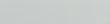


ADM

Dudas?



ADM

Dudas?



Componentes

- Como la mayoría de los frameworks actuales, el desarrollo de las funcionalidades de una app o sistema se basa en el concepto de componentes
- Los componentes son una de las características más destacadas de Vue.js.
- Un componente es una instancia reutilizable
- Permiten extender elementos HTML básicos para encapsular código
- Son elementos personalizados a los que el compilador de Vue.js asociará un comportamiento específico.

VueJS

- Un **componente** es un elemento de la interfaz del programa que tiene su propio **contenido** (ej: HTML), sus propios **datos** y su propio **comportamiento**.
- Para usar un componente tenemos que **registerarlo** primero.
- Una de las formas de **registerar** un componente es usar el **método Vue.component** y pasar la **etiqueta** y el **constructor**.
- La etiqueta será el **nombre del Componente** y el **constructor** las **opciones del mismo**.

VueJS

- Recibe un *string con el nombre de nuestro componente*, que será el *nombre que tendrá el elemento en el DOM*, y un *objeto de configuración* que contendrá el nombre de las *props*, el *template*, datos, métodos, etc

```
Vue.component('mi-componente', {  
    //opciones  
})
```

- Se recomienda usar la nomenclatura "kebab-case" que es todo en minúsculas, separado con "-".
- Adicionalmente, Vue recomienda que todos los componentes que creen tengan al menos 2 palabras.



VueJS

- Esto se debe a que en HTML las etiquetas van a ser de una sola palabra.
- Al crear nuestros componentes de al menos 2, nos aseguramos evitar cualquier conflicto a futuro.
- Una vez registrado, un componente puede ser utilizado en el html como un elemento personalizado `<mi-componente></mi-componente>`

```
<div class="contenedor">  
  <mi-componente></mi-componente>  
</div>
```

VueJS

- Los **componentes** van a tener, como mínimo, una propiedad "template" que contenga el template del componente en **HTML**, incluyendo las directivas de Vue u otros componentes que use.
- *Una vez registrado el componente debemos crear la instancia principal de vue*

```
Vue.component('mi-componente', {  
  template: '<div><h1>Esto es un mensaje desde un componente</h1></div>'  
});  
  
var app= new Vue({  
  el: ".contenedor",  
})
```

VueJS

```
▼ <body>
  ▼ <div class="contenedor">
    ▼ <div>
      <h1>Esto es un mensaje desde un componente</h1> == $0
    </div>
```

- Si examinamos el código de la consola, renderiza el componente como si estuviera escrito directamente en el HTML.
- Desde la opción de vue Devtools, en la consola:

```
▼ <root>
  <mi-componente> = $vm0
```

- El componente es hijo de la instancia principal

VueJS

- Estos componentes se registran a nivel global, por lo tanto hay que tener presente:
 - Todos los componentes deben tener un nombre único.
 - Todos los componentes están disponibles en todas partes de nuestro código Javascript

Propiedad template

- Nos permite determinar un **bloque de código HTML** con las funcionalidades que nosotros le asignemos.
- Dentro de todas las formas que se pueden implementar vamos a mencionar 2.

VueJS

- 1) Podemos crear el template de un componente con **comillas simples o dobles**:

```
Vue.component('mi-componente', {  
  template: '<div><h1>Dato desde un componente</h1></div>',  
})
```

- No permite el uso de **enter** ni **tabular** el código. Es decir todo es una sola línea continua, sino arrojará error de consola.
- 2) El template HTML del componente es un **string delimitado por backticks (`)**, llamadas **comillas obliquas** que permite insertar enters e indentación.

Vuejs

- Este tipo de comilla dependiendo de la configuración del teclado suele aparecer en la tecla donde aparece la llave y el corchete de cierre.
- Con el atajo Ctrl + Alt + } se dispara, si no con el atajo alt+96
- Este tipo de sintaxis forma parte de ES6

template:

```
<div>
```

```
    <p>El contenido se puede escribir usando enter y tabulacion del  
    código, Mucho mas prolijo</p>
```

```
</div>,
```

VueJS

- Tiene una restricción: Debe tener un único elemento raíz. Caso contrario arroja error de consola

✖ ➔ [Vue warn]: Error compiling template: [vue.js:634](#)

Component template should contain exactly one root element. If you are using v-if on multiple elements, use v-else-if to chain them instead.

```
1 | <p>Sumar :</p>
2 |           <button v-on:click="sumar++"> {{ sumar }}</button>
```

Propiedad data

- La opción ***data*** de un componente debe ser una función, de modo que cada instancia pueda mantener una copia independiente del objeto de datos devuelto:

VueJS

```
Vue.component('btn-contador', {  
  data: function () {  
    return {  
      sumar: 0,  
    }  
  },  
  template: `<div>  
    <p>Sumar : {{ sumar }}</p>  
    <button v-on:click="sumar++">Dale Sumar </button>  
  </div>`  
})
```

La función debe retornar un objeto.,

Si no le pasaramos una función, y solo fuera un objeto, cada instancia del componente compartirá esta propiedad.

index:

```
<btn-contador></btn-contador>
```

Browser:

Sumar : 2

Dale Sumar

VueJS

- Los componentes que creamos son reusables, por lo tanto podemos usarlos en el html varias veces

Sumar : 5

Dale Sumar

Sumar : 4

Dale Sumar

- *Cada uno mantiene su propio contador por separado.*

- *Cada vez que usamos un componente, se crea una nueva instancia del mismo.*

Atributo Props

- Sirven para pasar parámetros o información al propio web component para poder personalizarlo y ajustarlo dependiendo de las necesidades.

VueJS

- El **objetivo** de crear un componente es recibir información
- Las **props** son **atributos** personalizados que podemos **registrar** en un componente.
- *Cuando se pasa un valor a un atributo prop, se convierte en una propiedad en esa instancia de componente*
- Un componente puede **tener varias props**, y se puede pasar cualquier valor a cualquier prop de forma predeterminada.
- Es una buena practica escribir las props con **camelCase**

VueJS

```
Vue.component('componente-props',{
  data:function(){
    return {
      edad: 34
    }
  },
  props: [ ' nombre '],
  template: `<div>
    <h1>{{ nombre }} : {{ edad }}</h1>
  </div>`
})
```

La variable nombre está declarada en la instancia principal

```
<componente-props v-bind:nombre="nombre"></componente-props>
```

- Esto nos permite que nuestro componente muestre el nombre declarado en la instancia.

VueJS

- En caso de querer **generar componentes** de manera **dinámica**
- Es probable que estos datos vengan desde algún lugar.
- En nuestro caso tenemos creado desde la instancia un array de objetos llamado **articulos**.
- Cada articulo es un object y tiene varias propiedades: **id**, **titulo**, **favorito**, **texto**, **img**, **alt**
- Desde nuestro componente, llamado **componente-multilinea** vamos a pasar las **propiedades hacia el atributo props como un array**

VueJS

```
Vue.component('componente-multilinea', {  
  props: ['titulo', 'texto', 'img', 'id', 'alt'],  
  template: `  
    <div class="articulos">  
      <h1>{{ titulo }}</h1>  
      <p>{{ texto }}</p>  
      <img v-bind:src='img' v-bind:alt='alt' />  
    </div>  
  `})
```

- Desde nuestra **vista** vamos a tener que iterar el array articulos y vamos a generar por cada uno, un **componente** con **datos distintos**.

```
<componente-multilinea v-for="x in articulos"
```

VueJS

- Con el siguiente código vamos ir repasando algunas cosas que ya vimos, e integrándolas al componente
- Por cada iteración vamos crear un **atributo** y mostrar la propiedad del objeto que deseamos pasarle al componente.

```
<componente-multilinea v-for="x in articulos" v-if="x.favorito === true"
  v-bind:key="x.id"
  v-bind:class="x.id %2==1 ? 'lavanda' : 'coral'"
  v-bind:titulo="x.titulo"
  v-bind:id="x.id"
  v-bind:texto="x.texto"
  v-bind:img="x.img">
</componente-multilinea>
```

- Por medio del atributo **key**, *facilitamos a vue el trabajo de la iteración ante modificaciones en los elementos.*

VueJS

- Con **v-if** evaluará si el valor de favorito es *true*, caso contrario lo retirará del dom el artículo

v-if="x.favorito === true"

- Con **v-bind:class** vamos a manipular la aplicación de clases para cada componente dependiendo de si su id es par o impar.

v-bind:class="x.id %2==1 ? 'lavanda' : 'coral'"

- Si el operador de módulo nos devuelve 1, el **id** del objeto es **impar** se aplicará la clase de css **lavanda**, caso contrario, tendrá la clase **coral**
- El componente mostrará el **título, el texto y la imagen**

VueJS

Jardín floreciente en otoño

Plantas que florecen en otoño de variados colores, super resistentes como el Crisantemo, Zinnia, Geranio, nos ofrecen variados colores para alegrar la casa



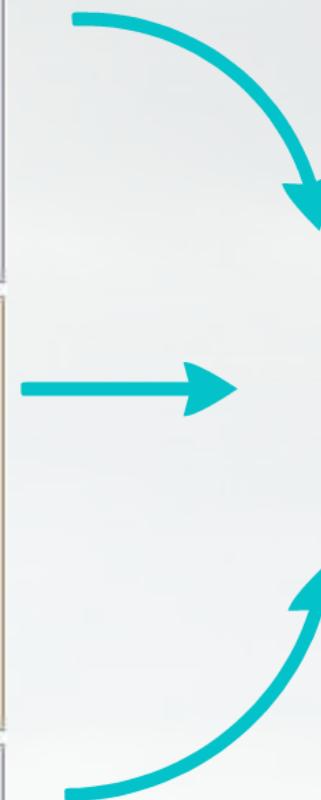
Suculentas y su cambio de color



Las suculentas que suelen cambiar de color dependen fundamentalmente del clima y el estrés a la cual se somete a la planta. Factores como el grado de luz, frío o calor puede afectar su color, tal es el caso del Aeonium, las Echeverias, también algunas cactus.

Jardines verticales

La tendencia en medio de la pandemia es buscar verde y sol para tratar de sobrevivir al encierro. Para aquellos que no tienen suficiente espacio, la opción vertical es una gran opción para disfrutar.



Componentes

VueJS

- Desde la consola, el panel de vuejs podremos ver cada componente creado y las respectivas propiedades de cada elemento.

Filter components

<root>

- <componente-props> = \$vm3
- <componente-multilinea key=1> = \$vm0
- <componente-multilinea key=2> = \$vm2
- <componente-multilinea key=3> = \$vm1

<componente-multilinea> Filter inspected data

props

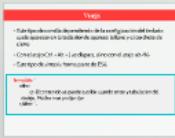
- `id: 1`
- `img: "img/jardin_otono.jpg"`
- `texto: "Plantas que florecen en otoño de varios"`
- `titulo: "Jardin floreciente en otoño"`

Dudas?



ADM

Dudas?



Prezi