

# ADM



Dudas?

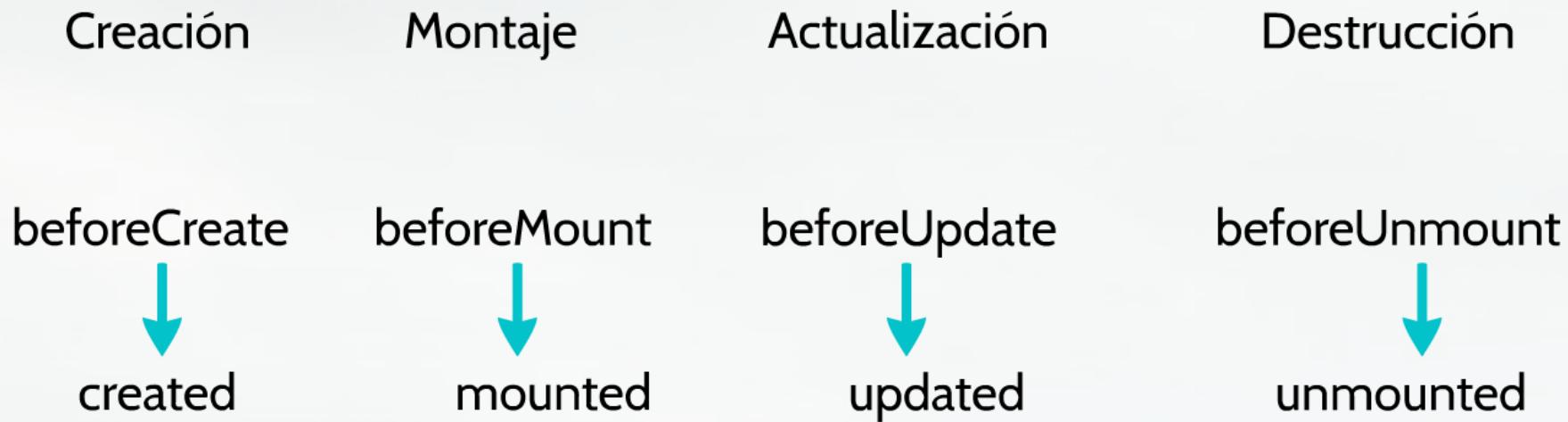
# ADM



Dudas?

## Ciclo de vida

- Cada instancia de Vue pasa a través de una serie de pasos de inicialización .
- Se ejecutan funciones llamadas ***hooks del ciclo de vida***, lo que nos permite agregar código propio en etapas específicas:



# VueJS

## beforeCreate:

Se ejecuta antes de que se cree la instancia de Vue y antes de que se inicialicen las opciones del componente.

En este gancho no podemos acceder a las propiedades declaradas en el data, su contenido si lo queremos referenciar será undefined.

## created:

Se ejecuta después de que se ha creado la instancia de Vue y se han inicializado las opciones del componente.

En este punto, la instancia ya está disponible y se pueden acceder a las opciones de datos, métodos, props y eventos.

# VueJS

## beforeMount:

Se ejecuta antes de que el componente se monte en el DOM.

## mounted:

Se ejecuta después de que el componente se ha montado en el DOM. En este punto, el componente ya está visible en la página y se puede acceder al DOM y a los elementos del mismo.

## beforeUpdate:

Se ejecuta antes de que el componente se actualice debido a cambios en los datos o props.

## updated:

Se ejecuta después de que el componente se ha actualizado debido a cambios en los datos o props. En este punto, el DOM ya ha sido actualizado para reflejar los cambios

# VueJS

## beforeUnmount:

Se ejecuta antes de que el componente se desmonte y destruya.

## unmounted:

Se ejecuta después de que el componente se ha desmontado y destruido. En este punto, el componente ya no está en el DOM y se liberan los recursos.

- Estos ganchos nos permiten realizar acciones específicas en cada etapa del ciclo de vida del componente,
- Sirven para inicializar datos, realizar llamadas a API, manipular el DOM, limpiar recursos, entre otros.



## Organizando componentes

- Cuando empezamos a dividir responsabilidades por medio de los componentes, tener todos en un mismo archivo puede resultar confuso.
- Para ordenar nuestro código podemos crear **por cada componente un archivo js**
- Facilita la **localización de código y funcionalidades**.
- Nuestro **index.html** y **formulario.html** tendrán primero la **inserción de vuejs** y luego **por cada componente un archivo js** y por último el archivo **app.js** con la instancia principal.

```
<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
<script src="scripts/componentes/componente-lista.js"></script>
<script src="scripts/componentes/componente-form.js"></script>
<script src="scripts/componentes/componente-navegacion.js"></script>
<script src="scripts/app.js"></script>
```

# VueJS

- index.html, tendrá solo el componente navegación

```
<componente-navegacion></componente-navegacion>
```

- formulario.html, tendrá la navegación y el formulario como componentes

```
<componente-navegacion></componente-navegacion>
<componente-form></componente-form>
```

- *Estos componentes se declaran como constantes dentro de cada archivo .js y tendrán un objeto de opciones.*

```
const ComponenteForm = {
    //data - template - methods
}
```

# VueJS

- El archivo **app.js** tendrá la instancia principal con sus propios datos, etc
- Luego, registramos los componentes **ComponenteForm** y **ComponenteLista** y **ComponenteNavegacion** (que deben estar definidos en archivos separados). Esto se hace con el *método app.component()*.



```
app.js

1 const app = Vue.createApp({
2   data() {
3     return {
4       mensaje: "Ejemplo-la instancia root"
5     }
6   },
7   app.component('componente-form', ComponenteForm)
8   app.component('componente-lista', ComponenteLista)
9   app.component('componente-navegacion', ComponenteNavegacion)
10
11 app.mount('.contenedor');
```

# VueJS

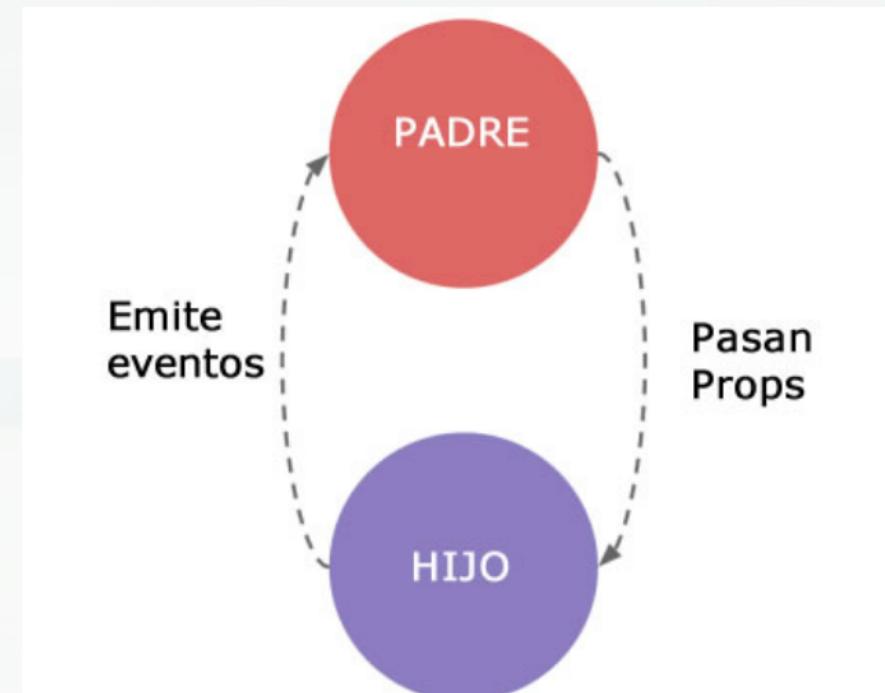
- Primer argumento de `app.component()` es una cadena de texto que representa el nombre con el cual podremos usar el componente en nuestro HTML.
- El segundo argumento es la definición del componente, que en este caso es la constante que creamos en el archivo js donde creamos el componente.
- Entonces, cuando hacemos `app.component('componente-form', ComponenteForm)`, estamos diciendo:

*"Vue, registrá un nuevo componente llamado 'componente-form' y usá esta definición de componente (ComponenteForm) que tengo acá".*

- De esta forma, Vue sabe que cuando encuentre una etiqueta `<componente-form>` en nuestro HTML, debe renderizar el componente que registramos con esa definición.

## Comunicacion por emit

- La creación de componentes facilita el código modular, con división de responsabilidades.
- Los componentes suelen comunicarse entre si, para el paso de información.



# VueJS

- Los **hijos tienen otra forma para enviar datos hacia un componente padre. Lo hacen emitiendo eventos.**
- Los componentes **hijos emiten eventos** utilizando el método:  
`this.$emit('nombreEvento', datos).`
- Esto significa que un **componente hijo puede notificar a su componente padre sobre un evento específico junto con cualquier dato relevante.**

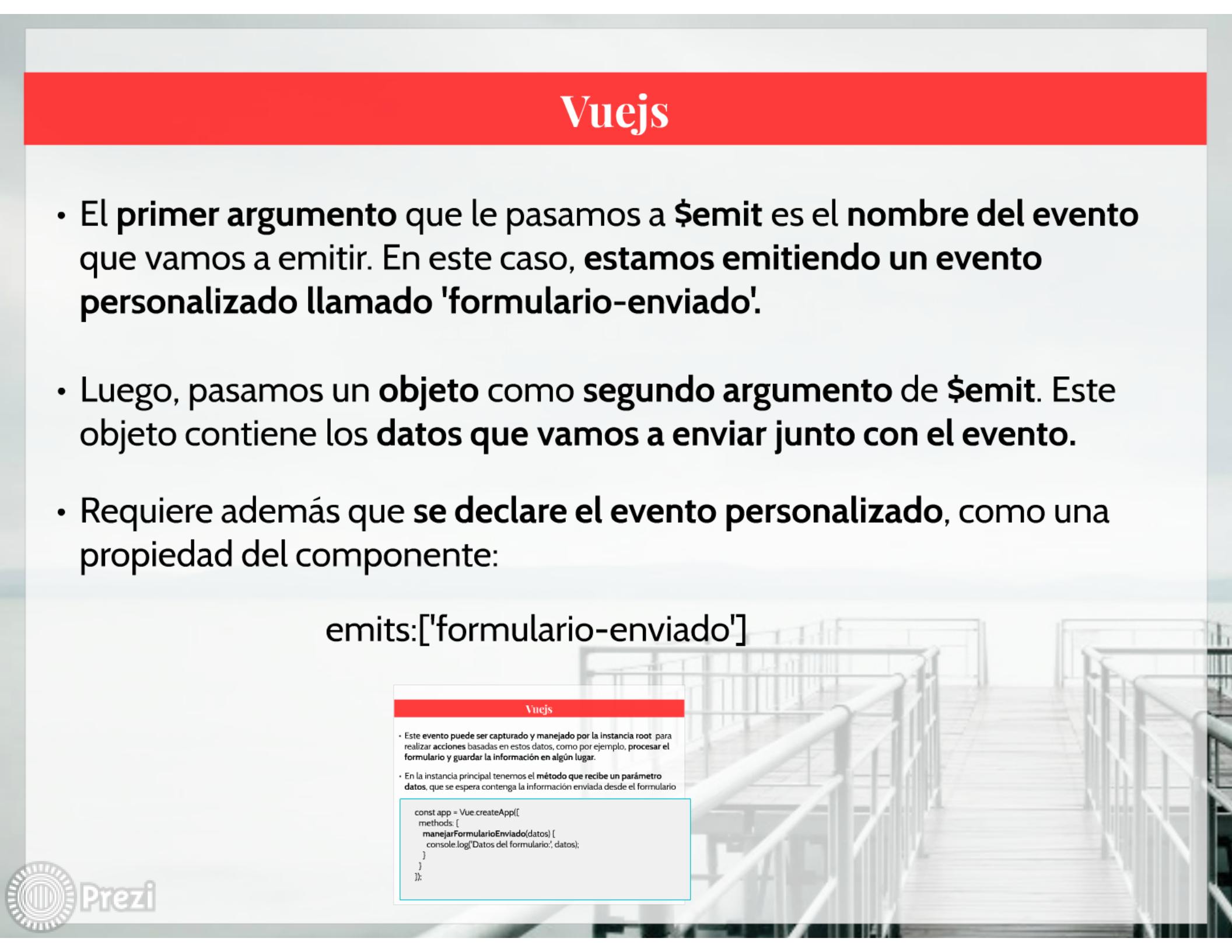
Por ejemplo podríamos **emitir los datos de un form hijo hacia un padre:**

```
this.$emit('formulario-enviado', { nombre: this.nombre, apellido: this.apellido });
```

# Vuejs

- El primer argumento que le pasamos a \$emit es el **nombre del evento** que vamos a emitir. En este caso, **estamos emitiendo un evento personalizado llamado 'formulario-enviado'**.
- Luego, pasamos un **objeto** como segundo argumento de \$emit. Este objeto contiene los **datos que vamos a enviar junto con el evento**.
- Requiere además que **se declare el evento personalizado**, como una propiedad del componente:

emits:['formulario-enviado']



Vuejs

Este evento puede ser capturado y manejado por la instancia root para realizar acciones basadas en estos datos, como por ejemplo, procesar el formulario y guardar la información en algún lugar.

En la instancia principal tenemos el **método que recibe un parámetro datos**, que se espera contenga la información enviada desde el formulario

```
const app = Vue.createApp([
  methods: [
    manejarFormularioEnviado(datos) {
      console.log('Datos del formulario:', datos);
    }
  ]
]);
```

# Vuejs

- Este **evento puede ser capturado y manejado por la instancia root** para realizar acciones basadas en estos datos, como por ejemplo, **procesar el formulario y guardar la información en algún lugar**.
- En la instancia principal tenemos el **método que recibe un parámetro datos**, que se espera contenga la información enviada desde el formulario

```
const app = Vue.createApp({  
  methods: {  
    manejarFormularioEnviado(datos) {  
      console.log('Datos del formulario:', datos);  
    }  
  }  
});
```

# VueJS

- En nuestro HTML, insertamos el **componente componente-formulario** y **escuchamos el evento formulario-enviado**:

```
<div id="app">
  <componente-formulario @formulario-enviado="manejarResultado">
    </componente-formulario>
</div>
```

El **@formulario-enviado** es el evento que el componente **componente-formulario** emitirá cuando se envíe el formulario.

- Cuando **este evento se emita**, se ejecutará la función `manejarResultado` en la instancia de Vue, permitiendo la comunicación entre la instancia root y su componente hijo.

# VueJS

- Desde el panel de Vue.js Devtools podemos ver la emisión del evento del componente, con sus respectivos datos enviados:

A screenshot of a web browser showing a simple form. The form has two input fields: 'Nombre:' and 'Apellido:', both currently empty. Below the inputs is a blue rectangular button labeled 'Enviar'.

A screenshot of the Vue.js Devtools interface. The top navigation bar shows tabs: Elementos, Consola, Fuentes, Red, Rendimiento, Memoria, Aplicación, Seguridad, Lighthouse, **Vue**, and more. On the left is a sidebar with icons for 4 layers, Keyboard, Component events (which is selected), and Performance. The main area shows a list of events. One event is highlighted in green: "formulario-enviado by componente-formulario 22:21:44". To the right of this event, there is a detailed view with sections for "event info" and "params". The "event info" section shows "component: componente-formulario" and "event: 'formulario-enviado'". The "params" section shows an array with one object: "0: Object" containing "apellido: 'Perez'" and "nombre: 'Pepe'".

```
formulario-enviado by componente-formulario 22:21:44
  event info
    component: componente-formulario
    event: "formulario-enviado"
  params: Array[1]
    0: Object
      apellido: "Perez"
      nombre: "Pepe"
```



# Dudas?

# ADM



Dudas?