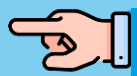
Manejo de eventos y reutilización de componentes

Comunicación entre componentes

{desafío} latam_

Implementar interacción en los elementos de una interfaz web utilizando manejo de eventos Vue para dar solución a un requerimiento y componentes reutilizables utilizando el framework Vue para el desarrollo de una aplicación web mantenible en el tiempo.

- Unidad 1: Introducción a Componentes Web y Vue Js
- Unidad 2: Binding de formularios
- Unidad 3: Templates y rendering en Vue
- Unidad 4: Manejo de eventos y reutilización de componentes
- Unidad 5: Consumo de datos desde una API REST



Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Implementa una aplicación web utilizando comunicación entre componentes padres e hijos para resolver un problema determinado.*

{desafío} latam_

¿Cómo accedemos al índice de cada iteración dentro de un v-for?

/* Comunicación entre componentes */

Emitir eventos a componente padre

comunicación entre componentes

Ya hemos aprendido anteriormente el uso de **props** y cómo éstos nos permiten entregarle datos de un componente padre a un componente hijo.

*Pero, ¿Qué sucede si necesitamos enviar información de un componente **hijo** a un componente **padre**?*

Para esto deberemos siempre comprender la jerarquía de los componentes y ocupar la emisión de eventos a través del método nativo **`$emit().`**

Emitir eventos a componente padre

`$emit`

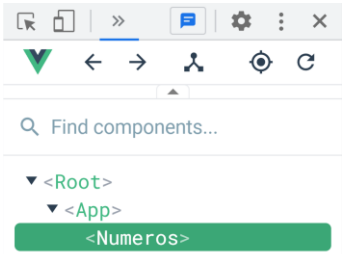
En cada uno de nuestros componentes disponemos de una gran biblioteca de **métodos** y **atributos** que Vue nos provee para realizar diferentes tareas.

el **`$emit`** nos ayudará a **emitirle** a un componente padre la notificación de que necesitamos que haga algo o que reciba algo.

Emitir eventos a componente padre

$emit

Los **$emit** siempre invocará el llamado de un evento a su componente padre.



Jerarquía:

Componente Padre: **App**
Componente hijo: **Numeros**

El término de **padre** e **hijo** se refiere a la jerarquía con la que vamos construyendo nuestro árbol de componentes.

De manera que si tuviéramos un componente **nieto** y éste hiciera uso del **$emit**, su abuelo no se enteraría de esta acción, solo su **padre.**

Ejercicio guiado 1

{desafío} latam_

Emitir eventos a componente padre

Ejemplo del $emit

En la presentación anterior vimos un ejemplo de una grilla de productos finalizando con un ejemplo de instrumentos musicales.

Desde cada componente hijo **(Instrumento)** procedamos a agregar un botón en su template que al ser clickeado, emita un **evento** al padre para que éste sea eliminado.

{desafío} latam_

Emitir eventos a componente padre

Ejemplo del $emit

El template del componente hijo nos quedaría así

```
<template>
 <div class="persona">
   <div :style="{ backgroundImage: `url(${foto})` }"
class="imagen"></div>
   <h2>{{ nombre }}</h2>
   <p>{{ descripcion }}</p>
   <button @click="$emit('eliminarInstrumento')">Eliminar</button>
 </div>
</template>
```

El **$emit** recibe como argumento un **String** con el nombre de un **evento** que declararemos ahora en el componente padre.

Batería electrónica

$649.900

Eliminar

{desafío} latam_

Emitir eventos a componente padre

Ejemplo del $emit

Así como los eventos ya conocidos como el **click**, **dbclick**, **scroll**, **change**, entre otros, podemos crear nuestros propios eventos.

Agreguemos entonces el evento **`@eliminarInstrumento`** que utilice el índice de la iteración para eliminar el elemento del instrumento del arreglo.

```
<template>
 <div>
   <Header />
   <section class="instrumentos">
     <div v-for="(instrumento, i) in instrumentos">
       <Instrumento
         :foto="instrumento.foto"
         :nombre="instrumento.nombre"
         :descripcion="instrumento.descripcion"
         @eliminarInstrumento="instrumentos.splice(i, 1)"
       />
     </div>
   </section>
 </div>
</template>
```

Emitir eventos a componente padre

El índice en el v-for

El **v-for** trabaja muy similar al método **forEach**, donde podemos extraer el índice de cada iteración.

Solo necesitamos meter entre paréntesis el argumento para declarar un segundo parámetro equivalente al índice.

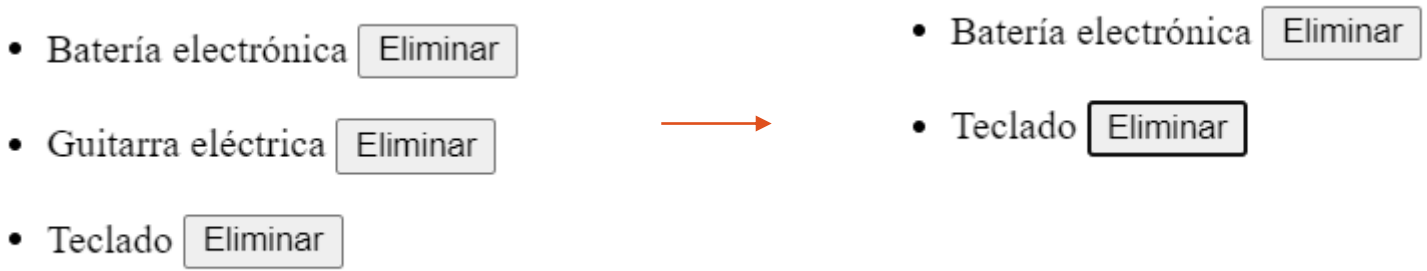
```
…
  <div v-for="(instrumento, i) in instrumentos">
…
```

Ejercicio propuesto 1

Emitir eventos a componente padre

$emit

- Crea una lista **TODO** donde cada **li** tenga un botón **eliminar** y use el **$emit** para indicarle al componente padre que dicha tarea se debe eliminar



/* Slots */

Distribución de contenido en Slots

slot

Es posible que necesitemos que parte de un componente sea completamente dinámico debido a que no tengamos la certeza de qué es lo que deba ir ahí.

Para estos casos donde el contenido es desconocido, podemos ocupar los `slots`.

Veamos un ejemplo básico:

Ejercicio guiado 2

Distribución de contenido en Slots

slot

Crea un componente **button** con el siguiente template y el siguiente style

```
<template>
 <button>
   <slot /> ←
 </button>
</template>

<style scoped>
button {
 background: darkgreen;
 box-shadow: 4px 4px 20px 4px black;
 color: white;
 padding: 5px 20px;
 text-shadow: 1px 1px 1px black;
 border-radius: 15px;
}
</style>
```

La “etiqueta” <slot /> se sustituirá por el contenido que le agreguemos desde el componente padre

{desafío} latam_

Distribución de contenido en Slots

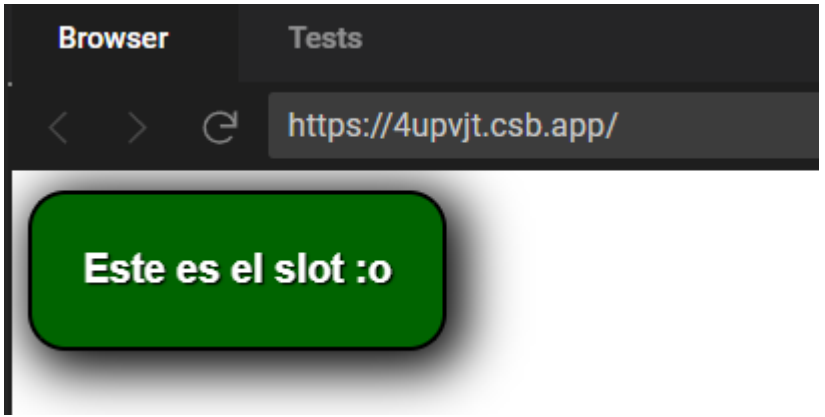
slot

Desde el componente App, agrega el componente **Button** y dentro escribe un texto.

```
<template>
 <div>

   <Button>
     <h3>Este es el slot :o</h3>
   </Button>

 </div>
</template>
```



Distribución de contenido en Slots

slot

Si usamos el inspector de elementos veremo que efectivamente el **h3** que hemos colocado como contenido del componente, tomó el lugar de la etiqueta `<slot />`

Ejercicio propuesto 2

Distribución de contenido en Slots

slot

- Crea un componente Card cuya imágen sea una prop pero el resto del contenido sea proveído por un slot

La imagen es un src proveído como una **prop**

Toda esta sección es proveído en un **<slot />**

¿Qué otros casos de uso imaginas que pudiera tener los slots?

Componentes dinámicos

En algunos casos la incertidumbre de una aplicación y su contenido puede escalar a no solo una parte de un componente sinó de qué componente debemos renderizar.

Para este escenario en Vue contamos con los componentes dinámicos que nos **permite definir con una variable del estado cuál componente se renderiza en un lugar en específico**.

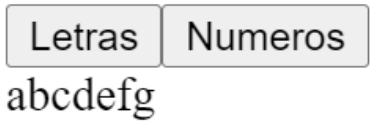
Veamos un ejemplo:

Ejercicio guiado 3

Componentes dinámicos

1. Crea un pequeño template que incluya 2 botones que al presionar cambie el valor de una variable del estado **componenteARenderizar** al nombre del componente a renderizar.

```
<template>
 <div>
   <button @click="componenteARenderizar = 'Letras'">Letras</button>
   <button @click="componenteARenderizar =
'Numeros'">Numeros</button>
   <div>
     <component :is="componenteARenderizar" />
   </div>
 </div>
</template>
```



Componentes dinámicos

2. Crea 2 componentes llamados **Letras** y **Numeros** usando el siguiente código

```
<template>
 <div>abcdefg</div>
</template>
```

Letras.vue

```
<template>
 <div>1234567</div>
</template>
```

Numeros.vue

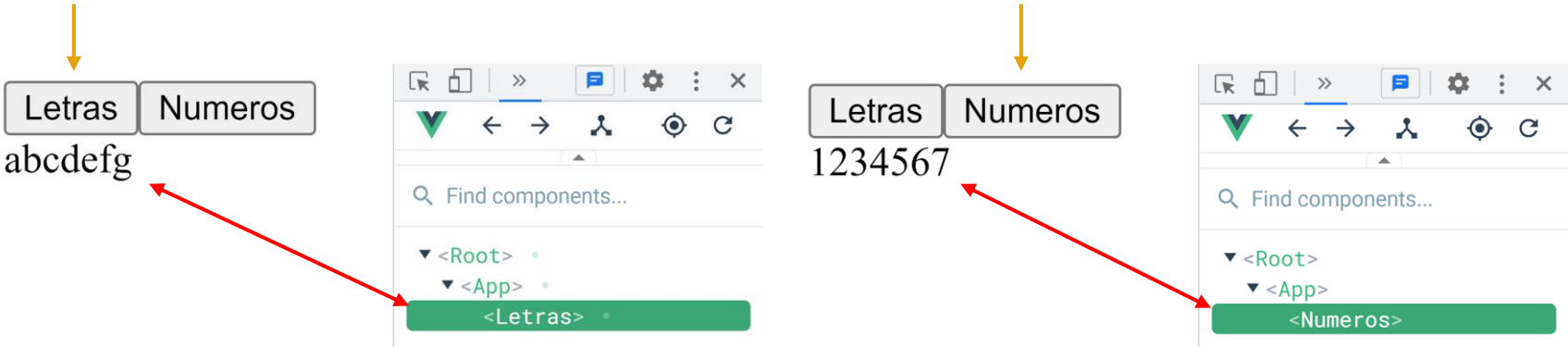
Componentes dinámicos

3. En el script del componente **App**, importa ambos componentes creados y crea la variable en el estado **componenteARenderizar**

```
<script>
import Letras from "./components/Letras.vue";
import Numeros from "./components/Numeros.vue";
export default {
 data() {
   return {
     componenteARenderizar: "Letras",
   };
 },
 components: {
   Letras,
   Numeros,
 },
};
</script>
```

Componentes dinámicos

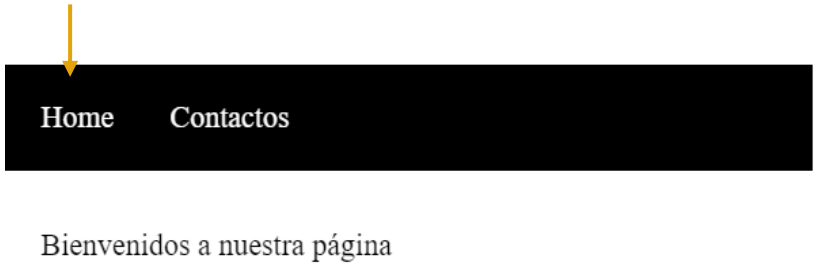
Con lo realizado ahora podemos presionar los botones y veremos como cambia el componente debajo.

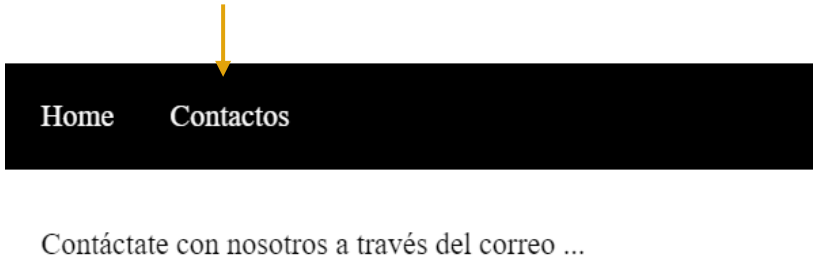


Ejercicio propuesto 3

Componentes dinámicos

- Crea un menú de navegación con 2 opciones que cambie el contenido principal de la aplicación.





¿Qué es lo que más te ha gustado de lo aprendido hasta el momento?

Próxima sesión...

- *Guía de ejercicios.*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales