se presentan tres supuestas formas de enlazar el atributo **href** a los enlaces. Pero sólo una es correcta al revisar el ejemplo en el navegador.



Como fue demostrado en anteriores ejemplos, para enlazar atributos desde una instancia Vue al DOM se utiliza la directiva **v-bind:atributo**. En este caso trabajaremos sobre **href.**

```
<a v-bind:href = "hreflink" target = "_blank">Click Me </a>
```

Vue.JS nos brinda un atajo a dicha directiva eliminando **v-bind.** Ninguna de las propiedades de Vue.JS será mostrada en el inspector del navegador.

```
<a :href = "hreflink" target = "_blank">Click Me </a>
```

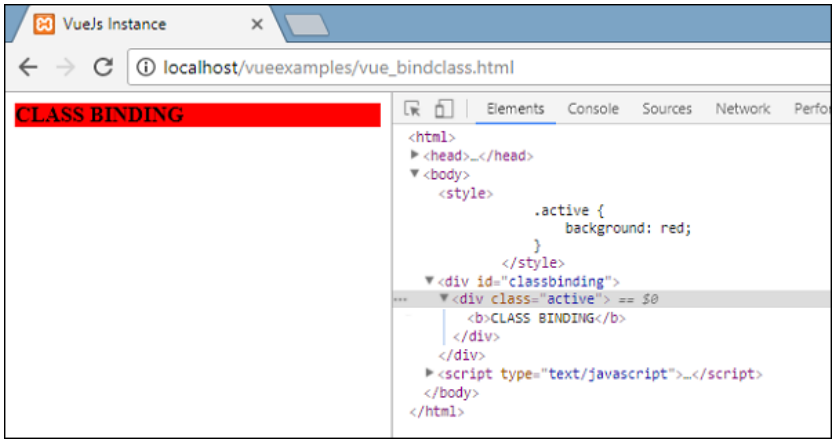
Enlazando clases HTML

Para enlazar clases se utiliza también la directiva **v-bind**, pero utilizando el atributo **class**.

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <style>
      .active {
        background: red;
      }
    </style>
    <div id = "classbinding">
      <div v-bind:class = "{active:isactive}"><b>{{title}}</b></div>
    </div>
    <script type = "text/javascript">
      const vm = new Vue({
        el: '#classbinding',
        data: {
          title : "CLASS BINDING",
          isactive : true
        }
      });
```

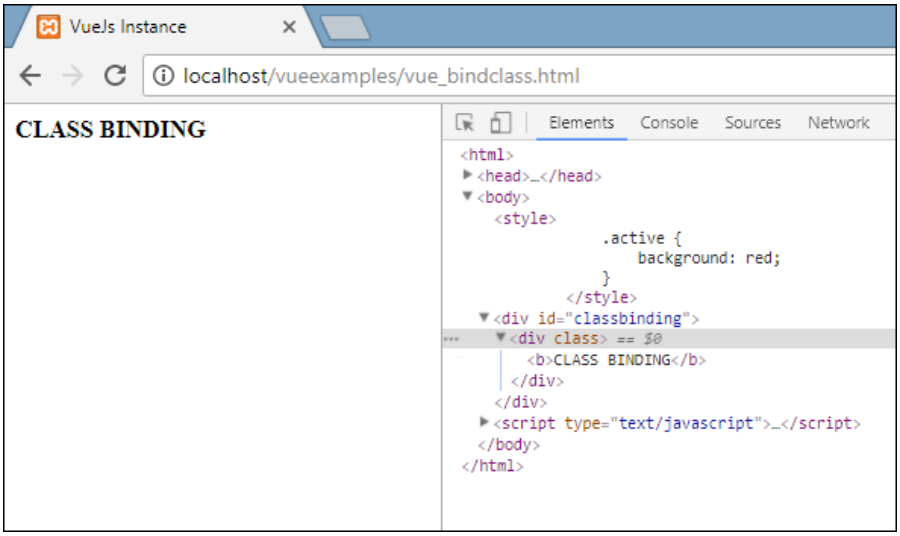
```
    </script>
  </body>
</html>
```

isactive es una variable booleana, la cual puede contener en su interior true o false lo cual hará que se aplique o no la clase **active** al div enlazado a la instancia Vue. La clase active cambia el fondo del elemento a un color rojo.



En el siguiente ejemplo a dicha variable se le asigna el valor false.

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <style>
      .active {
        background: red;
      }
    </style>
    <div id = "classbinding">
      <div v-bind:class = "{active:isactive}"><b>{{title}}</b></div>
    </div>
    <script type = "text/javascript">
      const vm = new Vue({
        el: '#classbinding',
        data: {
          title : "CLASS BINDING",
          isactive : false
        }
      });
    </script>
  </body>
</html>
```



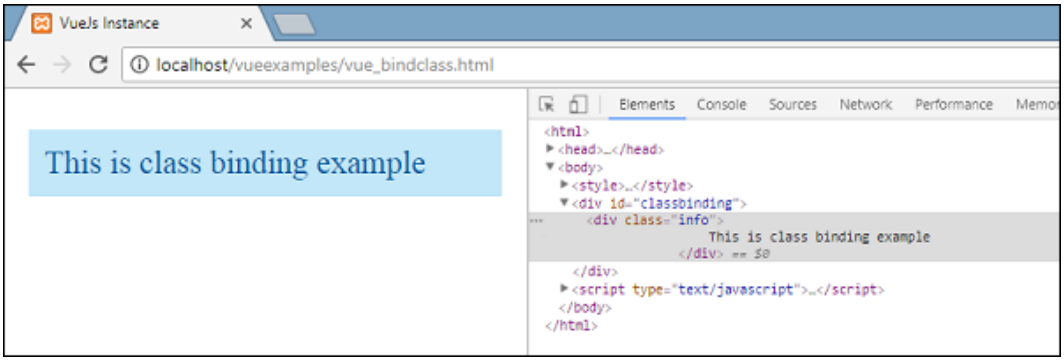
El siguiente ejemplo enlaza múltiples clases al elemento HTML.

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <style>
      .info {
        color: #00529B;
        background-color: #BDE5F8;
      }
      div {
        margin: 10px 0;
        padding: 12px;
      }
      .active {
        color: #4F8A10;
        background-color: #DFF2BF;
      }
      .displayError{
        color: #D8000C;
        background-color: #FFBABA;
      }
    </style>
```

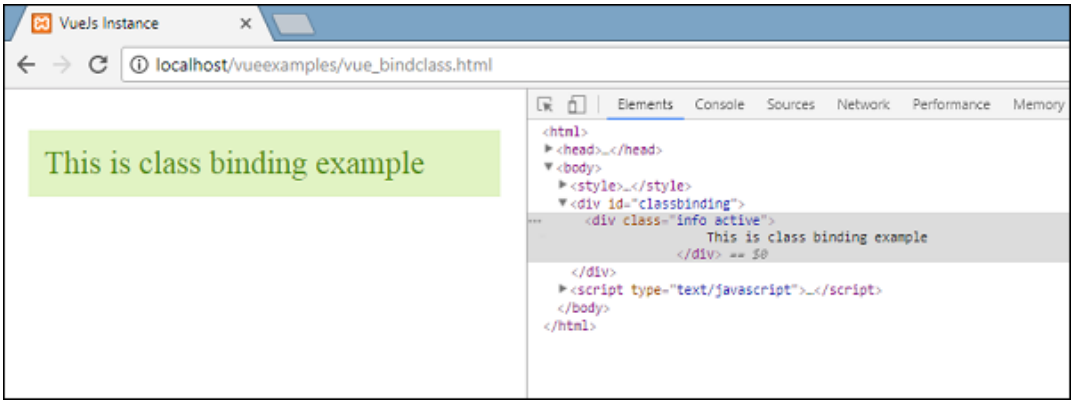
```
<div id = "classbinding">
```

```
        <div class = "info"  v-bind:class = "{ active: isActive, 'displayError': hasError }">
          {{title}}
        </div>
    </div>
    <script type = "text/javascript">
        const vm = new Vue({
          el: '#classbinding',
          data: {
            title : "This is class binding example",
            isActive : false,
            hasError : false
          }
        });
    </script>
  </body>
</html>
```

Con ambas variables con valores false el elemento HTML se ve de la siguiente manera:



Con la variable isactive en true:



Con ambas variables en true:

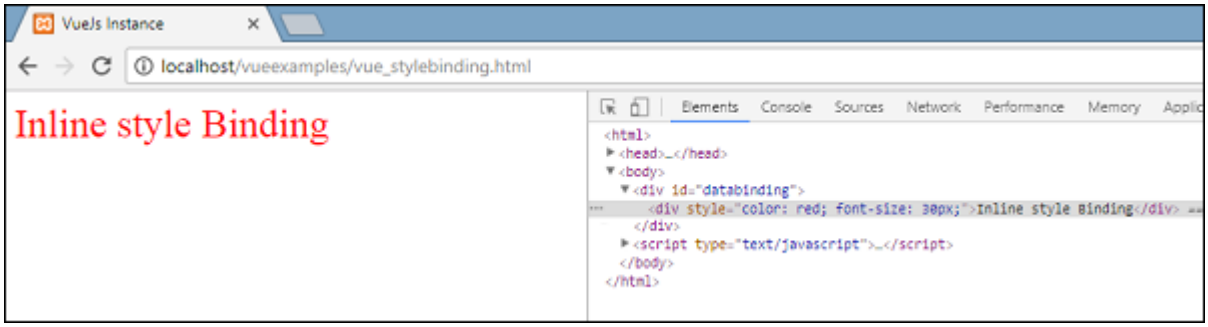
El siguiente ejemplo aplica **v-bind** para clases en un componente.

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <style>
      .info {
        color: #00529B;
        background-color: #BDE5F8;
      }
      div {
        margin: 10px 0;
        padding: 12px;
        font-size : 25px;
      }
      .active {
        color: #4F8A10;
        background-color: #DFF2BF;
      }
      .displayError{
        color: #D8000C;
        background-color: #FFBABA;
      }
    </style>
    <div id = "classbinding">
      <new_component class = "active"></new_component>
    </div>
    <script type = "text/javascript">
      var vm = new Vue({
        el: '#classbinding',
        data: {
          title : "This is class binding example",
          infoclass : 'info',
          errorclass : 'displayError',
          isActive : false,
          haserror : true
```

```
        },
        components:{
          'new_component' : {
            template : '<div class = "info">Class Binding for component</div>'
          }
        }
      });
    </script>
  </body>
</html>
```

Enlazar estilos inline

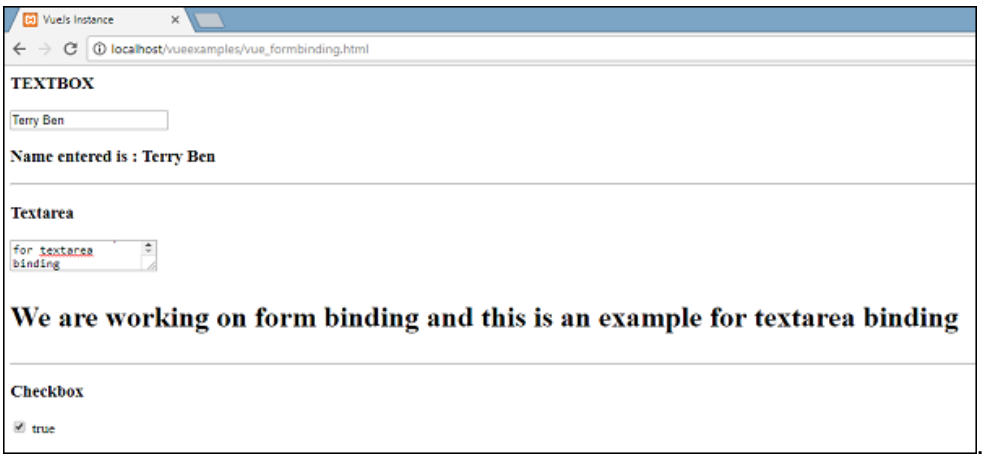
```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "databinding">
      <div v-bind:style = "{ color: activeColor, fontSize: fontSize + 'px' }">{{title}}</div>
    </div>
    <script type = "text/javascript">
      const vm = new Vue({
        el: '#databinding',
        data: {
          title : "Inline style Binding",
          activeColor: 'red',
          fontSize :'30'
        }
      });
    </script>
  </body>
</html>
```



Enlazar elementos input de un formulario

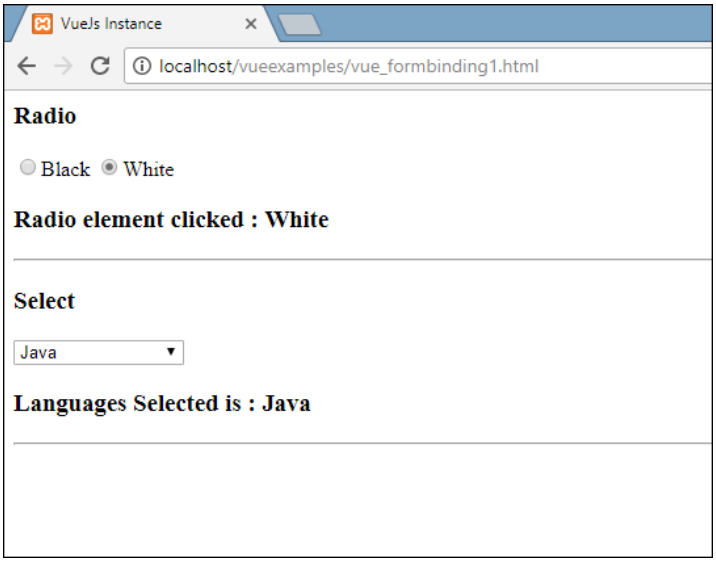
```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "databinding">
      <h3>TEXTBOX</h3>
      <input  v-model = "name" placeholder = "Enter Name" />
      <h3>Name entered is : {{name}}</h3>
      <hr/>
      <h3>Textarea</h3>
      <textarea v-model = "textmessage" placeholder = "Add Details"></textarea>
      <h1><p>{{textmessage}}</p></h1>
      <hr/>
      <h3>Checkbox</h3>
      <input type = "checkbox" id = "checkbox" v-model = "checked"> {{checked}}
    </div>
    <script type = "text/javascript">
      const vm = new Vue({
        el: '#databinding',
        data: {
          name:'',
          textmessage:'',
          checked : false
        }
      });
    </script>
  </body>
</html>
```

Lo que escribamos en el textbox se mostrará debajo. **v-model** tiene asignado el valor de la propiedad **name** y dicho nombre es mostrado en {{name}}, que muestra lo que tenga escrito en su interior el textbox, lo mismo sucede con el checkbox y el textarea.



Radio y Select

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "databinding">
      <h3>Radio</h3>
      <input type = "radio" id = "black" value = "Black" v-model = "picked">Black
      <input type = "radio" id = "white" value = "White" v-model = "picked">White
      <h3>Radio element clicked : {{picked}} </h3>
      <hr/>
      <h3>Select</h3>
      <select v-model = "languages">
        <option disabled value = "">Please select one</option>
        <option>Java</option>
        <option>Javascript</option>
        <option>Php</option>
        <option>C</option>
        <option>C++</option>
      </select>
      <h3>Languages Selected is : {{ languages }}</h3>
      <hr/>
    </div>
    <script type = "text/javascript">
      var vm = new Vue({
        el: '#databinding',
        data: {
          picked : 'White',
          languages : "Java"
        }
      });
    </script>
  </body>
</html>
```



Modificadores

Veremos tres tipos: **trim**, **number**, y **lazy**.

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "databinding">
      <span style = "font-size:25px;">Enter Age:</span> <input v-model.number = "age"
type = "number">
      <br/>
      <span style = "font-size:25px;">Enter Message:</span> <input v-model.lazy = "msg">
      <h3>Display Message : {{msg}}</h3>
      <br/>
      <span style = "font-size:25px;">Enter Message : </span><input v-model.trim =
"message">
      <h3>Display Message : {{message}}</h3>
    </div>
    <script type = "text/javascript">
      var vm = new Vue({
        el: '#databinding',
        data: {
          age : 0,
          msg: '',
          message : ''
        }
      });
    </script>
  </body>
</html>
```

El modificador **number** sólo permite el ingreso de números, no tomará otra entrada salvo que sean números.

```
<span style = "font-size:25px;">Enter Age:</span> <input v-model.number = "age" type = "number">
```

El modificador **lazy** mostrará el contenido presente en el textbox cuando el usuario abandone el mismo.

```
<span style = "font-size:25px;">Enter Message:</span> <input v-model.lazy = "msg">
```

El modificador **trim** eliminará los espacios al principio y al final de lo ingresado en el textbox.

```
<span style = "font-size:25px;">Enter Message : </span><input v-model.trim = "message">
```

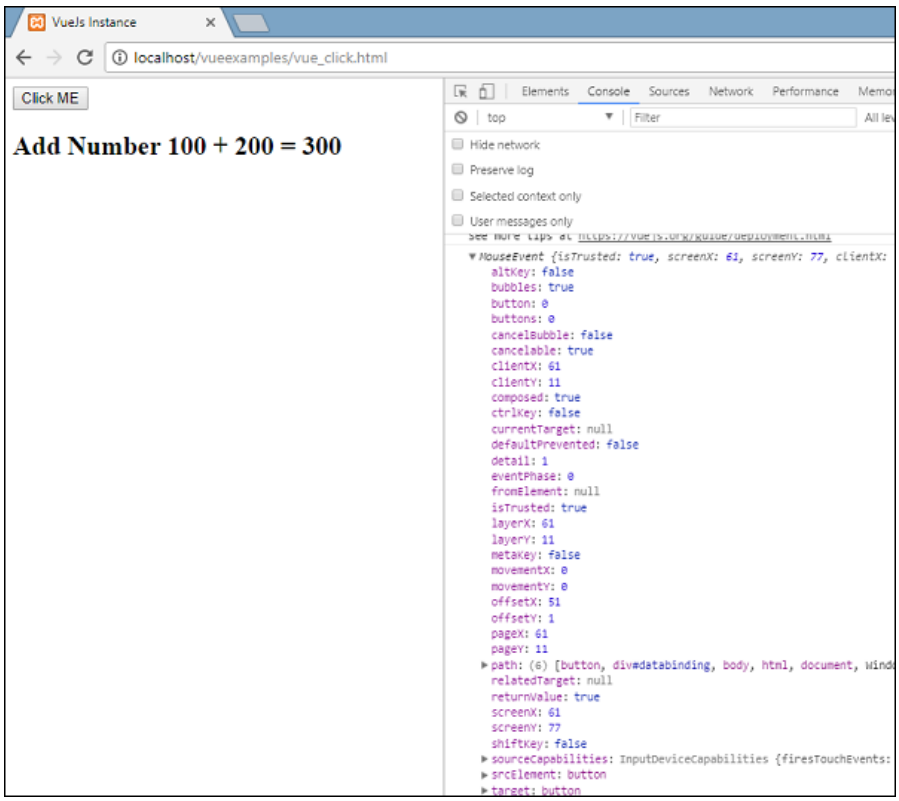
Eventos

Utilizaremos la directiva **v-on** en los elementos HTML del DOM para escuchar a los eventos.

Evento de Click

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "databinding">
      <button v-on:click = "displaynumbers">Click ME</button>
      <h2> Add Number 100 + 200 = {{total}}</h2>
    </div>
    <script type = "text/javascript">
      const vm = new Vue({
         el: '#databinding',
         data: {
```

```
            num1: 100,
            num2 : 200,
            total : ''
        },
        methods : {
            displaynumbers : function(event) {
              console.log(event);
              return this.total =  this.num1+ this.num2;
            }
        },
    });
  </script>
 </body>
</html>
```



El siguiente código asignará un evento de click al elemento del DOM.

```
<button v-on:click = "displaynumbers">Click ME</button>
```

Vue.JS nos brinda una propiedad atajo para los eventos:

```
<button @click = "displaynumbers">Click ME</button>
```

Cuando se cliquee el botón, se llamará al metodo displaynumbers.

Eventos mouseover, mouseout

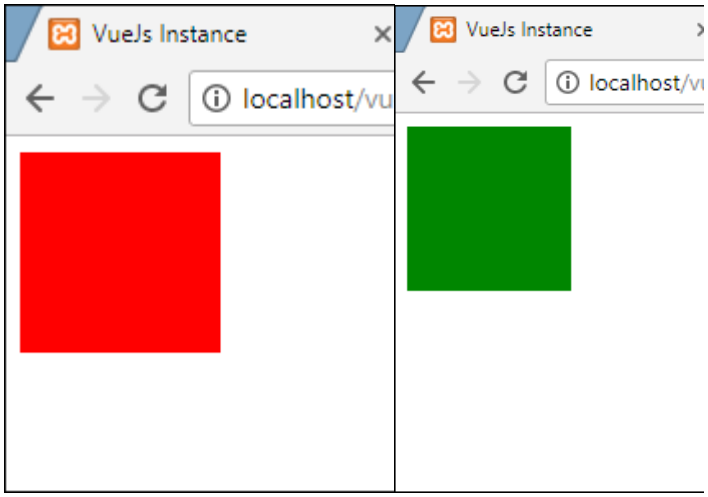
```
<html>
```

```
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "databinding">
      <div v-bind:style = "styleobj" v-on:mouseover = "changebgcolor" v-on:mouseout =
"originalcolor"></div>
    </div>
    <script type = "text/javascript">
      const vm = new Vue({
        el: '#databinding',
        data: {
          styleobj : {
            width:"100px",
            height:"100px",
            backgroundColor:"red"
          }
        },
        methods : {
          changebgcolor : function() {
            this.styleobj.backgroundColor = "green";
          },
          originalcolor : function() {
            this.styleobj.backgroundColor = "red";
          }
        },
      });
    </script>
  </body>
</html>
```

Se crea un div con ancho y alto de 100 px. Se le asigna un background de color rojo. Cuando el mouse se posiciona sobre él se cambiará dicho fondo al color verde y cuando el mouse salga del elemento volverá al color rojo mediante los métodos changebgcolor y originalcolor.

```
<div v-bind:style = "styleobj" v-on:mouseover = "changebgcolor" v-on:mouseout =
"originalcolor"></div>

changebgcolor : function() {
  this.styleobj.backgroundColor = "green";
}
originalcolor : function() {
  this.styleobj.backgroundColor = "red";
}
```



Modificadores de eventos

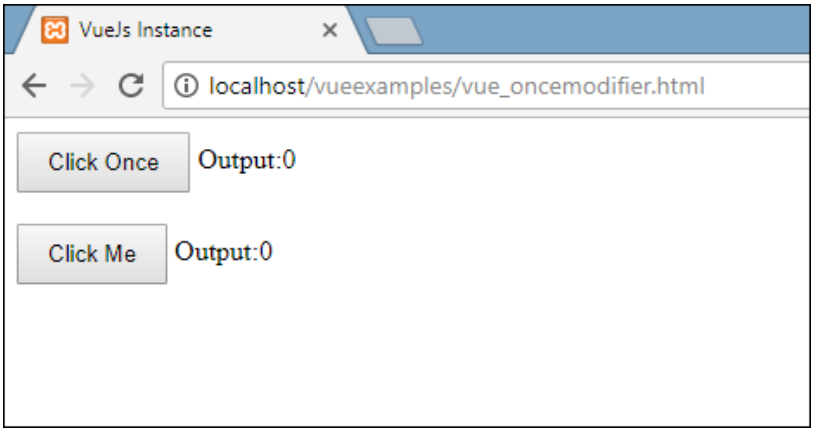
Vue.JS tiene disponibles modificadores de eventos en la directiva **v-on**.

.once

Permite que el evento se ejecute sólo una vez.

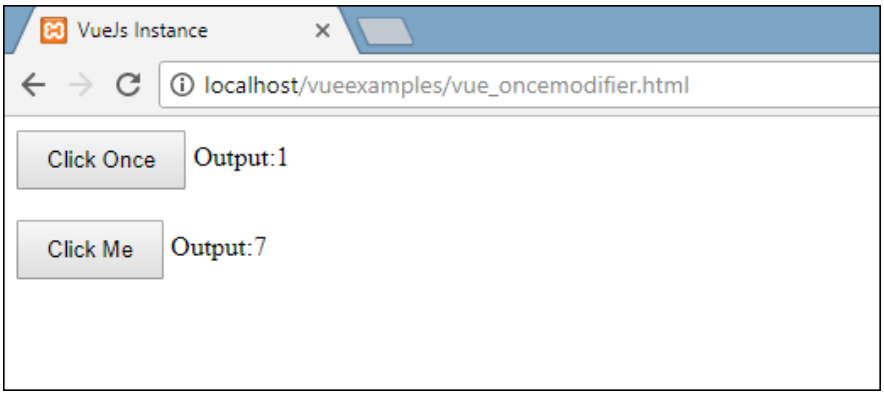
```
<button v-on:click.once = "buttonclicked">Click Once</button>
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "databinding">
      <button v-on:click.once = "buttonclickedonce" v-bind:style = "styleobj">Click
Once</button>
      Output:{{clicknum}}
      <br/><br/>
      <button v-on:click = "buttonclicked"  v-bind:style = "styleobj">Click Me</button>
      Output:{{clicknum1}}
    </div>
    <script type = "text/javascript">
      const vm = new Vue({
        el: '#databinding',
        data: {
          clicknum : 0,
          clicknum1 :0,
          styleobj: {
            backgroundColor: '#2196F3!important',
            cursor: 'pointer',
            padding: '8px 16px',
            verticalAlign: 'middle',
          }
        },
```

```
        methods : {
            buttonclickedonce : function() {
              this.clicknum++;
            },
            buttonclicked : function() {
              this.clicknum1++;
            }
          }
        });
    </script>
  </body>
</html>
```



En el anterior ejemplo se crean dos botones. Uno que posee el modificador **.once** y otro no. Uno podrá ejecutar tantas veces como el usuario desee el evento y el otro (con el modificador .once) no.

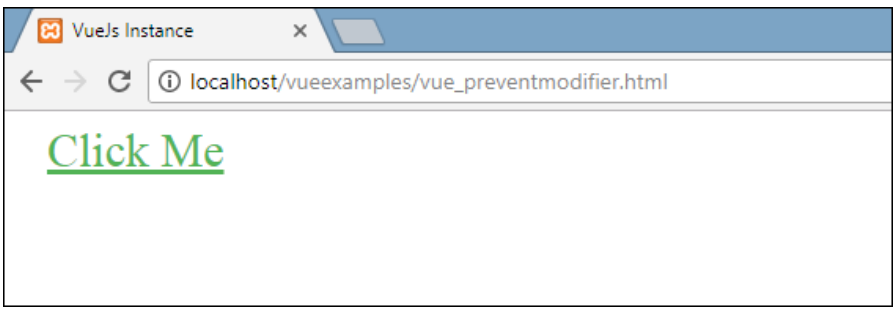
```
<button v-on:click.once = "buttonclickedonce" v-bind:style = "styleobj">Click Once</button>
<button v-on:click = "buttonclicked"  v-bind:style = "styleobj">Click Me</button>
buttonclickedonce : function() {
  this.clicknum++;
},
buttonclicked : function() {
  this.clicknum1++;
}
```



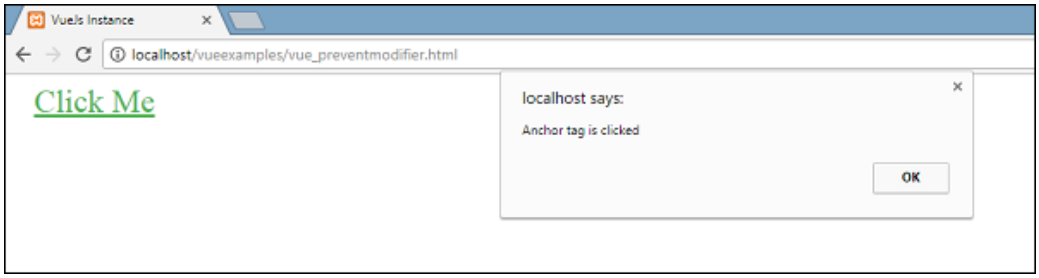
.prevent

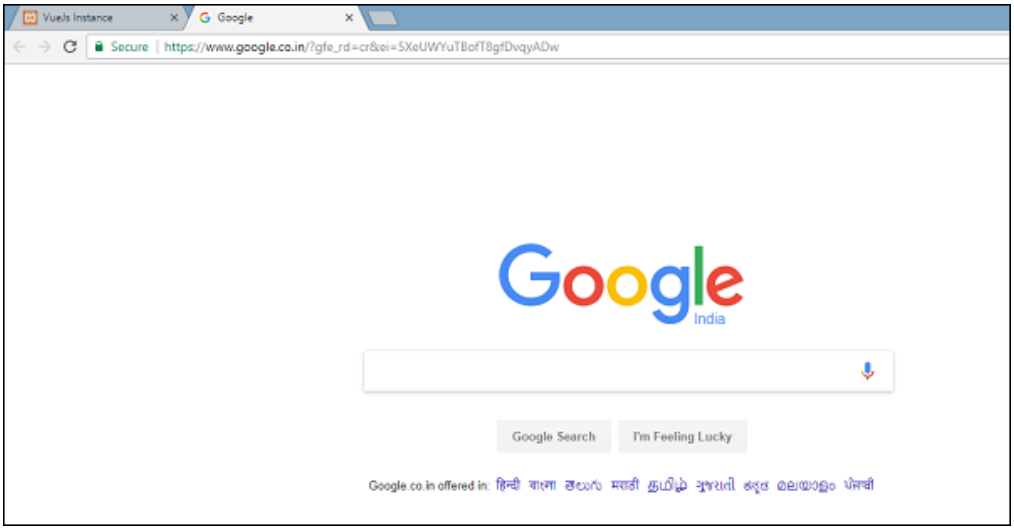
```
<a href = "http://www.google.com" v-on:click.prevent = "clickme">Click Me</a>
```

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "databinding">
      <a href = "http://www.google.com" v-on:click = "clickme" target = "_blank" v-bind:style
= "styleobj">Click Me</a>
    </div>
    <script type = "text/javascript">
      const vm = new Vue({
        el: '#databinding',
        data: {
          clicknum : 0,
          clicknum1 :0,
          styleobj: {
            color: '#4CAF50',
            marginLeft: '20px',
            fontSize: '30px'
          }
        },
        methods : {
          clickme : function() {
            alert("Anchor tag is clicked");
          }
        }
      });
    </script>
  </body>
</html>
```



Si cliqueamos el link nos aparecerá un alert() que nos informa que el link fué cliqueado.





Si el link no posee el modificador **.prevent** se abrirá el enlace al que apunta el link. Dicho comportamiento es el que tienen por defecto los enlaces al ser clickeados. Con .prevent evitamos dicho comportamiento y ejecutamos la función especificada en las comillas.

```
<a href = "http://www.google.com" v-on:click.prevent = "clickme" target = "_blank" v-bind:style = "styleobj">Click Me</a>
```

Modificadores de teclas

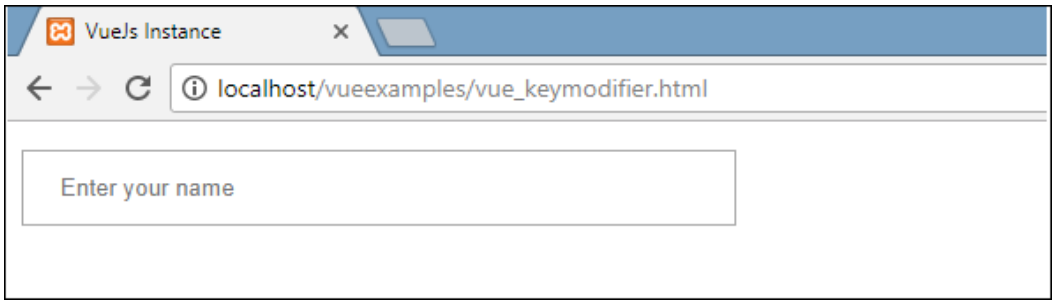
Vue.JS ofrece modificadores de teclas mediante los cuales se manejan los eventos. Si necesitaramos que un textbox llame a un método definido en una instancia Vue sólo cuando se pulsa la tecla enter deberíamos agregar un modificador.

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

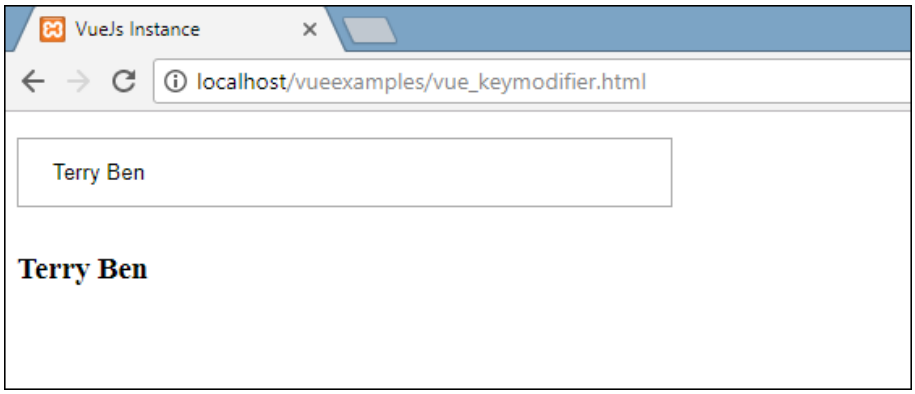
Podemos utilizar múltiples teclas. Por ejemplo: V-on.keyup.ctrl.enter

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "databinding">
      <input type = "text" v-on:keyup.enter = "showinputvalue" v-bind:style = "styleobj" placeholder = "Enter your name"/>
```

```
        <h3> {{name}}</h3>
    </div>
    <script type = "text/javascript">
        const vm = new Vue({
            el: '#databinding',
            data: {
                name:'',
                styleobj: {
                    width: "30%",
                    padding: "12px 20px",
                    margin: "8px 0",
                    boxSizing: "border-box"
                }
            },
            methods : {
                showinputvalue : function(event) {
                    this.name=event.target.value;
                }
            }
        });
    </script>
  </body>
</html>
```



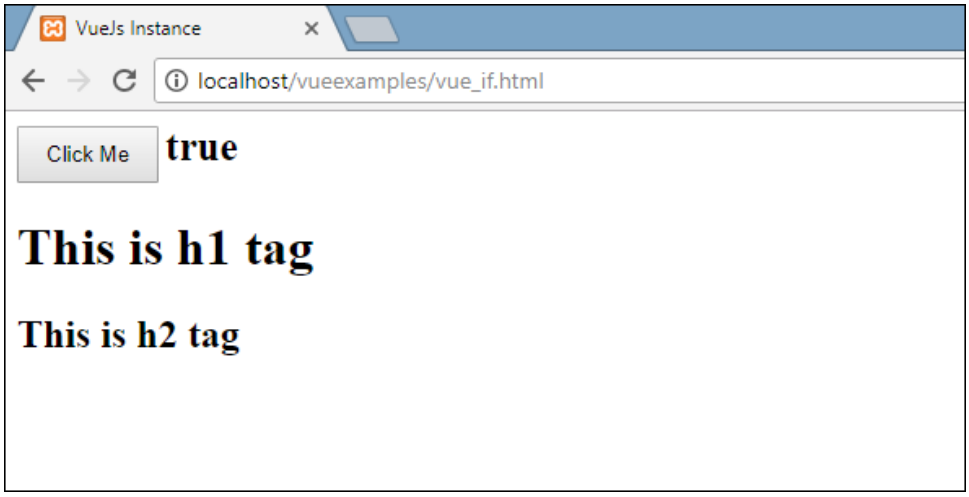
Al presionar Enter:



Renderizado condicional

v-if

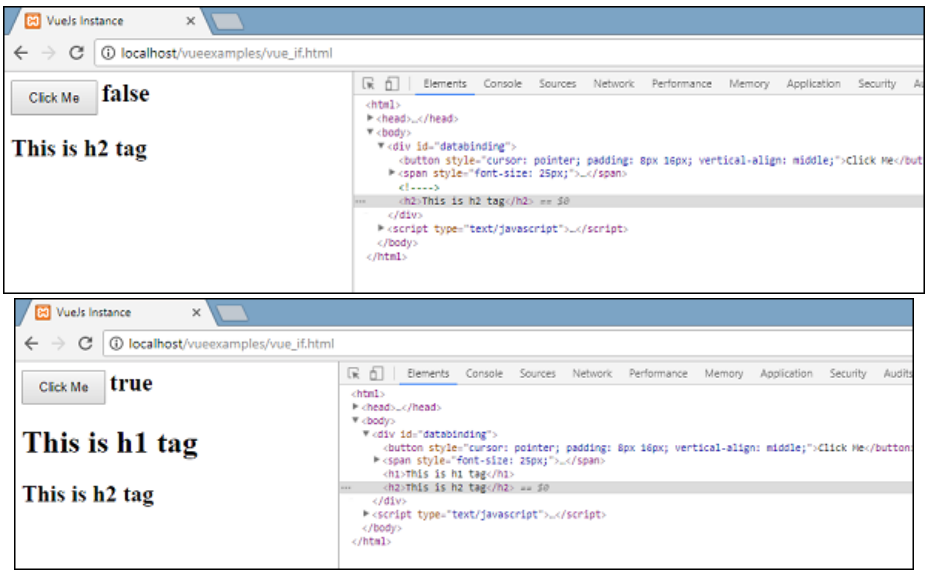
```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "databinding">
      <button v-on:click = "showdata" v-bind:style = "styleobj">Click Me</button>
      <span style = "font-size:25px;"><b>{{show}}</b></span>
      <h1 v-if = "show">This is h1 tag</h1>
      <h2>This is h2 tag</h2>
    </div>
    <script type = "text/javascript">
      const vm = new Vue({
        el: '#databinding',
        data: {
          show: true,
          styleobj: {
            backgroundColor: '#2196F3!important',
            cursor: 'pointer',
            padding: '8px 16px',
            verticalAlign: 'middle',
          }
        },
        methods : {
          showdata : function() {
            this.show = !this.show;
          }
        },
      });
    </script>
  </body>
</html>
```



Se crea un botón con dos etiquetas de encabezados con un mensaje en el interior de ellas. Una variable llamada **show** es declarada e inicializada con un valor **true**. Su valor se muestra cerca del botón. Cada vez que se clickea el botón se llama al método **showdata** que cambia el estado de dicha variable de false a true y viceversa

```
<button v-on:click = "showdata" v-bind:style = "styleobj">Click Me</button>
<h1 v-if = "show">This is h1 tag</h1>
```

Si el valor de la variable show es falso el tag h1 no se mostrará.



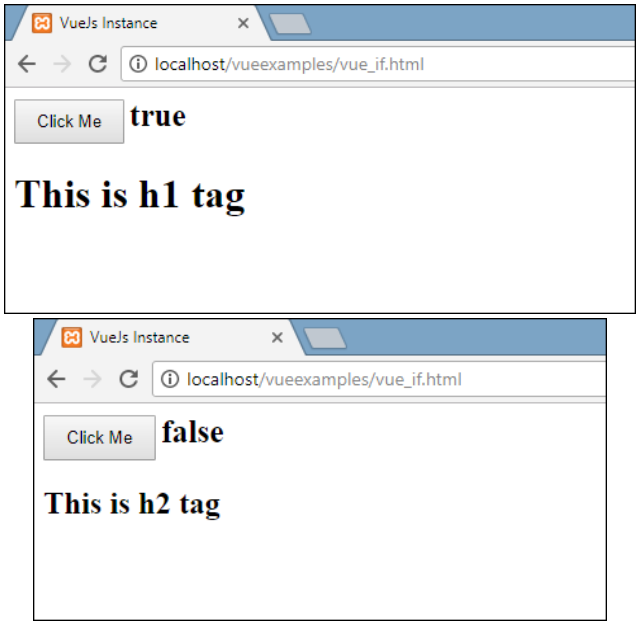
v-else

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "databinding">
      <button v-on:click = "showdata" v-bind:style = "styleobj">Click Me</button>
      <span style = "font-size:25px;"><b>{{show}}</b></span>
      <h1 v-if = "show">This is h1 tag</h1>
      <h2 v-else>This is h2 tag</h2>
    </div>
    <script type = "text/javascript">
      const vm = new Vue({
        el: '#databinding',
        data: {
          show: true,
          styleobj: {
            backgroundColor: '#2196F3!important',
            cursor: 'pointer',
            padding: '8px 16px',
            verticalAlign: 'middle',
          }
        },
        methods : {
          showdata : function() {
            this.show = !this.show;
          }
        },
      });
    </script>
  </body>
</html>
```

v-else se añade utilizando la siguiente sintaxis:

```
<h1 v-if = "show">This is h1 tag</h1>
<h2 v-else>This is h2 tag</h2>
```

Ahora, si **show** es true, se mostrará “This is h1 tag”, si es false, se mostrará “This is h2 tag”



v-show

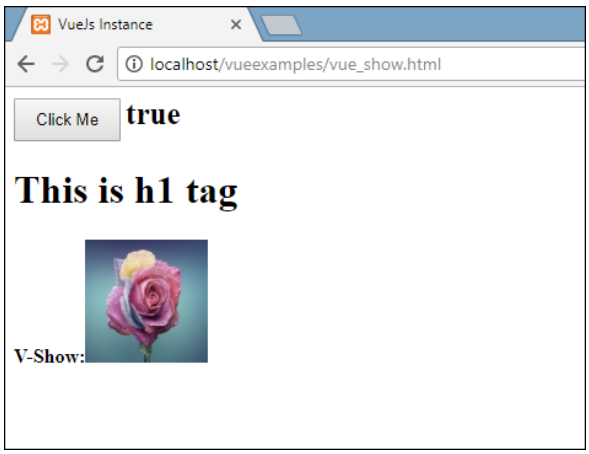
v-show se comporta igual que v-if. También muestra y oculta los elementos basado en una condición. La diferencia es que v-if los elimina del DOM si la condición es falsa y v-show sólo los oculta.

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "databinding">
      <button v-on:click = "showdata" v-bind:style = "styleobj">Click Me</button>
      <span style = "font-size:25px;"><b>{{show}}</b></span>
      <h1 v-if = "show">This is h1 tag</h1>
      <h2 v-else>This is h2 tag</h2>
      <div v-show = "show">
        <b>V-Show:</b>
        <img src = "images/img.jpg" width = "100" height = "100" />
      </div>
    </div>
    <script type = "text/javascript">
      const vm = new Vue({
        el: '#databinding',
        data: {
          show: true,
          styleobj: {
            backgroundColor: '#2196F3!important',
            cursor: 'pointer',
            padding: '8px 16px',
            verticalAlign: 'middle',
```

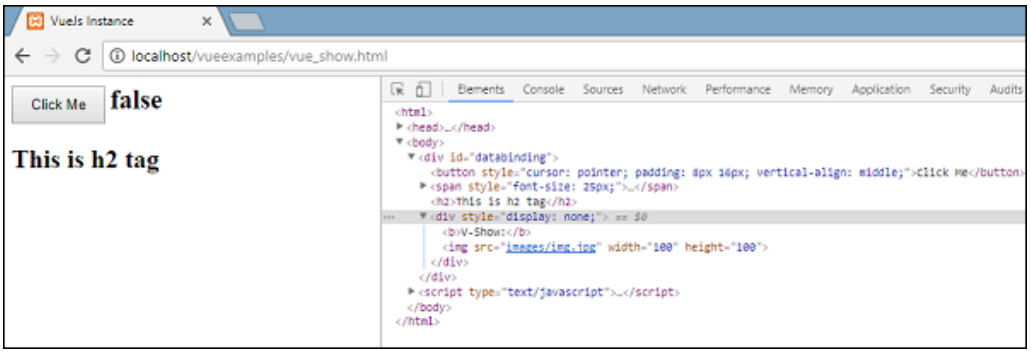
```
            }
        },
        methods : {
            showdata : function() {
                this.show = !this.show;
            }
        },
    });
    </script>
  </body>
</html>
```

Se asigna v-show a un elemento HTML de la siguiente manera:

```
<div v-show = "show"><b>V-Show:</b><img src = "images/img.jpg" width = "100" height = "100" /></div>
```



Si la variable **show** es true se muestra la imagen, si no, la misma se oculta. Si inspeccionamos el DOM vemos que sigue estando, nada más que está ocultada.



Renderizado de Listas

v-for

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "databinding">
      <input type = "text" v-on:keyup.enter = "showinputvalue"
        v-bind:style = "styleobj" placeholder = "Enter Fruits Names"/>
      <h1 v-if = "items.length>0">Display Fruits Name</h1>
      <ul>
        <li v-for = "a in items">{{a}}</li>
      </ul>
    </div>
    <script type = "text/javascript">
      const vm = new Vue({
        el: '#databinding',
        data: {
          items:[],
          styleobj: {
            width: "30%",
            padding: "12px 20px",
            margin: "8px 0",
            boxSizing: "border-box"
          }
        },
        methods : {
          showinputvalue : function(event) {
            this.items.push(event.target.value);
          }
        },
      });
    </script>
  </body>
</html>
```

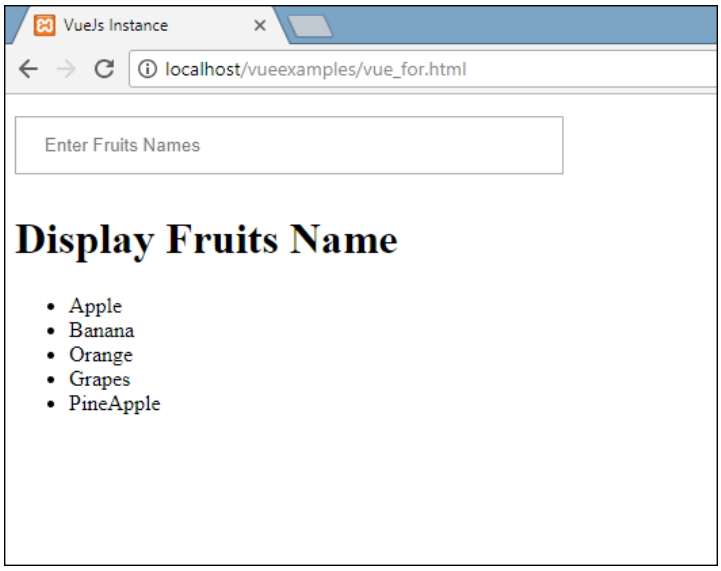
Se declara como array una variable llamada **items**. En la sección de métodos hay uno que se llama showinputvalue, el cual es asignado al textbox y recibe el nombre de las frutas. En el método dichos nombres serán agregados al array mediante el siguiente código.

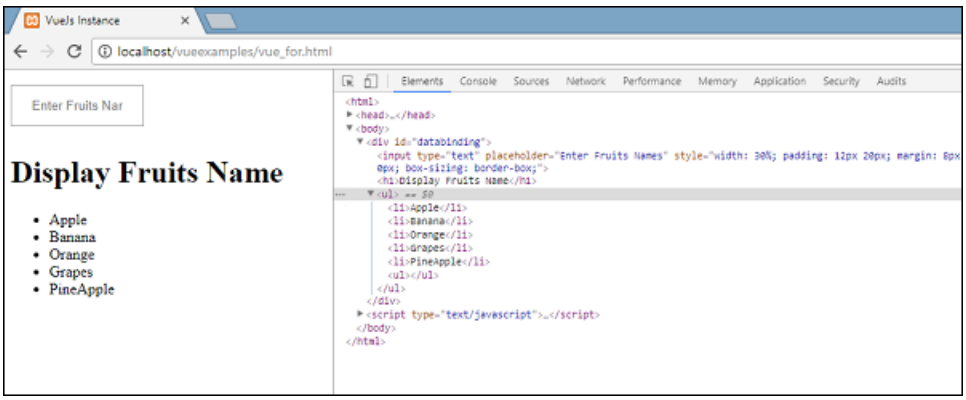
```
showinputvalue : function(event) {
  this.items.push(event.target.value);
}
```

Usaremos **v-for** para mostrar las frutas ingresadas al array mediante el siguiente código. v-for sirve para iterar sobre los elementos presentes en el array.

```
<ul>
  <li v-for = "a in items">{{a}}</li>
</ul>
```

Para iterar sobre el array en bucle utilizamos v-for = ”a in items” en donde **a** tendrá en su interior cada uno de los valores del array de forma secuencial y los mostrará mientras que queden ítems restantes que recorrer.





Si quisieramos mostrar el índice del array se debería utilizar el siguiente código.

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "databinding">
      <input type = "text" v-on:keyup.enter = "showinputvalue"
        v-bind:style = "styleobj" placeholder = "Enter Fruits Names"/>
      <h1 v-if = "items.length>0">Display Fruits Name</h1>
      <ul>
```

```
        <li v-for = "(a, index) in items">{{index}}--{{a}}</li>
      </ul>
    </div>
    <script type = "text/javascript">
      const vm = new Vue({
        el: '#databinding',
        data: {
          items:[],
          styleobj: {
            width: "30%",
            padding: "12px 20px",
            margin: "8px 0",
            boxSizing: "border-box"
          }
        },
        methods : {
          showinputvalue : function(event) {
            this.items.push(event.target.value);
          }
        },
      });
    </script>
  </body>
</html>
```

Para conseguir el índice se agrega entre paréntesis y separado por una coma de **a**, la palabra **index**, cuyo valor es posteriormente mostrado en {{index}}.

```
<li v-for = "(a, index) in items">{{index}}--{{a}}</li>
```

En **(a, index)**, **a** es el valor e **index** es la llave.