Clase 04

Diseño y Programación Web

Materia:

Aplicaciones para Dispositivos Móviles

Docente contenidista: GARCIA, Mabel

Revisión: Coordinación

Contenido

Clase 4

¡Te damos la bienvenida a la materia **Aplicaciones para Dispositivos Móviles**!

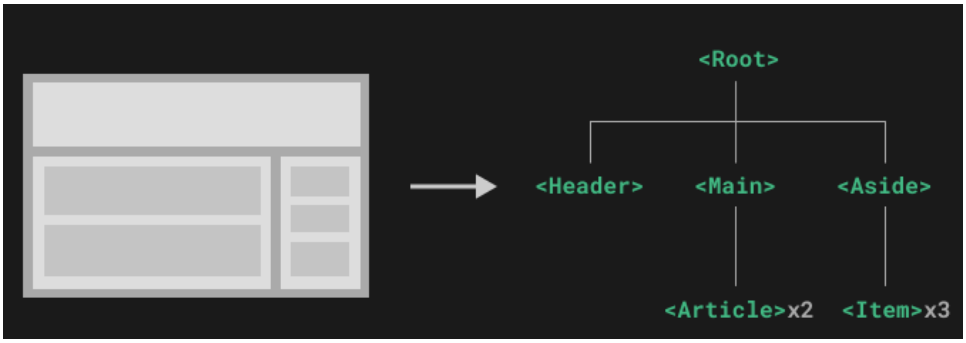
En esta clase vamos a ver los siguientes temas:

- Creación de componentes.
- Propiedad template.
- Propiedad data.
- Atributo Props.
- Componentes dinámicos.

Creación de Componentes

Como ocurre en la mayoría de los frameworks actuales, el desarrollo de las funcionalidades de una aplicación o sistema se basa en el concepto de componentes.

Los componentes son una parte fundamental en el ecosistema de Vue.js. Estos permiten dividir la interfaz de usuario en piezas más pequeñas y reutilizables, lo que facilita la construcción y mantenimiento de aplicaciones web complejas.



Un componente es una instancia de Vue con opciones específicas. Representa una parte de la interfaz de usuario que puede contener su propio estado, comportamiento y HTML.

Para que podamos entenderlo mejor, estos componentes pueden ser simples, como un botón o un cuadro de texto, o complejos, como un formulario completo o una lista de elementos.

Antes de usar un componente, es necesario registrarlo. En nuestro archivo app.js declaramos la instancia principal, y luego los componentes que necesitemos. Por último, estableceremos en qué elemento del HTML montamos la instancia de Vue.

Cuando nombramos nuestros componentes, seguimos convenciones de nomenclatura como PascalCase para nombres de componentes de una sola palabra y kebab-case para nombres de componentes de múltiples palabras. Por ejemplo:

- PascalCase (una sola palabra): MiComponente.
- kebab-case (múltiples palabras): mi-componente.

Hay que tener en cuenta que no podemos nombrar un componente con el mismo nombre de una etiqueta HTML. Es decir, no podemos crear un componente llamado "form" porque ese nombre ya está reservado para su uso en HTML. Además, esto provocará un error de consola, por consiguiente, no renderiza el componente.

app.js

```
//Primero se crea la instancia de Vue
const app = Vue.createApp({
  data() {
    return {
      titulo:"Desde la instancia root",
    }
  }
});

//Luego se registran los componentes globalmente
app.component('mi-componente', {
  data:function(){
    return {
      saludo:"Hola Chicos",
    }
  },
  template:'<div><h1>Esto es un mensaje desde un componente
</h1><p>{{saludo}}</p></div>',
});
//Por último,se monta la instancia de Vue en el elemento con
clase "contenedor"
app.mount('.contenedor');
```

Al definir un componente en Vue, creamos un objeto de configuración del componente 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, que se puede incluir en cualquier lugar donde sea necesario en la interfaz de usuario. Por ejemplo:

index.html

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

Propiedad template

La propiedad "template" es una de las propiedades básicas para un componente que contiene el marcado HTML definiendo la apariencia y estructura del componente. Dentro del template, podemos usar todas las características de HTML, así como las directivas y sintaxis específicas de Vue.

Por lo tanto, esta propiedad nos permitirá **definir el aspecto visual**, es decir, cómo se verá el componente cuando se renderiza en la página web. Puede contener **elementos HTML como divs, encabezados, párrafos, botones, etc., así como atributos y clases para aplicar estilos CSS.**

Dentro del template podemos **usar directivas de Vue, como v-bind, v-for, v-if, etc., para realizar manipulaciones dinámicas en el DOM**, además usar **interpolaciones para mostrar datos dinámicos del componente.**

Además de las etiquetas HTML que usamos habitualmente, el template también **puede incluir otros componentes** de Vue. Esto **permite la creación de componentes compuestos, donde un componente puede contener y reutilizar otros componentes dentro de su propio template.**

Cuando definimos el contenido del template, éste se incluye entre comillas. Es importante aclarar que si usamos comillas simples o dobles, no podemos realizar saltos de línea ni utilizar tabulaciones en nuestro código. Esto puede dificultar la legibilidad y mantenimiento del código, por lo que no es recomendable.

Se recomienda el uso de comillas oblicuas, también llamadas backticks para que sea más legible nuestro código ya que permite usar enter, tabulaciones e indentaciones. 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.

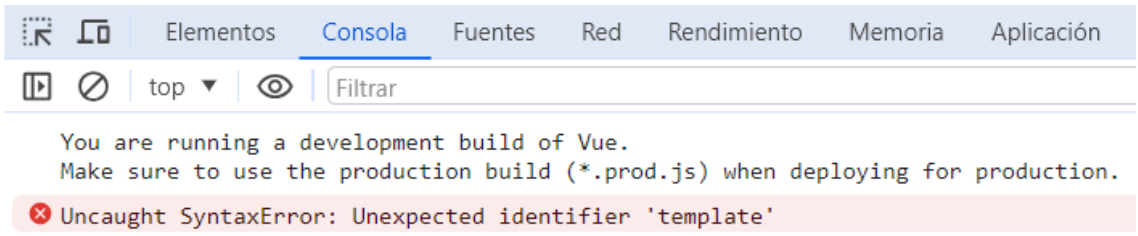
Propiedad data

La propiedad **data** en un componente de Vue es una función que devuelve un objeto, similar a cómo se usa en la instancia principal de Vue. Al devolver un objeto, Vue puede crear una instancia única de este objeto para cada instancia del componente, lo que evita problemas de compartición de datos entre diferentes instancias de componentes.

Este objeto contiene **los datos locales específicos del componente**. **Cada instancia del componente tendrá su propia copia de estos datos**, lo que permite que los componentes sean independientes y reutilizables.

Al estar creados de esta forma global, implica que el componente estará disponible para ser utilizado en cualquier parte de nuestro sistema donde esté dentro del alcance de esta instancia de Vue. Esto significa que una vez registrado con **app.component()**, podemos incluir y utilizar el componente **btn-contador** en cualquier parte de nuestra aplicación.

En el código tenemos inicializada la propiedad **sumar** con valor **0**. Este es el número que se mostrará en el contador. Luego de la llave de cierre de la propiedad "**data"** debemos poner una "coma", caso contrario arrojará error en la consola.



Dentro del template, se usa la interpolación **{{ sumar }}** para mostrar el valor actual de sumar en un párrafo. También hay dos botones que permiten al usuario aumentar o disminuir el valor de sumar cuando se hace click en ellos.

Los botones tienen las directivas **"v-on"** que escuchan los eventos de click. Cuando se hace click en el botón de sumar, se ejecuta la expresión **sumar++**, lo que aumenta el valor de sumar en 1. De manera similar, cuando se hace click en el botón de restar, se ejecuta sumar--, lo que disminuye el valor de sumar en 1.

Si tuviéramos una lógica más compleja, nos conviene crear una función dentro de los **methods** para por ejemplo, que no se pueda llegar a valores negativos o que al sumar tenga un límite, etc.

En el HTML, hemos insertado el nombre del componente registrado en diferentes secciones:

index.html

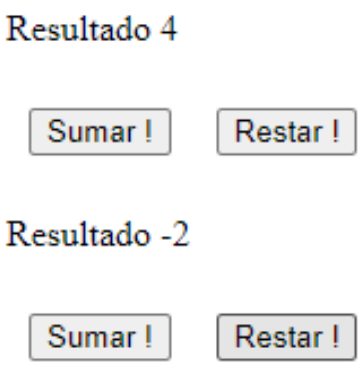
```
<div class="contenedor" >

  <section> <!--codigo html -->
    <btn-contador></btn-contador>
  </section>

  <section>  <!--codigo html -->
    <btn-contador></btn-contador>
  </section>

</div>
```

Cada sección representa un uso del componente **btn-contador** en diferentes partes del código HTML. Es importante destacar que estos son ejemplos simples para ilustrar cómo se puede utilizar el mismo componente en múltiples lugares de la página.

Es fundamental comprender que cada instancia del componente **btn-contador** mantiene su propio ámbito y datos independientes. Esto significa que las interacciones con un botón contador en una sección no afectarán el estado de otro botón contador en otra sección.

Es importante tener en cuenta que el ámbito de cada componente se limita al área del DOM donde se utiliza. Esto garantiza que los **datos y métodos del componente sean accesibles únicamente dentro de su alcance, lo que promueve la encapsulación y la reutilización del componente en la aplicación.**

En el navegador se renderizan ambos componentes y cada uno con sus resultados independientes.



Atributo Props

Las props en Vue.js son una característica fundamental que **permite la comunicación entre componentes**, permitiendo que **los datos se pasen desde el componente padre al hijo**. Esto hace que los componentes sean más flexibles y reutilizables al permitirles aceptar datos personalizados.

Por lo tanto, **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. El objetivo de crear un componente es recibir información.
Se definen como un **arreglo de cadenas de texto en la definición del componente**, y estas cadenas **representan los nombres de las props.**

Es importante tener en cuenta que las **props** son de **solo lectura**, lo que significa que si **necesitamos modificar el valor de una prop**, **se debe usar una propiedad de datos local en el componente.**

Un componente puede tener varias props y se puede pasar cualquier valor a cualquier prop de forma predeterminada.
Es una buena práctica escribir las props con **camelCase**.

app.js

```
// Creamos una instancia de Vue
const app = Vue.createApp({
// Definimos los datos en la instancia principal
  data() {
    return {
      nombre: "Pepe", // Propiedad que será pasada al componente
    }
  }
});

// Definimos un componente llamado 'componente-props'
app.component('componente-props', {
// Definimos datos locales del componente
  data() {
    return {
      edad: 34 // Dato propio del componente
    }
  },
// Definimos las props que este componente aceptará
  props: ['nombre'], // La propiedad 'nombre' será pasada al componente
// Definimos el template del componente
  template: `
    <div>
      <h1>{{ nombre }} : {{ edad }}</h1>
    </div>
  `
});
```

En este caso la variable **nombre** está declarada en la instancia principal.

En el index tendremos :

index.html

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

Esto nos permite que nuestro componente muestre el **nombre declarado en la instancia**. Las props permiten la comunicación de datos de un componente padre a un componente hijo, a través del "pasaje de props" (prop drilling).

Es importante destacar que las props se manejan de manera diferente a los atributos HTML regulares en Vue.

Desde la consola del navegador veremos la muestra de las props que recibe como hijo de la instancia principal.

Componentes dinámicos

Los componentes dinámicos en Vue son útiles en diversas situaciones, especialmente cuando necesitamos renderizar componentes de manera dinámica según ciertas condiciones o datos variables. Algunos escenarios donde los componentes dinámicos son útiles incluyen:

- **Listas de elementos**: Cuando necesitamos mostrar una lista de elementos donde **cada elemento es un componente distinto**. Por ejemplo, una lista de productos en un ecommerce, donde cada producto se muestra como un componente separado.
- **Menús o pestañas**: En aplicaciones con menús o pestañas donde cada opción requiere un componente diferente para mostrar contenido específico.
- **Renderización condicional**: Cuando queremos renderizar un componente basado en una condición o estado dinámico. Por ejemplo, mostrar un componente de bienvenida si el usuario está autenticado o un formulario de inicio de sesión si no lo está.
- **Componentes compuestos**: En la construcción de componentes más complejos que requieren la inclusión y organización de otros componentes más pequeños. Esto puede facilitar la modularidad y el mantenimiento del código.

Para ejemplificar, vamos a aprovechar la relación entre la instancia root como elemento padre que posee la información inicializada, con esto trabajaremos el paso de props y vamos a generar una serie de artículos para un blog, donde cada uno será un componente distinto.

En nuestro caso, hemos creado desde la instancia un array de objetos llamado **artículos**. Cada artículo es un **objeto** y tiene **varias propiedades: id, título, favorito, texto, img, alt.**

app.js

```
const app = Vue.createApp({
  data() {
    return {
     articulos: [ //se pasará al componente por props
          { id: 1,
            titulo: 'jardin floreciente en otoño',
            favorito: true,
            texto:"Plantas que florecen en otoño de varios colores, super resistentes como el Crisantemo, Zinnia, Geranio, nos ofrecen variados colores para alegrar la casa",
            img:'img/jardin_otono.jpg',
            alt:"Flores en otoño en el jardin"
          },
          { id: 2,
            titulo: 'suculentas y su cambio de color',
            favorito: true,
            texto:"Las suculentas que suelen cambiar de color dependen fundamentalmente del clima y el strees a la cual se somete a la planta. Factores como el grado de luz, frío o calor pueden afectar su color, tal es el caso del Aeonium, las Echeverias, también algunos cactus.",
            img:"img/suculentas.jpg",
            alt:"Suculentas estresadas"
          },
          { id: 3,
            titulo: 'jardines verticales',
            favorito: true,
            texto:"La tendencia en diseño de interiores es llevar la naturaleza a la vida diaria. Para aquellos que no tienen suficiente espacio, la opción vertical es una gran forma para disfrutarla",
            img: "img/vertical.jpg",
            alt:"Jardin vertical"
          },
          { id: 4,
            titulo: 'macetas de cerámica',
            favorito: false,
            texto: "Siempre pensando que no hay nada más lindo que ver una planta saludable, las posibilidades que nos ofrecen los distintos tipos de macetas y estilos, nos permiten elegir una temática particular, acorde a nuestro gusto.",
            img:"img/macetas.jpg",
            alt:"Macetas de cerámica"
            }
          ],
        }
    }
});
```

Nuestro **componente-dinámico** tendrá las **props** con formato de array. Luego, en el **template** del componente, utilizamos estas props para mostrar la información correspondiente en la interfaz del usuario, por medio de las **interpolaciones**:

app.js

```
app.component('componente-dinamico', {
  props:['titulo', 'texto', 'img', 'id', 'alt'],
    template:
    `<div class="articulos" >

        <h1>{{titulo}}</h1>

          <div class="d">
              <p>{{texto}}</p>
              <img v-bind:src="img" v-bind:alt="alt"/>
          </div>
     </div>`
});
```

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

index.html

```
<template v-for="x in articulos">
  <componente-dinamico 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"
    v-bind:alt="x.alt">
  </componente-dinamico>
</template>
```

Con el código anterior vamos ir repasando algunas cosas que ya vimos, e integrándolas al componente.

Cada componente estará envuelto por un template usando la directiva **v-for** para **recorrer el array de artículos.**

Dentro de este envoltorio, tendremos el componente con la directiva v-if para renderizar el componente **componente-dinámico** sólo si la condición **x.favorito === true se cumple**. Caso contrario, lo retirará del DOM el artículo.

Con la directiva **v-bind:key** le daremos a Vue **una clave única a cada iteración cuando se utiliza v-for**. Esto ayuda a Vue a rastrear la identidad de cada elemento y optimizar la actualización del DOM.

Para cambiar el aspecto visual afectamos con **v-bind:class** la aplicación de clases para cada componente **dependiendo de si su id es par o impar**. 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 CSS "coral"**.

Por último, usamos la directiva **v-bind** para **pasar propiedades dinámicamente al componente**. Estas props son : **título**, **id**, **texto**, **img**, y **alt**, del artículo actual **x**.

En el navegador veremos el resultado:

Pensemos que por cada artículo queremos ofrecerle al usuario un botón para que el usuario califique cada uno de ellos. Anteriormente ya habíamos hecho un componente **btn-contador** que podremos reutilizar en este ejemplo.

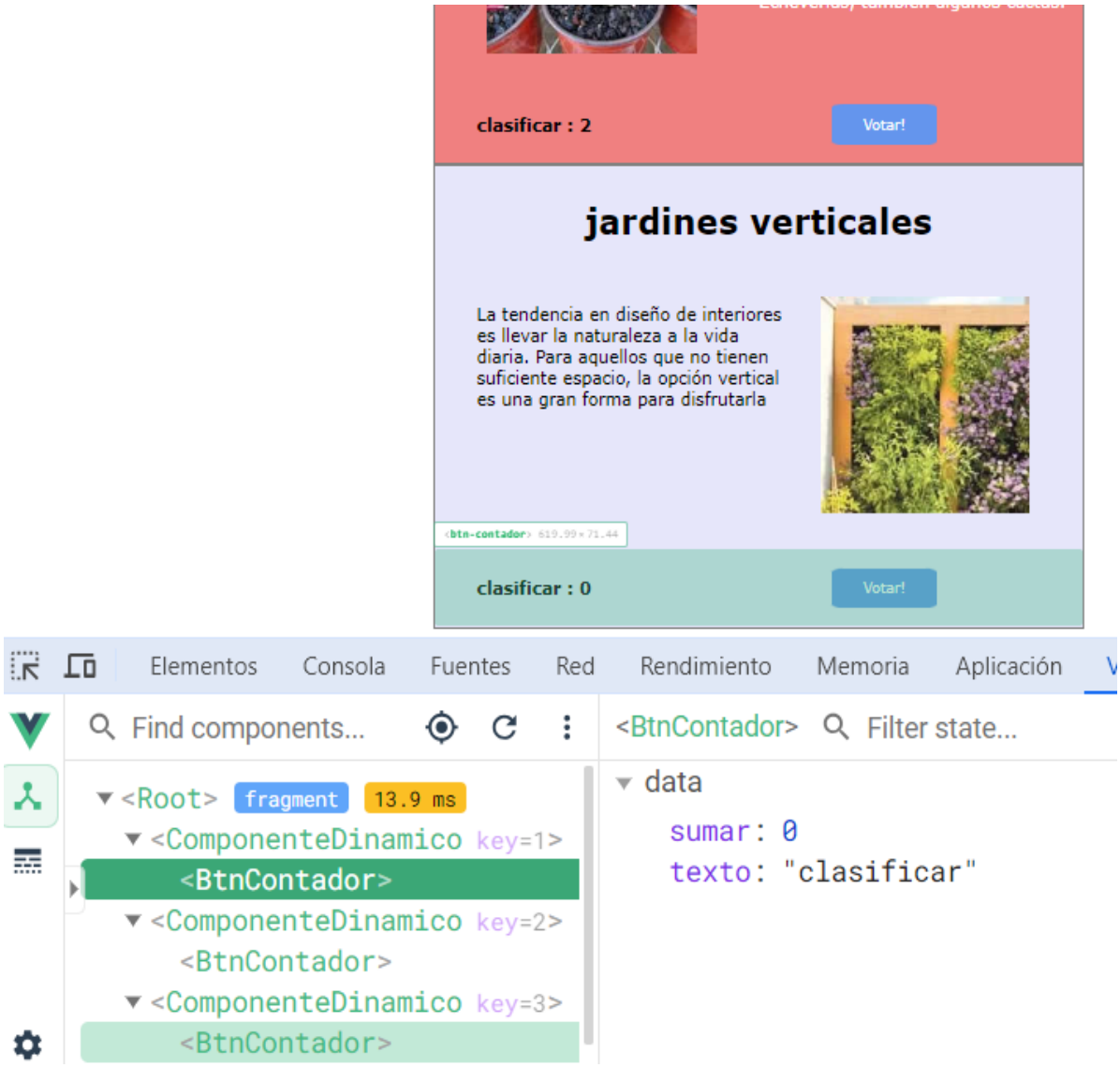
Con esto vamos a generar un **árbol de componentes más complejo**.
Dentro de la propiedad template del artículo insertamos el componente btn-contador.

Luego en la lógica tendremos que tener el código necesario para que este funcione y tenga sus propios datos, template con la estructura html y funcionalidad.

app.js

```
app.component('componente-dinamico', {
  props:['titulo', 'texto', 'img', 'id', 'alt'],
    template:
    `<div class="articulos" >

        <h1>{{titulo}}</h1>
        <div class="d">
          <p>{{texto}}</p>
          <img v-bind:src="img" v-bind:alt="alt"/>
        </div>

        <btn-contador></btn-contador>

     </div>`
});
app.component('btn-contador', {
 data: function () {
    return {
      sumar: 0,
      texto:"Clasificar"
    }
  },
  template:
    `<div class="voto">
       <p>{{texto}} : {{ sumar }}</p>
       <button v-on:click="sumar++">Votar!</button>
     </div>`
})
```

Si examinamos la consola veremos la división de responsabilidades y funcionalidades cuando usamos componentes.



Ya estamos en condiciones de empezar a desarrollar lógica y elaborar componentes. ¡Manos a la obra!.

Hemos llegado así al final de esta clase en la que vimos:

- Creación de componentes.
- Propiedad template.
- Propiedad data.
- Atributo Props.
- Componentes dinámicos.

Te esperamos en la **clase en vivo** de esta semana.
No olvides realizar el **desafío semanal**.

¡Hasta la próxima clase!

Bibliografía

Vue.js. (s.f.). Introducción a componentes. En Documentación Oficial de Vue.js. Recuperado de https://vuejs.org/guide/essentials/component-basics.html

Vue.js. (s.f.). Uso de Props. En Documentación Oficial de Vue.js. Recuperado de https://vuejs.org/guide/components/props.html