

¿Qué es Vue?

Vue (pronunciado /vju:/) es un framework de JavaScript para construir interfaces de usuario. Se basa en HTML, CSS y JavaScript estándar y proporciona un modelo de programación declarativo y basado en componentes que lo ayuda a desarrollar interfaces de usuario de manera eficiente, ya sean simples o complejas.

Aquí hay un ejemplo mínimo:

```
import { createApp } from 'vue'

createApp({
  data() {
    return {
      count: 0
    }
  }
}).mount('#app')

<div id="app">
  <button @click="count++">
    Count is: {{ count }}
  </button>
</div>
```

El ejemplo anterior demuestra las dos características principales de Vue:

- **Representación declarativa** : Vue amplía HTML estándar con una sintaxis de plantilla que nos permite describir declarativamente la salida HTML en función del estado de JavaScript.
- **Reactividad** : Vue rastrea automáticamente los cambios de estado de JavaScript y actualiza de manera eficiente el DOM cuando ocurren cambios.

El Framework Progresista.

Vue es un framework y un ecosistema que cubre la mayoría de las características comunes necesarias en el desarrollo frontend. Pero la web es extremadamente diversa: las cosas que construimos en la web pueden variar drásticamente en forma y escala. Con eso en mente, Vue está diseñado para ser flexible y adoptarse de forma incremental. Dependiendo de nuestro caso de uso, Vue se puede usar de diferentes maneras:

- Mejora de HTML estático sin un paso de compilación
- Incrustación como componentes web en cualquier página
- Solicitud de una sola página (SPA)
- Fullstack / Representación del lado del servidor (SSR)
- Jamstack / Generación de sitios estáticos (SSG)
- Orientación a escritorio, móvil, WebGL e incluso al terminal

A pesar de la flexibilidad, el conocimiento central sobre cómo funciona Vue se comparte en todos estos casos de uso. Incluso si ahora es solo un principiante, el conocimiento adquirido en el camino seguirá siendo útil a medida que crezca para abordar metas más ambiciosas en el futuro. Si es un veterano, puede elegir la forma óptima de aprovechar Vue en función de los problemas que está tratando de resolver, manteniendo la misma productividad. Es por eso por lo que llamamos a Vue "The Progressive Framework": es un marco que puede crecer contigo y adaptarse a tus necesidades.

Componentes de un solo archivo.

En la mayoría de los proyectos de Vue habilitados para herramientas de compilación, creamos componentes de Vue utilizando un formato de archivo similar a HTML llamado **componente de archivo único** (también conocido como archivos *.vue, abreviado como **SFC**). Un SFC de Vue, como sugiere el nombre, encapsula la lógica del componente (JavaScript), la plantilla (HTML) y los estilos (CSS) en un solo archivo. Aquí está el ejemplo anterior, escrito en formato SFC:

```
<script>
export default {
  data() {
    return {
      count: 0
    }
  }
}
</script>

<template>
  <button @click="count++">Count is: {{ count }}</button>
</template>

<style scoped>
button {
  font-weight: bold;
}
</style>
```

Estilos de API.

Los componentes de Vue se pueden crear en dos estilos de API diferentes: API de **opciones** y API de **composición**.

API de opciones.

Con la API de opciones, definimos la lógica de un componente usando un objeto de opciones como **data**, **methods** y **mounted**. Las propiedades definidas por opciones se exponen en funciones **this** internas, que apuntan a la instancia del componente:

```
<script>
export default {
  // Propiedades devueltas por data() . Variables reactivas
  // las podemos recuperar con `this`.
  data() {
    return {
      count: 0
    }
  },

  // Los métodos los podemos definir como funciones
  methods: {
    increment() {
      this.count++
    }
  },

  // Esta función se llama cuando se monta el componente.
  mounted() {
    console.log(`The initial count is ${this.count}.`)
  }
}
</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

API de composición.

Con la API de composición, definimos la lógica de un componente utilizando funciones API importadas. En SFC, la API de composición se usa normalmente con **<script setup>**. El atributo **setup** hace que Vue realice transformaciones en tiempo de compilación que nos permiten usar la API de composición con menos repeticiones. Por ejemplo, las importaciones y las variables/funciones de nivel superior declaradas en **<script setup>** se pueden usar directamente en la plantilla.

Aquí està el mismo componente, con exactamente la misma plantilla, pero usando la API de composición y `<script setup>` en su lugar:

```
<script setup>
import { ref, onMounted } from 'vue'

// reactive state
const count = ref(0)

// functions that mutate state and trigger updates
function increment() {
  count.value++
}

// lifecycle hooks
onMounted(() => {
  console.log(`The initial count is ${count.value}.`)
})
</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

¿Cuál elegir?.

Ambos estilos de API son totalmente capaces de cubrir casos de uso comunes. Son interfaces diferentes impulsadas por el mismo sistema subyacente. De hecho, ¡la API de opciones se implementa sobre la API de composición! Los conceptos y conocimientos fundamentales sobre Vue se comparten entre los dos estilos.

La API de opciones se centra en el concepto de una "instancia de componente" (**this** como se ve en el ejemplo), que generalmente se alinea mejor con un modelo mental basado en clases para usuarios que provienen de entornos lingüísticos de programación orientada a objetos. También es más amigable para principiantes al abstraer los detalles de reactividad y hacer cumplir la organización del código a través de grupos de opciones.

La API de composición se centra en declarar variables de estado reactivas directamente en el ámbito de una función y componer el estado de varias funciones juntas para manejar la complejidad. Tiene una forma más libre y requiere una comprensión de cómo funciona la reactividad en Vue para ser utilizado de manera efectiva. A cambio, su flexibilidad permite patrones más poderosos para organizar y reutilizar la lógica.

Si es nuevo en Vue, esta es nuestra recomendación general:

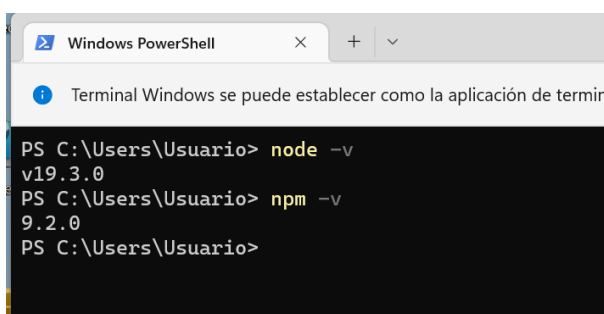
- Para propósitos de aprendizaje, elige el estilo que te parezca más fácil de entender. Una vez más, la mayoría de los conceptos básicos se comparten entre los dos estilos. Siempre puedes elegir el otro estilo más tarde.
- Para uso en producción:
 - Opte por la API de opciones si no está utilizando herramientas de compilación o planea usar Vue principalmente en escenarios de baja complejidad, por ejemplo, mejora progresiva.
 - Elija Composición API + Componentes de un solo archivo si planea crear aplicaciones completas con Vue.

Herramientas a Instalar.

Para realizar este curso vamos a utilizar las siguientes herramientas/software:

- Chrome
- Vue.js devtools. <https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjimejigglicpcpnannhbledajbpd?hl=es>
- Node.js. <https://nodejs.org/es/download/current/>.
Sirve para crear sitios web dinámicos muy eficientes, escritos con el lenguaje de programación JavaScript. Normalmente, los desarrolladores se decantan por este entorno de ejecución cuando buscan que los procesos se ejecuten de forma ágil y sin ningún tipo de bloqueo cuando las conexiones se multiplican.
- Visual Studio Code.
 - Volar.
 - ES7+React

Una vez instalado todo vamos a comprobar que esta funcionando. Para ello vamos a abrir el terminal de Windows y vamos a probarlo.



```
Windows PowerShell
Terminal Windows se puede establecer como la aplicación de termin...

PS C:\Users\Usuario> node -v
v19.3.0
PS C:\Users\Usuario> npm -v
9.2.0
PS C:\Users\Usuario>
```

Creación de nuestra 1ª aplicación Vue.

En esta sección, presentaremos cómo montar una aplicación de una sola página de Vue en nuestra máquina local. El proyecto creado utilizará una configuración de compilación basada en [Vite](#) y nos permitirá usar Vue [Single-File Components](#) (SFC).

Ejecutamos el siguiente comando en el terminal de Windows dentro de nuestra carpeta del 1er proyecto.

> npm init vue@latest

Este comando instalará y ejecutará [create-vue](#), la herramienta oficial de andamiaje del proyecto Vue. Se nos presentarán indicaciones para varias funciones opcionales, como TypeScript y compatibilidad con pruebas:

- ✓ Project name: ... <your-project-name>
- ✓ Add TypeScript? ... No / Yes
- ✓ Add JSX Support? ... No / Yes
- ✓ Add Vue Router for Single Page Application development? ... No / Yes
- ✓ Add Pinia for state management? ... No / Yes
- ✓ Add Vitest for Unit testing? ... No / Yes
- ✓ Add Cypress for both Unit and End-to-End testing? ... No / Yes
- ✓ Add ESLint for code quality? ... No / Yes
- ✓ Add Prettier for code formatting? ... No / Yes

Scaffolding project in ./<your-project-name>...
Done.

De momento vamos a seleccionar 'No' en todas las opciones. Una vez creado el proyecto, seguimos las instrucciones para instalar las dependencias e iniciar el servidor de desarrollo:

> cd <nombre del proyecto>
> npm install
> npm run dev

¡Ahora deberíamos tener nuestro primer proyecto Vue en ejecución!
Tengamos en cuenta que los componentes de ejemplo en el proyecto generado se escriben utilizando la [API de composición](#) y **<script setup>**.

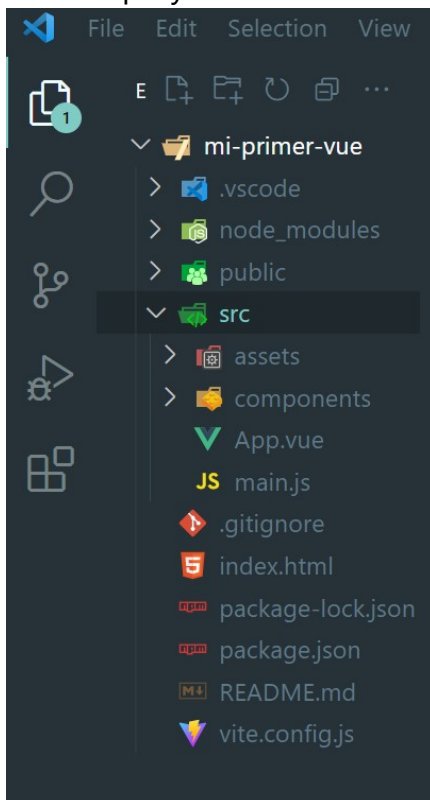
```
C:\WINDOWS\system32\cmd. X + v

VITE v4.0.3 ready in 363 ms

→ Local:   http://localhost:5173/
→ Network: use --host to expose
→ press h to show help
```

Vamos a abrir el navegador y copiar la anterior dirección para comprobar que funciona.

Ahora vamos a abrir nuestra carpeta y veremos todos ficheros que componen nuestro proyecto. Vamos a arrastrar esta carpeta a VS Code.



Vamos a abrir el fichero App.vue y lo vamos a borrar todo y añadiremos nuestro primer código.

```
<template>
  <h1>Hola Mundo.</h1>
</template>
```

Si miramos nuestro navegador veremos que se ha actualizado y que ahora pondrá 'Hola Mundo'.

Ahora vamos a modificar el estilo de la etiqueta.

```
<template>
```



```
<h1>Hola Mundo.</h1>
</template>

<style>
h1 {
  color: blue;
}
</style>
```

Los componentes de un solo archivo de Vue (**SFC**) son un formato de archivo especial que nos permite encapsular la plantilla (**template**), la lógica (**script**) y el estilo (**style**) de un componente de Vue en un solo archivo. Aquí hay un ejemplo de SFC:

```
<script setup> //LOGICA--JAVASCRIPT
</script>

<template>      //PLANTILLA en HTML
  <h1>Hola Mundo.</h1>
</template>

<style>         //ESTILO
h1 {
  color: blue;
}
</style>
```

El único componente que no puede faltar es el **template (plantilla)**.

Sintaxis de plantilla.

- Vue utiliza una sintaxis de plantilla basada en HTML que le permite vincular declarativamente el DOM representado a los datos de la instancia del componente subyacente.
- Todas las plantillas de Vue son HTML sintácticamente válidas que pueden ser analizadas por navegadores compatibles con las especificaciones y analizadores de HTML.
- Debajo de la capa, Vue compila las plantillas en un código JavaScript altamente optimizado.
- Combinado con el sistema de reactividad, Vue puede calcular de manera inteligente la cantidad mínima de componentes para volver a renderizar

y aplicar la cantidad mínima de manipulaciones DOM cuando cambia el estado de la aplicación.

Interpolación de texto.

La forma más básica de enlace de datos es la interpolación de texto utilizando la sintaxis "Bigotes" (llaves dobles), es decir, es una forma de insertar valores en una plantilla:

```
<script setup>
  const valor= "Mundo"
</script>

<template>
  <h1>Hola {{ valor }}.</h1>
</template>
```

La etiqueta de bigote se reemplazará con el valor de la propiedad **valor** de la instancia del componente correspondiente. También se actualizará cada vez que cambie el **valor** la propiedad.

Enlaces de atributo.

Los bigotes no se pueden usar dentro de los atributos HTML. En su lugar, utilizamos una directiva **v-bind**:

```
<script setup>
  const name = "Mundo";
  const estiloColor = "color: red";
</script>

<template>
  <h1>Hola {{ name }}.</h1>
  <h2 v-bind:style="estiloColor">Esto es ROJO.</h2>
</template>
```

Debido a que **v-bind** se usa con tanta frecuencia, tiene una sintaxis abreviada dedicada:

```
<template>
  <h1>Hola {{ name }}.</h1>
  <h2 :style="estiloColor">Esto es ROJO.</h2>
</template>
```

Uso de expresiones de JavaScript.

Hasta ahora, solo hemos estado vinculando claves de propiedad simples en nuestras plantillas. Pero Vue en realidad admite todo el poder de las expresiones de JavaScript dentro de todos los enlaces de datos.

Estas expresiones se evaluarán como JavaScript en el ámbito de datos de la instancia del componente actual.

En las plantillas de Vue, las expresiones de JavaScript se pueden usar en las siguientes posiciones:

- Interpolaciones de texto interior (bigotes)
- En el valor del atributo de cualquier directiva de Vue (atributos especiales '*directivas*' que comienzan con **v-**)

Ejemplos:

```
{{ number + 1 }}
```

```
{{ ok ? 'YES' : 'NO' }}
```

```
{{ message.split('').reverse().join('') }}
```

En nuestro proyecto:

```
<script setup>
  const name = "Mundo";
  const estiloColor = "color: red";
  const arrayColores= ["red","green","gray","black","white"];

  const valor=true;
</script>

<template>
  <h1>Hola {{ name.toUpperCase() }}.</h1>
  <h2 :style="estiloColor">Esto es ROJO.</h2>
  <h2>{{ arrayColores }}</h2>
  <h2 :style="`color: ${arrayColores[1]}`">VERDE</h2>
  <h2>
```

```
{{ valor ? 'SI' : 'NO' }}  
</h2>  
</template>
```

Directivas.

Las directivas son atributos especiales con el prefijo **v-**. Vue proporciona una serie de directivas integradas.

El trabajo de una directiva es aplicar actualizaciones de forma reactiva al DOM cuando cambia el valor de su expresión.

- ***v-if***

Representa condicionalmente un elemento o un fragmento de plantilla en función de la veracidad del valor de la expresión.

- **Detalles**

Cuando **v-if** se alterna con un elemento, el elemento y sus directivas/componentes contenidos se destruyen y reconstruyen. Si la condición inicial es falsa, el contenido interno no se representará en absoluto.

- ***v-else***

- **Detalles:** El elemento hermano anterior debe tener **v-if** o **v-else-if**. Lo que significa que entre en **v-if** y un **v-else** no puede haber ninguna etiqueta que no contenga un condicional.

- ***v-else-if***

- **Detalles:** El elemento hermano anterior debe tener **v-if** o **v-else-if**.

Ejemplo:

```
<h2 v-if="valor==true"> Valor es TRUE</h2>  
<h2 v-else> Valor es !TRUE</h2>
```

Esto sería incorrecto:

```
<h2 v-if="valor==true"> Valor es TRUE</h2>
<h2> INCORRECTO </h2>
<h2 v-else> Valor es !TRUE</h2>
```

Si nos fijamos en la consola de Chrome veremos que el elemento que no cumple la condición no se crea al renderizar.

```
... <body> == $0
  <div id="app" data-v-app>
    <h1>Hola MUNDO.</h1>
    <h2 style="color: red;">Esto es ROJO.</h2>
    <h2>[ "red", "green", "gray" ]</h2>
    <h2 style="color: green;">VERDE</h2>
    <h2>SI</h2>
    <h2> Valor es TRUE</h2>
  </div>
  <script type="module" src="/src/main.js?t=1672424124430"></script>
</body>
```

- **v-show**

Alterna la visibilidad del elemento en función de la veracidad del valor de la expresión.

- **Detalles: v-show** funciona configurando la propiedad **display** CSS a través de estilos en línea.

Si observamos en la consola veremos que el elemento esta creado pero oculto.

```
<h2 v-show="valor==true">Valor es TRUE v-show</h2>
```

```
<h1>Hola MUNDO.</h1>
<h2 style="color: red;">Esto es ROJO.</h2>
<h2>[ "red", "green", "gray" ]</h2>
<h2 style="color: green;">VERDE</h2>
<h2>SI</h2>
<h2> Valor es !TRUE</h2>
<h2 style="display: none;">Valor es TRUE v-show</h2>
```

- **v-for**

Renderiza el elemento o el bloque de plantilla varias veces en función de los datos de origen. La estructura de un **v-for** es muy sencilla y se basa en la posibilidad de crear un bucle **for** desde los **templates** de código HTML de Vue. Está basado en el bucle **'forEach'** de Javascript.



- **Espera:** Array | Object | number | string | Iterable
- **Detalles:** El valor de la directiva debe usar la sintaxis especial: **alias in expresion** para proporcionar un alias para el elemento actual que se está iterando:

```
<ul>
  <li v-for="color in arrayColores">
    {{ color }}
  </li>
</ul>
```

También podemos especificar un alias para el índice (index), que, aunque no sea necesario ponerlo siempre lo devuelve.

Ejemplo:

```
<ul>
  <li v-for="(color,index) in arrayColores">
    {{index}} -- {{ color }}
  </li>
</ul>
```

Aunque el código anterior es teóricamente correcto, se nos pide identificar de forma única a cada elemento de un array y que lo asociemos con una clave que sea única. Por lo tanto, lo correcto sería aplicar a todo elemento que lleve un **v-for**, un atributo, **'key'** al que se le asigne un valor único del ítem.

```
<ul>
  <li v-for="(color,posi) in arrayColores" :key="posi">
    {{posi}}--{{ color }}
  </li>
</ul>
```

Ejercicio1: Vamos a declarar el siguiente array de objetos. Vamos a utilizar el v-for para realizar una lista desordenada que nos muestre todos los objetos del array. Como clave utilizaremos el campo nombre.

```
const arrayFrutas=[
  {
    nombre: "pera",
    precio: "1,5€",
    descripcion: "Esto es una pera.",
    stock: 10,
  },
  {
    nombre: "manzana",
    precio: "2,5€",
    descripcion: "Esto es una manzana.",
    stock: 15,
  },
  {
    nombre: "kiwi",
    precio: "3,5€",
    descripcion: "Esto es una kiwi.",
    stock: 8,
  },
];
```

Resultado:

- pera -- 1,5€ -- Esto es una pera. -- 10
- manzana -- 2,5€ -- Esto es una manzana. -- 15
- kiwi -- 3,5€ -- Esto es una kiwi. -- 8

• **v-for** con **v-if**

Cuando coexisten en el mismo nodo, **v-if** tiene una prioridad más alta que **v-for**. Eso significa que la condición **v-if** no tendrá acceso a las variables del alcance de **v-for**. *Por lo que no se recomienda usar **v-if** y **v-for** en el mismo elemento.*

Imaginemos que queremos sacar el listado anterior, pero con los artículos que tienen un stock => que 10.

Podríamos pensar que el siguiente código sería el correcto:

```
<ul>
  <li
    v-for="fruta in arrayFrutas" :key="fruta.name"
    v-if="fruta.stock > 0"
  >
    {{ fruta }}
  </li>
</ul>
```

Pero al tener v-if una prioridad más alta no se evaluará.

Esto se puede solucionar moviéndose a una etiqueta **v-for** envolvente (que también es más explícita): `<template>`

```
<template>
  <template v-for="fruta in arrayFrutas">
    <li v-if="fruta.stock >= 10">
      {{ fruta.nombre }} -- {{ fruta.stock }}
    </li>
  </template>
</template>
```


Resultado:

- pera -- 10
- manzana -- 15

Eventos.

Podemos usar la directiva **v-on**, que normalmente acortamos con el símbolo **@**, para escuchar eventos DOM y ejecutar JavaScript cuando se activan. El uso sería **v-on:click="manejador"** o con el atajo, **@click="manejador"**.

Ejemplo: Probarlo con la consola del navegador.

```
<script setup>
// método Boton
const Boton = () => {
  console.log("Pulsado Boton");
};
</script>

<template>
  <button v-on:click="Boton">Click Boton1</button>
  <button @click="Boton">Click Boton2</button>
</template>
```

También podemos pasar parámetros al método. Probarlo.

```
<script setup>
// método Boton
const Boton = (mensaje, valor) => {
  console.log(mensaje+valor);
};
</script>

<template>
  <button v-on:click="Boton('Boton1','1')">Click Boton1</button>
  <button @click="Boton('Boton2','2')">Click Boton2</button>
</template>
```

Los eventos son iguales o similares a los de JavaScript:

- ***Teclado:***

- @keydown
- @keypress
- @keyup
-

- ***Raton:***

- @click
- @dblclick
- @mouseenter
- @mouseleave
- @mousedown
- @mouseup
- @mouseout
-

Modificadores de Eventos.

Vue proporciona **modificadores de eventos** para **v-on**. Recordemos que los modificadores son sufijos de directiva indicados por un punto.

- **.prevent:** Desactiva la función por defecto del evento
- **.stop:** Detiene la propagación del evento

Modificadores del Teclado.

Vue proporciona alias para las teclas más utilizadas:

- .enter
- .tab
- .delete(captura las teclas "Eliminar" y "Retroceso")
- .esc
- .space
- .up
- .down
- .left
- .right

EJEMPLO:

```
<input @keyup.enter="submit" />
```

Teclas modificadoras del sistema

Podemos usar los siguientes modificadores para activar detectores de eventos del mouse o del teclado solo cuando se presiona la tecla modificadora correspondiente:

- .ctrl
- .alt
- .shift
- .meta

EJEMPLOS:

```
<!-- Alt + Enter -->  
<input @keyup.alt.enter="clear" />
```

```
<!-- Ctrl + Click -->  
<div @click.ctrl="doSomething">Do something</div>
```

Modificadores del botón del raton.

Estos modificadores restringen el controlador a eventos activados por un botón específico del mouse.

- .left
- .right
- .middle

NOTA: Como se puede observar en los ejemplos anteriores, se pueden poner varios modificadores en los eventos, teniendo en cuenta que el orden de estos es importante.

Variables reactivas.

Las variables reactivas se llaman así porque son capaces de refrescar la vista de los componentes del DOM cuando éstas cambian.

Ejemplo:

```
<script setup>
  let counter = 0;
  const increment = () => {
    counter++;
    // efectivamente aumentar
    console.log(counter);
  };
</script>
<template>
  <h2>{{ counter }}</h2>
  <button @click="increment">Incrementar</button>
</template>
```

</template>

Si copiamos y ejecutamos este código veremos que en la consola del navegador la variable 'counter' se va incrementando cada vez que pulsamos el botón, pero en nuestra página web esto no ocurre. Esto es debido a que no hay nada que indique que la variable 'counter' es reactiva, es decir, que provoque un renderizado de nuestro DOM.

ref()

- **ref()** es una forma de trabajar con la reactividad de Vue 3.
- **ref()**: Es una referencia reactiva. En nuestro ejemplo necesitamos un entero que sea "**rastreable**", para ello utilizaremos **ref()**, que es una de las formas de trabajar con la reactividad de Vue 3.
- **ref()** toma el argumento y lo devuelve envuelto dentro de un objeto con una propiedad **.value**, que luego puede usarse para acceder o mutar el valor de la variable reactiva.
- DOM: Cuando muta el estado reactivo, **el DOM se actualiza automáticamente.**
- En el **template** no es necesario acceder al **.value**, ya que el valor de la variable reactiva se puede acceder directamente.

Ejemplo:

```
<script setup>
  import { ref } from "vue"; //Importamos de Vue la funcion ref()
  let counter = ref(0);      //Le indicamos a Vue que counter es una
                             //variable reactiva. ref() nos devuelve un objeto.
  const increment = () => {
    counter.value++;         //al ser ahora un objeto para acceder al valor
    console.log(counter.value); //tenemos que utilizar .value
  };
</script>

<template>
  <h2>{{ counter }}</h2> <!--Aqui no es necesario utilizar .value -->
  <button @click="increment">Incrementar</button>
</template>
```

EJERCICIO 2:

Partiendo del ejemplo anterior, vamos a:

- Agregar un botón para disminuir el contador.
- Agregar un botón para resetear el contador.
- Pinta el contador en rojo cuando el valor sea menor a cero.
- Pinta el contador en azul cuando el valor sea mayor a cero.
- Pinta el contador en gris cuando el valor sea cero.

RESULTADO:

EJERCICIO 2

Curso de Vue3

-5

Incrementar

Decrementar

Resetear

Computed

Las expresiones en plantilla son muy convenientes, pero están pensadas para operaciones simples. Poner demasiada lógica en las plantillas puede hacerlas infladas y difíciles de mantener. Si vemos el código del ejercicio anterior intuimos que podría estar mejor. Por ejemplo:

```
<p>Has published books:</p>  
<span>{{ author.books.length > 0 ? 'Yes' : 'No' }}</span>
```

- Las propiedades computadas nos sirven para generar cálculos en nuestros componentes, por ejemplo, no se recomienda colocar demasiada lógica en nuestras plantillas HTML, ya que dificulta la interpretación de nuestros componentes.

- Por eso, para la lógica compleja que incluye datos reactivos, se recomienda utilizar una propiedad calculada (**computed**).
- Vamos a utilizar funciones flecha para definir las, pero teniendo en cuenta que **siempre nos tienen que devolver un valor ('return')**.

El ejercicio2 utilizando **computed** podría quedar de la siguiente manera, que como podemos observar queda mucho más legible:

```
<script setup>
import { ref, computed } from "vue"; //Importamos de Vue computed

let counter = ref(0);
const increment = () => {
  counter.value++;
};
const decrement = () => {
  counter.value--;
};
const reset = () => {
  counter.value=0;
};

const classCounter = computed(() => { //Aquí definimos la propiedad
computada
  if (counter.value == 0) return "cero"; //siempre tenemos que devolver
  if (counter.value > 0) return "positivo"; //un valor
  if (counter.value < 0) return "negativo";
});
</script>

<template>
  <h1>Curso de Vue3</h1><br>
  <h2 :class="classCounter"> //Aquí cambiamos la clase (color)
    {{ counter }}
  </h2>
  <button @click="increment">Incrementar</button>
  <button @click="decrement">Decrementar</button>
  <button @click="reset">Resetear</button>
</template>

<style>
.negativo {
  color: red;
}

.positivo {
```

```
color: blue;
}

.cero {
  color: gray;
}
</style>
```

NOTA

- En lugar de una propiedad calculada, podemos definir la misma función como un método. Para el resultado final, los dos enfoques son exactamente iguales. Sin embargo, **la diferencia es que las propiedades calculadas se almacenan en caché en función de sus dependencias reactivas.**
- Una propiedad calculada solo se volverá a evaluar cuando algunas de sus dependencias reactivas hayan cambiado.

```
// método
const classCounter = () => {
  if (counter.value == 0) return "cero";
  if (counter.value > 0) return "positivo";
  if (counter.value < 0) return "negativo";
};
```

Es necesario invocar al método

```
<h2 :class="classCounter()">
  {{ counter }}
</h2>
```

EJERCICIO 3.

Utilizando el ejemplo anterior vamos a:

- Agregar un array y su respectivo método y un botón 'Añadir' para almacenar los números favoritos del usuario.
- Pinta ese array utilizando **v-for** en una lista.
- Utilizamos **:disabled** en el botón 'Añadir', para que solo se pueda presionar si el array no contiene números repetidos. (utiliza una propiedad computada).

:disabled si es true el botón se bloquea:


```
<button @click="add" :disabled="true">Añadir</button>
```

Resultado:

Curso de Vue3

7

- 3
- 4
- 7

Entradas de formulario.

Cuando se trata de formularios en la interfaz, a menudo necesitamos sincronizar el estado de los elementos de entrada del formulario con el estado correspondiente en JavaScript.

La directiva **v-model** nos ayuda a realizar esto:

```
<input v-model="text">
```

Además, **v-model** se puede utilizar en entradas de diferentes tipos, como los elementos **<textarea>** y **<select>**.

- Los elementos de tipo texto **<input>** y **<textarea>** usan la propiedad **value** y el evento **input**;
- **<input type="checkbox">** y **<input type="radio">** usan la propiedad **checked** y el evento **change**;
- **<select>** usa la propiedad **value** y el evento **change**.

Uso básico

Texto. <input>

```
<script setup>
import { ref } from 'vue'

const message = ref('')
</script>

<template>
  <p>El mensaje es: {{ message }}</p>
  <input v-model="message" placeholder="Escribe" />
</template>
```

Resultado:

El mensaje es: HOLA MUNDO

Texto de varias líneas. <textarea>

```
<script setup>
import { ref } from 'vue'

const message = ref('')
</script>

<template>
  <span>Mensaje Multilinea:</span>
  <p style="white-space: pre-line;">{{ message }}</p>
  <textarea v-model="message" placeholder="Añade varias
lineas"></textarea>
</template>
```

Tengamos en cuenta que la interpolación interna en **<textarea>**, **<input>**, ... no funcionará. Utilizamos **v-model** en su lugar.

```
<!-- incorrecto -->
<textarea>{{ text }}</textarea>

<!-- correcto-->
<textarea v-model="text"></textarea>
```

Caja

Casilla de verificación única, valor booleano:

```
<script setup>
import { ref } from 'vue'

const checked = ref(true)
</script>

<template>
  <input type="checkbox" id="checkbox" v-model="checked" />
  <label for="checkbox">{{ checked }}</label>
```

```
</template>
```

Resultado:

☒ true ☐ false

También podemos vincular varias casillas de verificación a la misma matriz o establecer el valor:

```
<script setup>
import { ref } from 'vue'

const checkedNames = ref([])
</script>

<template>
  <div>Checked names: {{ checkedNames }}</div>

  <input type="checkbox" id="antonio" value="Antonio" v-
model="checkedNames" />
  <label for="antonio">Antonio</label>

  <input type="checkbox" id="juan" value="Juan" v-model="checkedNames" />
  <label for="juan">Juan</label>

  <input type="checkbox" id="miguel" value="Miguel" v-
model="checkedNames" />
  <label for="miguel">Miguel</label>
</template>
```

En este caso, la matriz siempre contendrá los valores de las casillas actualmente marcadas.

Checked names: ["Antonio", "Miguel"]

☒ Antonio ☐ Juan ☒ Miguel

Radio

```
<script setup>
import { ref } from 'vue'
```

```
const picked = ref('One')
</script>

<template>
  <div>Picked: {{ picked }}</div>

  <input type="radio" id="one" value="One" v-model="picked" />
  <label for="one">One</label>

  <input type="radio" id="two" value="Two" v-model="picked" />
  <label for="two">Two</label>
</template>
```

Picked: Two

☐ One ☒ Two

Selección

```
<script setup>
  import { ref } from 'vue'
  const selected = ref('')
</script>
<template>
  <span> Selected: {{ selected }}</span>
  <select v-model="selected">
    <option disabled value="">Please select one</option>
    <option>A</option>
    <option>B</option>
    <option>C</option>
  </select>
</template>
```

Lo mismo utilizando **v-for**:

```
<script setup>
import { ref } from 'vue'

const selected = ref('A')

const options = ref([
  { text: 'One', value: 'A' },
  { text: 'Two', value: 'B' },
  { text: 'Three', value: 'C' }
])
```

```
</script>

<template>
  <select v-model="selected">
    <option v-for="option in options" :value="option.value">
      {{ option.text }}
    </option>
  </select>

  <div>Selected: {{ selected }}</div>
</template>
```

Resultado:

•

Selected: B

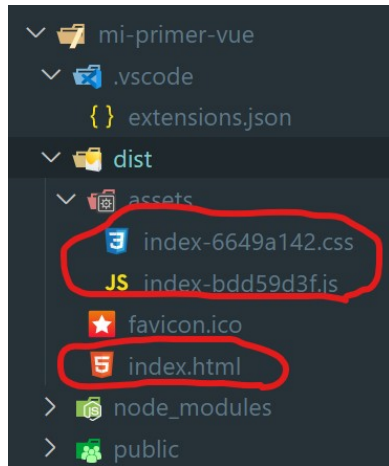
Deploy

Por último, en esta introducción a Vue3, vamos a compilar nuestro código y subirlo a Netlify que es una empresa de informática en la nube que ofrece una plataforma de desarrollo que incluye servicios de back-end de creación, implementación y sin servidor para aplicaciones web y sitios web dinámicos.

- <https://www.netlify.com/>

1.- Vamos a abrir el Terminal y paramos nuestro servidor, Pulsamos 'Ctrl+c'.

2.- Ahora compilamos nuestra aplicación con 'Vite', para ello escribimos en el terminal '**npm run build**'. Veremos que se nos crea una carpeta 'dist'. Si la abrimos podemos observar que tiene nuestro página .html, el fichero .js y el .css. Esta carpeta será la que subamos al servidor

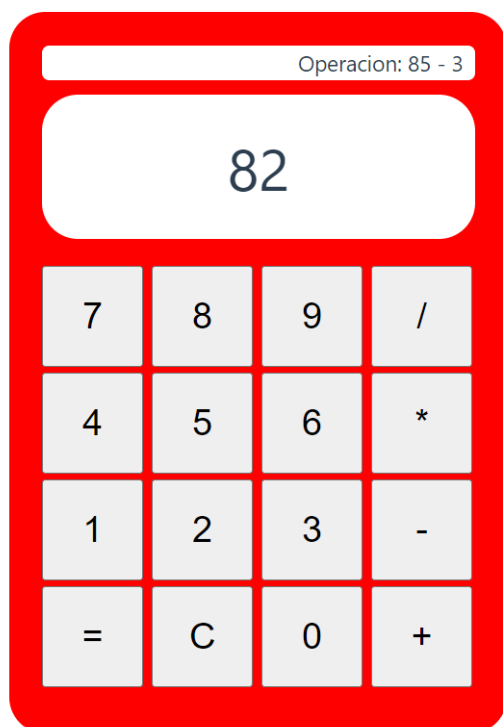


3.- Si escribimos en el terminal '**npm run preview**' se nos abrirá un servidor local pero de la carpeta 'dist'. Esto lo podemos utilizar para comprobar que se ha compilado correctamente.

Ejercicio4.

Utilizando la práctica de la calculadora que realizamos en la unidad de 'Eventos'. Vamos a modificarla para que funcione con Vue. No es necesario controlar los eventos del teclado. Tendremos una nueva línea donde nos aparecerán los operandos y la operación:

Nota: podéis comprobar el funcionamiento en: <https://vrrcalculadora.netlify.app/>



Tema elaborado a partir de:



- Material de la página oficial de Vue. <https://vuejs.org/>