# Componentes Vue

## Contenido

[Componentes Vue 1](#_Toc58345499)

## Video 1 – ¿Que son los componentes?

Los componentes son **instancias Vue** reutilizables, permiten **agrupar, encapsular y reutilizar** funcionalidad de cualquier tipo (desde un botón, una tabla y hasta una sección entera de una pagina web).

Un componente vendría a ser una especie de elemento HTML personalizado que se puede utilizar las veces que uno quiera simplemente llamándolo dentro del HTML.

Es importante definir nuestro componente antes de la instancia Vue principal, y para ello utilizamos el método ‘.component’ del objeto Vue. Luego veremos como crear componentes locales a una instancia, pero por ahora lo hacemos de esta manera.

A este método debemos pasarle dos parámetros:

* Primero el nombre del nuevo componente: Vue.component(‘gato’, \_).
* Segundo un objeto con diferentes “opciones”. Estas opciones son muy parecidas a las de la propia instancia Vue.

Recordar que no se debe usar el atributo ‘el: ‘ que utilizamos en una instancia Vue común, ya que no tiene sentido que atemos nuestro componente a un elemento en particular, sino todo lo contrario.

Para mostrarse se utiliza la propiedad ‘template:’ en la cual debemos definir el elemento que querramos (h1, img, p, div, lo que querramos).

Ahora ya expresado este componente y creando la instancia Vue, ya podemos “llamar” a nuestro componente simplemente como un elemento mas de HTML: <gato></gato>. A su vez a este componente lo podemos llamar las veces que querramos.

Un parámetro muy importante a utilizar dentro del objeto del componente es ‘data()’, que es un método el cual nos permite procesar alguna información y darnos un retorno. Este retorno es un objeto, porque asi nos aseguramos que cada uno de los componentes tiene un modelo independiente, se crea un contexto de ejecución nuevo (un universo nuevo) por cada una de las instancias nuevas de este componente, si no todo los componentes compartirían los mismos datos.

En nuestro ejemplo creamos dos variables a retornar que son aleatorias las cuales nos dan el ancho y alto de una foto.

Es una buena practica que, para no “ensuciar” con código ni interpolación el template, se definan propiedades computadas, de la misma manera que creamos una en una instancia Vue.

Otro punto importante es que podemos utilizar todas las directivas vue dentro del template que estamos definiendo, como son v-bind, v-on, etc. Lo que hay que tener en cuenta es que todo lo que se pueda sacar del template, es mejor afuera, ya que recordemos que tiene que ser lo mas “genérico” posible asi lo podemos reutilizar las veces que querramos.

<div id="app">

      <gato></gato>

      <gato></gato>

    </div>

Vue.component('gato', {

    template: `<img :src="urlCompleta"></img>`,

    data(){

        return{

            ancho: Math.floor(Math.random()\*600) + 100,

            alto: Math.floor(Math.random()\*600) + 100,

        }

    },

    computed: {

        urlCompleta(){

            return `https://placekitten.com/${this.ancho}/${this.alto}`;

        },

    },

});

const app = new Vue({

    el: '#app',

});

## Video 2 – Propiedades dinámicas

Ahora lo que vamos a hacer es creaer una estructura pero su información interna es dinámica. Vamos a crear para este ejemplo un botón con tres propiedades:

* Color de fondo: background-color
* Destino del enlace: href
* Texto del enlace o anchorText

Le daremos el nombre de ‘boton’ de manera practica. Y con template string creamos el contenido de este botón enlace. Es una buena practica crear primero la estructura estatica para ver como debería quedar para luego convertirla en dinámica. Para darle contenido variable lo haremos a través de las ‘props’ del componente, la cual en primera instancia la creamos como un arreglo con los nombres de esas variables a utilizar y que luego lo convertimos en un objeto para poder establecer los tipos de datos que esperamos recibir como a su vez establecer determinado valor predefinido (en este ejemplo el enlace).

Los valores de las propiedades los asociamos a traves de los atributos, pasandole en el maquetado los datos (que es lo que se ve dentro del código comentado en el HTML). Recordar hacer el bindeo de los datos, ya sea con v-bind o con las dobles llaves {{}}. Particularmente en este ejemplo tuvimos que hacer uso de string template dentro de la defincion del template del componente para ‘enlace’ lo cual no es recomendable, pero para este ejemplo resulta demostrativo.

Siempre es recomendable ver la guía y la documentación oficial para poder sacarle el mayor provecho a las props de los componentes.

Ahora, ¿que pasa cuando los datos provienen de un fetch o algún lugar remoto? Esa información supongamos que la tenemos preparada dentro de la instancia Vue, que es donde la vamos a llamar, y la tenemos que pasar a nuestro componente. Para este ejemplo Creamos un arreglo de objetos dentro de la instancia en el cual guardamos los 3 parametros que requerimos para un botón ( y lo llamamos ‘botones’). De esta manera utilizando un v-for y bindenando cada uno de estos argumentos vamos a lograr pasarlos al componente.

Vue.component("boton", {

  // props: ["color", "enlace", "anchorText"],

  props: {

    color: String,

    enlace: {

      type: String,

      default: "http://www.escuelavue.es",

    },

    anchorText: String,

  },

  template: `<a class="boton" :style="'background-color: ' + color" :href="enlace">{{anchorText}}</a>`,

});

const app = new Vue({

  el: "#app",

  data: {

    botones: [

      { enlace: "http://www.wmedia.es", anchorText: "Wmedia", color: "red" },

      {

        enlace: "http://www.escuelavue.es",

        anchorText: "Escuela Vue",

        color: "blue",

      },

      { enlace: "http://www.vuejs.org", anchorText: "Vue.js", color: "purple" },

    ],

  },

});

    <div id="app">

      <boton

        v-for="boton in botones"

        :color="boton.color"

        :enlace="boton.enlace"

        :anchor-text="boton.anchorText"

      />

      <!-- <boton color="blue" enlace="https://www.wmedia.es" anchor-text="Wmedia"></boton>

      <boton color="purple" enlace="https://www.escuelavue.es" anchor-text="Escuela Vue"></boton>

      <boton color="green" enlace="https://www.vuejs.org" anchor-text="VueJS"></boton>

      <boton color="black" anchor-text="Otro Enlace"></boton> -->

    </div>

## Video 3 – Componentes globales y locales

Ya hemos creado componentes globales que se pueden usar en cualquier lugar de nuestra aplicación, pero esto a veces no es conveniente por cuestiones de recursos.

Es bueno aclarar que podemos utilizar componentes globales uno dentro de otro, lo cual no presenta una gran ventaja pero es realizable:

Vue.component('destino', {

  template: `<span>Destino</span>`,

});

Vue.component('boton', {

  props: {

    color: String,

    enlace: {

      type: String,

      default: "http://www.escuelavue.es",

    },

    anchorText: String,

  },

  template: `<a class="boton" :style="'background-color: ' + color" :href="enlace">destino</a>`,

});

Los componentes locales se definen de manera diferente en cuanto a sintaxis y su uso. Primero debemos crear un objeto constante y su nombre será el que utilicemos como componente. Luego usaremos el atributo ‘template’ y en ehi agregamso el markup que querramos mostrar. Tambien se pueden agregar ‘props’y otros atributos que utilizan los componentes si quisiéramos, pero no puede faltar ‘template’.

Luego en nuestra instancia Vue debemos utilizar el atributo ‘components:’ el cual recibe un objeto con los diferentes nombres de los componentes que antes definimos. De esta manera, por ejemplo, podemos crear dos componentes locales (destino y botón) dentro de la instancia Vue (app), pero serán inexistentes uno para el otro, es por esto que si por ejemplo quisiéramos utilizar el componente ‘destino’ dentro del componente ‘boton’ debemos “agregar” el componente ‘destino’ dentro de ‘boton’ de manera similar a lo anterior: dentro del componente ‘boton’ utilizamos el atributo ‘components:’ el cual recibe un objeto con el o los nombres de componentes a utilizar.

De esta manera lo que hicimos fue definir 2 componententes locales y a su vez definir otro componente local dentro de otro.

const destino = {

  template: `<span>Destino</span>`,

};

const boton = {

  props: {

    color: String,

    enlace: {

      type: String,

      default: "http://www.escuelavue.es",

    },

    anchorText: String,

  },

  template: `<a class="boton" :style="'background-color: ' + color" :href="enlace">destino</a>`,

  components: {

    destino,

  },

};

const app = new Vue({

  el: "#app",

  data: {

    botones: [

      { enlace: "http://www.wmedia.es", anchorText: "Wmedia", color: "red" },

      {

        enlace: "http://www.escuelavue.es",

        anchorText: "Escuela Vue",

        color: "blue",

      },

      { enlace: "http://www.vuejs.org", anchorText: "Vue.js", color: "purple" },

    ],

  },

  components: {

    destino,

    boton,

  },

});

## Video 4 – Plantillas de componentes

Las plantillas de componentes son útiles cuando queremos dejar de construir un ‘template inline’ pero hay que tener cuidado ya que nos puede llegar a crear algunas inconsistencias. La forma de utilizarlo es muy sencila.

Primero debemos crear un componente que en este ejemplo va a ser local de una instancia Vue, pero en este caso en vez de definir el template desde aquí, lo haremos directamente desde el HTML. El componente nos quedara de la siguiente manera:

const pizza = {

  props: ["nombre", "imagen", "cantidad"],

  template: "#pizza-template",

  methods: {

    masPizza() {

      this.cantidad++;

    },

    menosPizza() {

      this.cantidad > 0 && this.cantidad--;

    },

  },

};

const app = new Vue({

  el: "#app",

  data: {

    pizzas: [

      {

        nombre: "Pizza de carne",

        imagen:

          "https://cocina-casera.com/wp-content/uploads/2011/12/pizaa-carne-receta.jpg",

        cantidad: 0,

      },

      {

        nombre: "Mini Pizza",

        imagen: "https://i.ytimg.com/vi/4wg09Xms\_wo/sddefault.jpg",

        cantidad: 0,

      },

      {

        nombre: "Pizza Pepperoni",

        imagen:

          "https://placeralplato.com/files/2016/01/Pizza-con-pepperoni-640x480.jpg",

        cantidad: 0,

      },

    ],

  },

  components: {

    pizza,

  },

});

Como vemos el ‘template:’ no esta definido dentro del componente, si no que hace referencia a un ‘id’ que definiremos (en este caso) en el HTML.

El HTML tampoco cambia, tratamos la información de la misma manera, salvo que debemos generar este nuevo template de la siguiente manera:

    <div id="app">

      <pizza

        v-for="pizza in pizzas"

        :nombre="pizza.nombre"

        :imagen="pizza.imagen"

        :cantidad="pizza.cantidad"

      ></pizza>

    </div>

    <script type="text/x-template" id="pizza-template">

      <article class="pizza">

        <h2 class="pizzas\_\_nombre">{{ nombre }}</h2>

        <section class="pizza\_\_cantidad">

          <button @click="masPizza">+</button>

          {{ cantidad }}

          <button @click="menosPizza">-</button>

        </section>

        <img :src="imagen" alt="Pizza" class="pizza\_\_imagen" />

      </article>

    </script>

Debemos utilizar un script de tipo ‘text/x-template’ y darle un ‘id’ que es el que utilizamos en el ‘template’ del componente.

Es importante recordar que cuando utilizamos este tipos de templates SIEMPRE debe haber un elemento root, es decir que no podemos poner dos elementos que esten a la misma “altura” o hermanos, y si lo único que queremos agregar son dos elementos debemos ponerlos al menos dentro de un <div>

De esta manera estamos agregando y quitando “pizzas” en las propiedades locales del componentes, si inspeccionamos el elemento dentro del <root> (ósea cada <pizza>) vemos como incrementa o decrementa la cantidad de cada una, pero sin embargo si observamos la instancia principal <root> en el primer elemento si vemos la cantidad nos va a mostrar cero. Entonces ¿cómo actualizamos desde el componente el modelo de la instancia padre (de este componente, porque es un componente local de esta instancia)? Se hace emitiendo **EVENTOS PERSONALIZADOS** desde “abajo” hacia “arriba”.

## Video 5 – Eventos personalizados

## Video 6 – Componentes dentro de componentes