

# 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**.
- Estos componentes pueden ser simples, como un botón o un cuadro de texto, o complejos, como un formulario completo o una lista de elementos.
- Para usar un componente tenemos que **registroarlo** primero.

En el archivo app.js :

- 1º Declaramos la instancia principal,
- 2º Definimos los componentes que necesitemos.
- 3º Estableceremos en qué elemento del HTML montamos la instancia de Vue.

# VueJS

- Para nombrar nuestros componentes, seguimos convenciones de nomenclatura :
  - PascalCase para nombres de componentes de una sola palabra : "MiComponente"
  - kebab-case para nombres de componentes de múltiples palabras "mi-componente"
- No podemos nombrar un componente con el mismo nombre de una etiqueta HTML existente. Esto arroja error en consola



The terminal window is titled "VueJS". The code in the terminal is as follows:

```
app.js
1 //Primero se crea la instancia de Vue
2 const app = Vue.createApp({
3   data() {
4     return {
5       titulo:"Desde la instancia root",
6     }
7   },
8 });
9
10 //Luego se registran los componentes globalmente
11 app.component('mi-componente', {
12   data:function(){
13     return {
14       salud:"Hola Chicos",
15     }
16   },
17   template:'<div><h1>Esto es un mensaje desde un componente
18 </h1><p>{salud}</p></div>',
19 });
20 //Por ultimo,se monta la instancia de Vue en el elemento con
21 //clase "contenedor"
22 app.mount('.contenedor');
```

# VueJS

```
app.js

1 //Primero se crea la instancia de Vue
2 const app = Vue.createApp({
3     data() {
4         return {
5             titulo:"Desde la instancia root",
6         }
7     }
8 });
9
10 //Luego se registran los componentes globalmente
11 app.component('mi-componente', {
12     data:function(){
13         return {
14             saludo:"Hola Chicos",
15         }
16     },
17     template:'<div><h1>Esto es un mensaje desde un componente
</h1><p>{{saludo}}</p></div>',
18 });
19 //Por último, se monta la instancia de Vue en el elemento con
//clase "contenedor"
20 app.mount('.contenedor');
```

# VueJS

- Al **definir un componente** creamos un **objeto de configuración** que contiene todas las opciones necesarias, como el nombre del componente, las props, el template, los datos, los métodos, etc.
- 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.

app.js

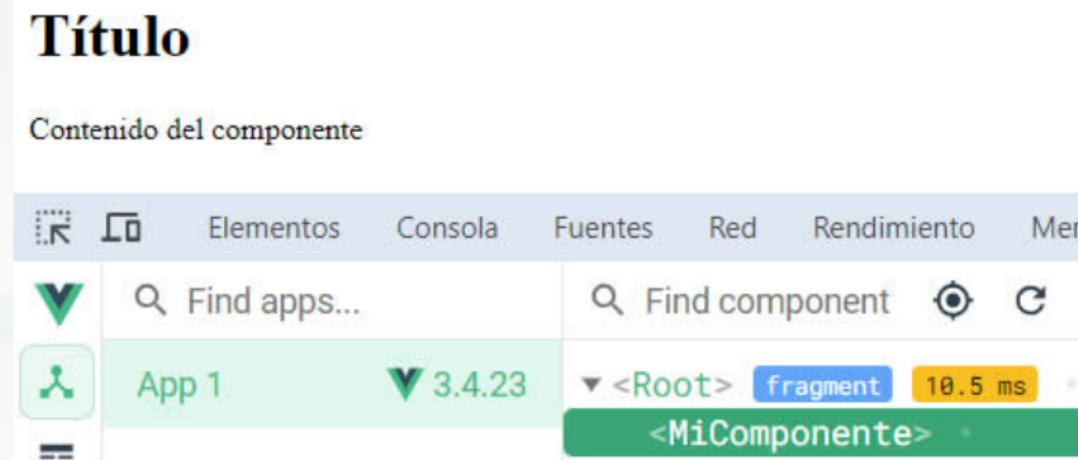
```
1 app.component('mi-componente', {  
2   template:  
3     <div>  
4       <h1>Título</h1>  
5       <p>Contenido del componente</p>  
6     </div>  
7   } );  
8 });
```

# VueJS

- Si examinamos el código de la consola, renderiza el componente como si estuviera escrito directamente en el HTML.

```
▼ <div>
  <h1>Título </h1> == $0
  <p>Contenido del componente</p>
</div>
```

- Desde la opción de vue Devtools, en la consola:



- 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**:

```
app.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

## 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

```
app.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

# VueJS

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

La funcion debe retornar un objeto.,

*Si no le pasaramos una funcion, 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

```
app.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

```
app.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.

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

- 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?

