



# Binding de formularios

Interpolaciones y directivas

***Implementar un formulario de datos interactivo utilizando form binding de Vue para dar solución a un requerimiento.***

- 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



## ¿Qué aprenderás en esta sesión?

- *Reconoce interpolaciones, directivas, tipos y directivas para el despliegue de variables en un componente Vue*
- *Reconoce la característica de binding de Vue para la implementación de formularios interactivos.*

¿Cuánta diferencia en tiempo  
consideras que podemos  
ahorrar ahora usando las  
interpolaciones?



# Obtención del value en inputs

*input.value*

En la unidad anterior aprendimos a **interpol**ar variables del estado en el template usando las doble llaves (`{{ }}`).

Si comparamos esta tarea con **JavaScript puro** debemos recurrir a la manipulación del DOM

Por ejemplo si tenemos este **párrafo**:

```
<p> </p>
```

Y queremos asignarle un contenido con una variable interpolada, podemos hacer lo siguiente:

```
const nombre = "Alan Turing";  
document.querySelector("p").innerHTML = `Hola mi nombre es ${nombre}`;
```

# Obtención del value en inputs

*input.value*

Y si el valor que deseamos interpolar existiese en un input de algún formulario, debemos primero **tomar el valor escrito por el usuario** y posteriormente **asignarlo** al elemento html en una interpolación.

```
const nombre = document.querySelector("input#nombre").value; ←  
  
document.querySelector("p").innerHTML = `Hola mi nombre es ${nombre}`;
```

En Vue Js esta tarea se puede hacer mucho más resumidamente usando las directivas que veremos a continuación.



¿Cómo actualizarías  
en tiempo real un contenido  
en el HTML según lo que  
el usuario va escribiendo  
en un input?



**/\* Directivas \*/**



# Directivas

En Vue Js podemos crear una conexión directa entre un **input** y una **variable del estado** a través de las directivas.

Las directivas son **atributos especiales** que el framework reconoce y que contiene una funcionalidad integrada.



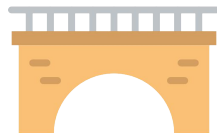
# Directiva

## *v-model*

La primera directiva que veremos es el **v-model**.

Con esta directiva vamos a conseguir un **puente directo** entre:

- el valor de un elemento de formulario.
- una variable del estado.



Ejercicio guiado

"Demostración del  
v-model"



# Demostración del v-model

Escribamos en el template del componente App lo siguiente:

```
<template>
  <div id="app">
    <h1>
      Bienvenido al curso de <span>{{ nombreDelCurso }}</span>
    </h1>

    <input v-model="nombreDelCurso" />

    <input id="nombre">
  </div>
</template>
```



# Demostración del v-model

Ahora en el script creamos una estado que contenga una variable **nombreDelCurso**

```
<script>
export default {
  name: "App",
  data() {
    return {
      nombreDelCurso: "",
    };
  },
};
</script>
```



# Demostración del v-model

Y finalmente un pequeño estilo que destaque de color verde las etiquetas span

```
span {  
  color: #3fb27f;  
}
```



# Demostración del v-model

Con lo anterior podemos ver como al escribir en el input, reactivamente va apareciendo el valor como contenido del span.

## Bienvenido al curso de Vue Js



## Ejercicio guiado

"Completa la frase  
con v-model"





# Completa la frase con v-model

Usa el v-model para completar la frase:

“Para más información contactarse al número {{ numeroDeTelefono }}”

Para más información contactarse al número +56 9 1234 5678



# Directiva

## *Data binding*

Las directivas nos obsequian un puente directo de comunicación entre el valor de un atributo html y el estado del componente.



# Directivas

## El patrón de diseño MVVM

```
<template>
  <div id="app">
    <p>Para más información contactarse al número {{ numeroDeTelefono }}</p>

    <input v-model="numeroDeTelefono" />
  </div>
</template>

<script>
export default {
  name: "App",
  data() {
    return {
      numeroDeTelefono: "",
    };
  },
};
</script>
```

En Vue Js le denominamos **vista** al template mientras que al objeto que se exporta en el script es el **modelo**.

Este enfoque en la programación se conoce como el patrón de diseño

### MVVM

(Modelo - Vista , Vista - Modelo )

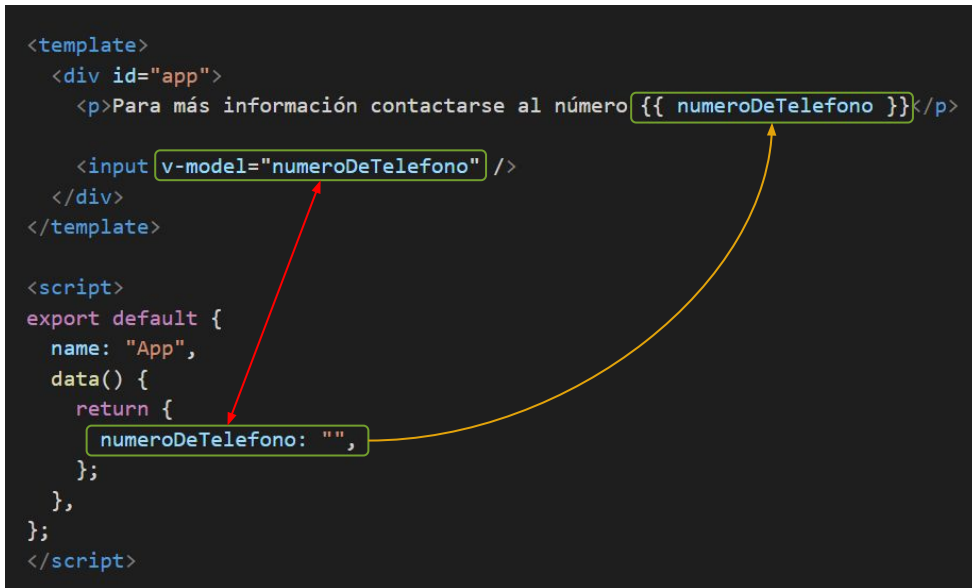
# Directivas

## One-Way y Two-Way data binding

Existen 2 tipos de enlaces entre el modelo y la vista: el **one-way** y el **two-way**.

En el caso del two-way si el valor del input cambia, el valor de la variable también lo hará (bidireccional)

Mientras que el one-way solo pasa el valor del modelo a la vista (una dirección)



The diagram illustrates two-way data binding in Vue.js. It consists of two parts: a template and a script. In the template, there is a paragraph with a placeholder for a phone number and an input field. Both are bound to a variable named 'numeroDeTelefono'. In the script, a default export object contains a 'data' function that returns an object with 'numeroDeTelefono' set to an empty string. A red arrow points from the input field's 'v-model' attribute to the 'numeroDeTelefono' property in the data object, representing the 'one-way' binding from view to model. A yellow curved arrow points from the 'numeroDeTelefono' property in the data object back to the placeholder in the paragraph, representing the 'one-way' binding from model to view. Together, they form a two-way binding.

```
<template>
  <div id="app">
    <p>Para más información contactarse al número {{ numeroDeTelefono }}</p>

    <input v-model="numeroDeTelefono" />
  </div>
</template>

<script>
export default {
  name: "App",
  data() {
    return {
      numeroDeTelefono: "",
    };
  },
};
</script>
```

# ¿Qué tipo de binding utilizamos en el desafío de la unidad anterior?



# **`/* Tipos de directivas y sus usos */`**

# Tipos de directivas

Además del **v-model** existen otras directivas que se identifican por prefijo **v-** como:

- **v-for**: Itera un arreglo u objeto directamente en el template para renderizar elementos dinámicamente
- **v-show**: Muestra un elemento según una expresión o valor booleano

Y muchas más que iremos viendo más adelante en el módulo.

# Binding de atributos conocidos





# Binding en atributos

Hasta ahora hemos utilizado el input como ejemplo para usar el v-model y obtener reactivamente su valor en el estado.

También podemos usar una variable del estado y asignar dinámicamente su valor en atributos html.

Bastará con agregar el prefijo **v-bind:** en el atributo para que utilice una variable del estado como valor.

```

```

O también podemos optar por el **modo abreviado** escribiendo solamente los 2 puntos (:)

```

```



# Binding en atributos

Veamos un ejemplo con el atributo **src** de una etiqueta imagen.

```

```

Cuyo valor es una variable del estado:

```
data() {  
  return {  
    perritoSrc: "https://i.imgur.com/Y0QQmIK.jpeg",  
  };  
}
```



# Binding en atributos

Con lo anterior podremos ver en el navegador la imagen correspondiente a la URL de la variable **perritoSrc**.

Lo que logró este enlace fueron los dos puntos (:) antes del atributo **src** de la imagen.

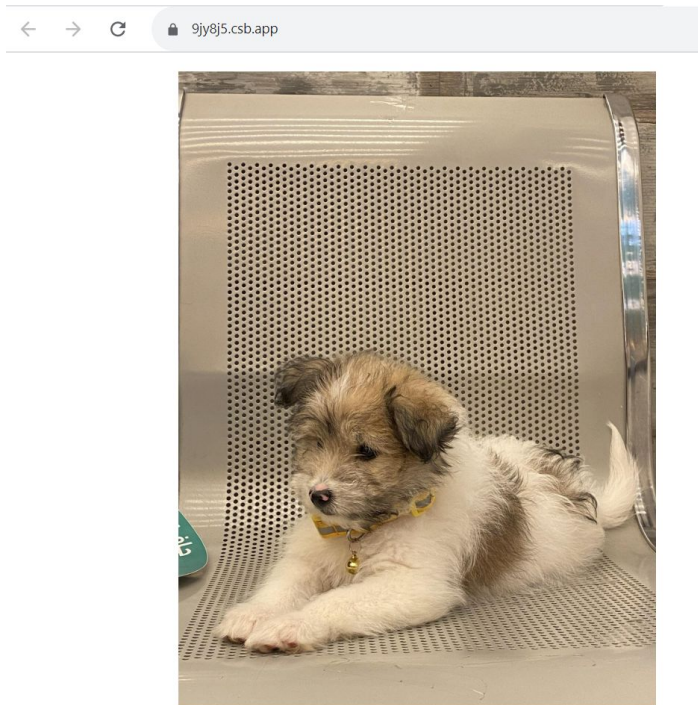
```

```

El valor entonces debe corresponder a una **variable del estado**.

*Esta es una abreviación de la directiva **v-on**:*

**{desafío}**  
latam\_



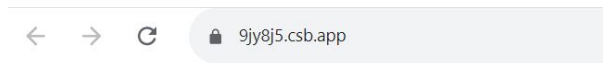
# Tu ciudad favorita

:src="<variable del estado>"



# Tu ciudad favorita

Busca una imagen de tu ciudad favorita y agrega el src dinámicamente a través de un binding al src de una etiqueta imagen.



**/\* El patrón de diseño MVVM \*/**

# Modelo Vista - Vista Modelo

MVVM es un patrón arquitectónico que separa la lógica de la aplicación (Modelo) de la presentación visual (Vista) y la interacción del usuario (Modelo de Vista).

Beneficios de este patrón:

- Divide la aplicación en capas, el código se vuelve más fácil de entender, modificar y mantener.
- Cada capa puede probarse de forma independiente
- Reutilización de componentes
- Al estar la lógica encapsulada en el Modelo, los cambios en la interfaz de usuario no afectan la lógica empresarial y viceversa.

# Modelo Vista - Vista Modelo

## Modelo

- Representa los datos de la aplicación, como objetos, estructuras o bases de datos.
- No contiene lógica de presentación ni interacción del usuario.
- Comunica los cambios a través de eventos o notificaciones.

## Vista

- Define la interfaz de usuario visual de la aplicación, utilizando elementos HTML, CSS y componentes Vue.js.
- No contiene lógica de negocios ni estado de la aplicación.
- Se conecta al Modelo de Vista para mostrar datos y responder a eventos.



**Gracias a las directivas  
y el enlace de variables  
en el template lograremos  
tener el HTML mucho  
más limpio y legible.**





## Próxima sesión...

- *Utiliza binding básico para la implementación de formularios interactivos.*
- *Utiliza bindings con valores para la implementación de formularios interactivos.*

**{desafío}**  
**latam\_**

*Academia de  
talentos digitales*

