

¿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

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

HTML

or:

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

HTML

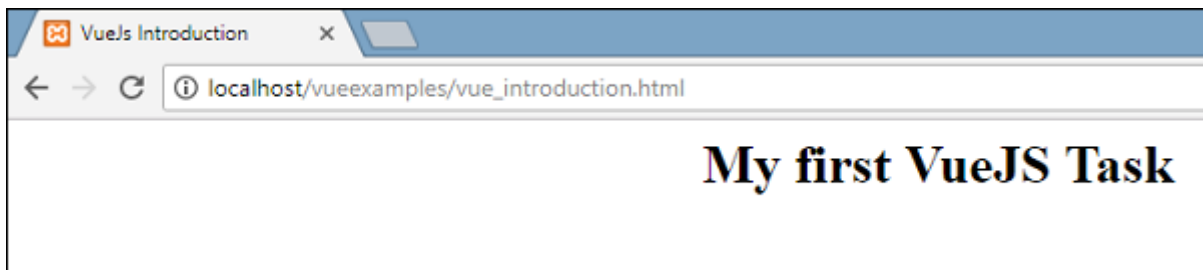
Ejemplo

```

<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.

```

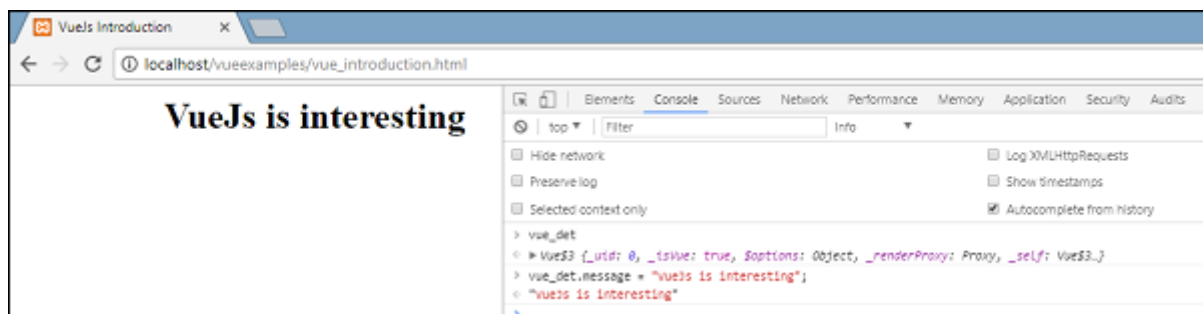
<div id = "intro">
  <h1>{{ message }}</h1>
</div>

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

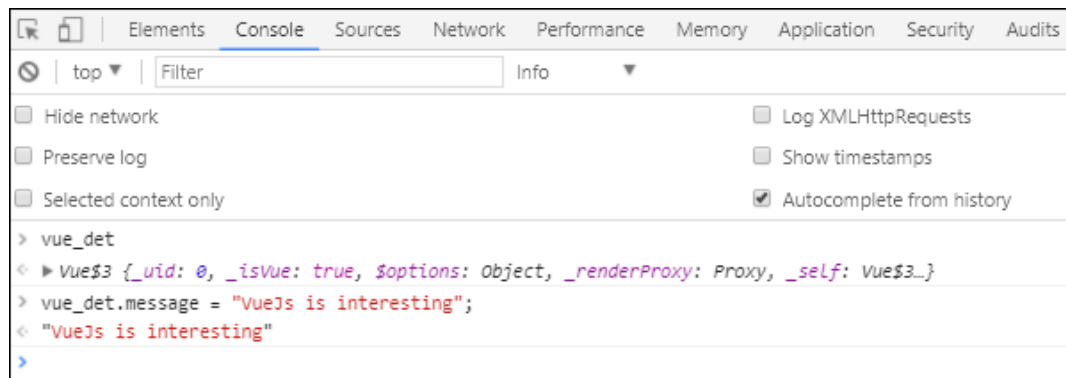
```

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:

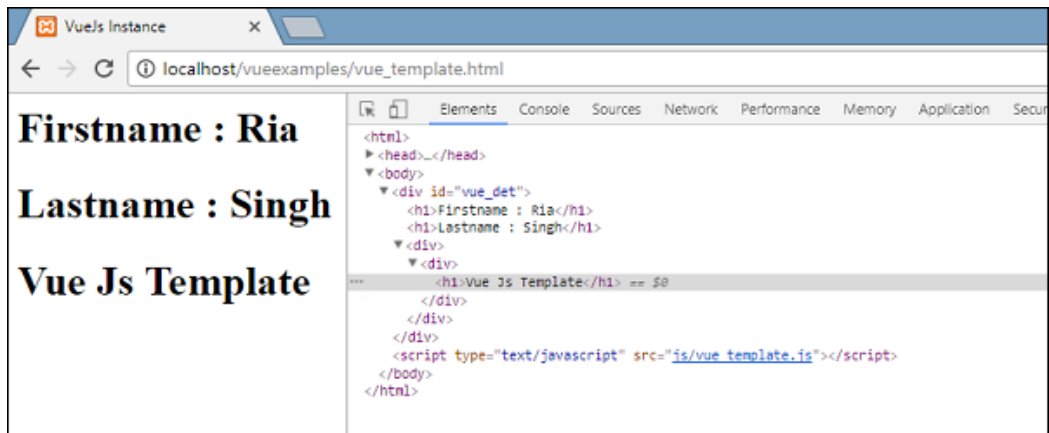


Si necesitamos insertar código HTML en el dom desde Vue.JS se utiliza la directiva **v-html**. Desde el momento que se usa Vue.JS ya sabe que tiene que mostrar dicho contenido como HTML en el navegador.

```
<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.

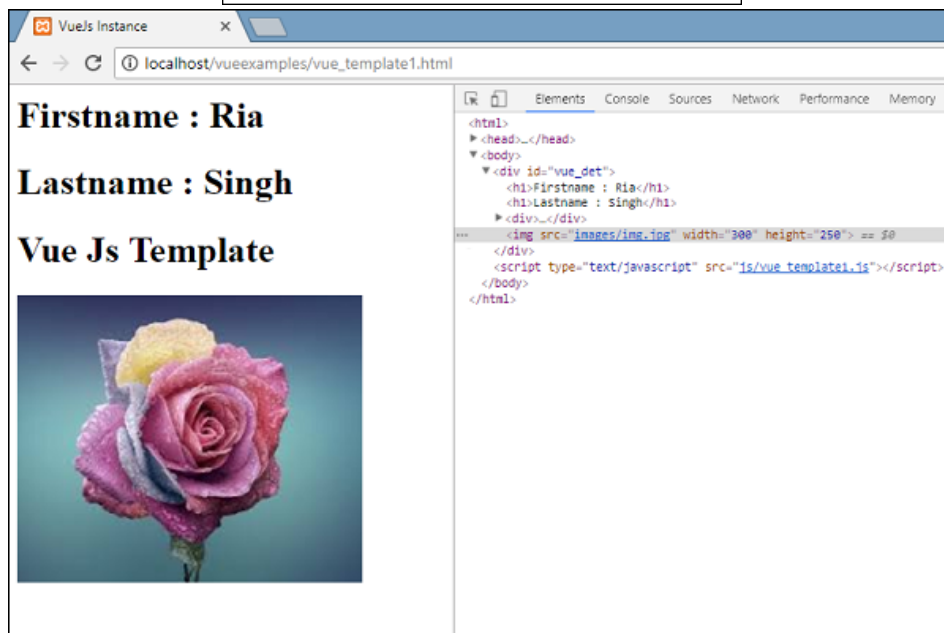
```

<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.

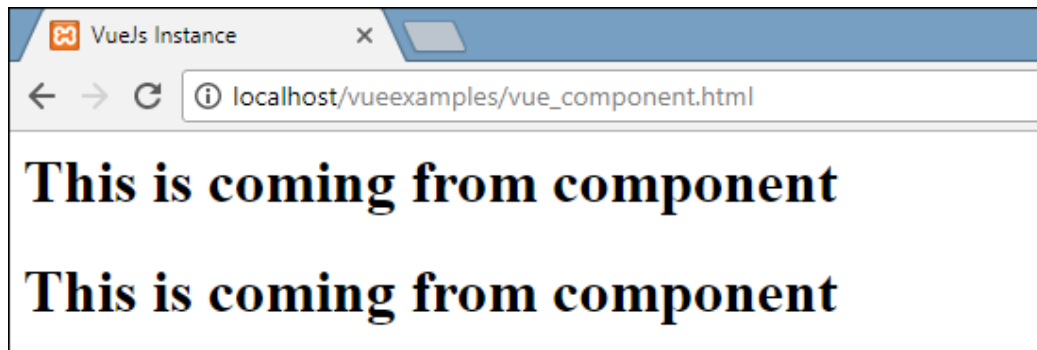
Para crear un componente se utiliza la siguiente sintaxis:

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

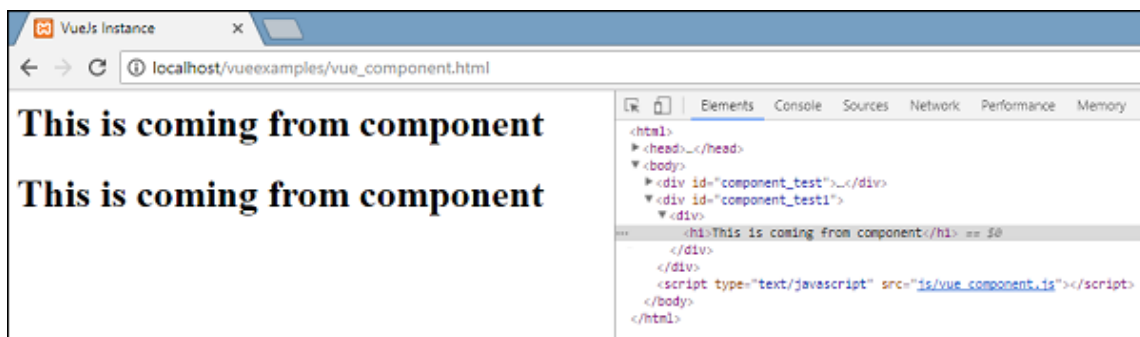
```
<div id = "component_test">
  <testcomponent></testcomponent>
</div>
<div id = "component_test1">
  <testcomponent></testcomponent>
</div>
```

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

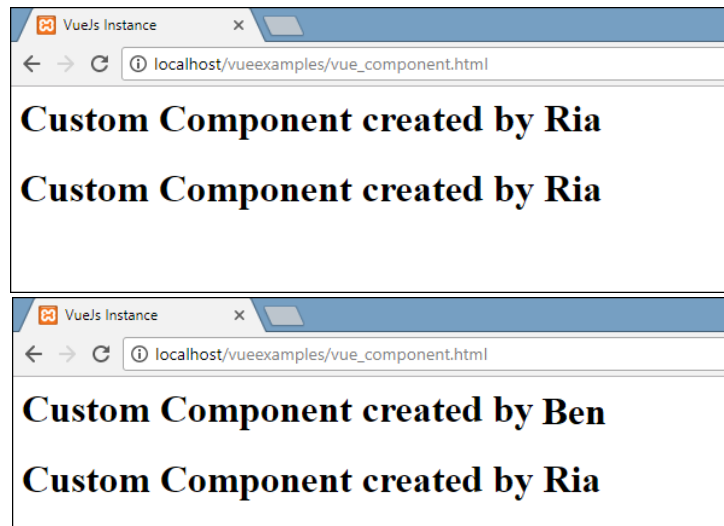
```
<html>
<head>
  <title>VueJs Instance</title>
  <script type = "text/javascript" src = "js/vue.js"></script>
</head>
<body>
  <div id = "component_test">
    <testcomponent></testcomponent>
  </div>
  <div id = "component_test1">
    <testcomponent></testcomponent>
  </div>
  <script type = "text/javascript"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

vue_component.js

```
Vue.component('testcomponent',{
  template : '<div v-on:mouseover = "changenname()" 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 `changenname()` 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 ligán 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>

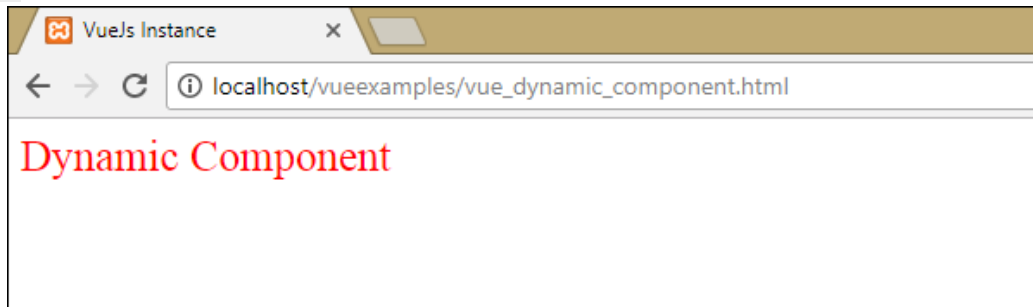
  <div id = "databinding">
    <component v-bind:is = "view"></component>
  </div>

  <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**.

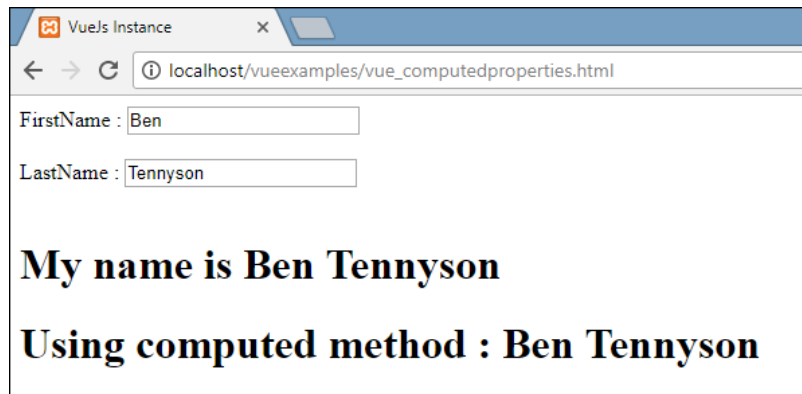
Posteriormente tenemos la propiedad computada getfullname, que devuelve el firstname y el lastname que ingresó el usuario concatenados.

```

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 exactamente “\$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>

```