

MIND HUB.

An aerial night photograph of a city, featuring a multi-lane highway with prominent light trails from vehicles. The city lights are visible in the background, and the overall image has a blue color cast. Decorative elements include pink L-shaped brackets at the top center, a dashed line at the top right, and a white bracket-like shape at the bottom center.


Vue.js




¿Qué es Vue?

Vue.js es un **framework** de JavaScript progresivo para la creación de interfaces de usuario.

Vue utiliza una sintaxis de templates basada en HTML para crear plantillas de interfaz de usuario, y luego utiliza una instancia de Vue para controlar y manipular el comportamiento de la interfaz de usuario en tiempo de ejecución. Vue también incluye características como la reactividad y el enlace de datos, lo que significa que puede crear aplicaciones dinámicas e interactivas sin tener que escribir mucho código JavaScript.



Vue es una opción popular para desarrollar aplicaciones web y móviles, y se puede utilizar tanto en proyectos individuales como en aplicaciones empresariales más grandes. Es fácil de aprender para los desarrolladores que ya tienen experiencia en HTML, CSS y JavaScript, y tiene una comunidad activa y una amplia gama de documentación y recursos disponibles en línea.




Instalación

Podemos instalar Vue incluyendo una etiqueta de **<script>** en tu página **HTML** apuntando a la dirección de las librerías de Vue.


```
. . .  
<!-- incluye Vue como una etiqueta de script -->  
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>  
. . .
```

crea una etiqueta que contenga la plantilla de tu interfaz de usuario. Esta plantilla debe ser una etiqueta de div.

```
. . .  
<!-- crea una plantilla de Vue -->  
<div id="app">  
  
</div>  
  
<script  
src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>  
. . .
```




En tu archivo JavaScript, crea una nueva **instancia** de Vue. Esto se hace con el método **createApp** de Vue, gracias al script que agregamos en nuestro archivo HTML pudiendo utilizar sus librerías.



```
//crea una instancia de Vue
const { createApp } = Vue
createApp({

//define el estado de la aplicación
  data() {
    return {
      message: '¡Hola, Mundo!'
    }
  },

//inicializa la instancia de Vue
}).mount('#app')
```



Data

En el contexto de Vue, **"data"** se refiere a las **variables de estado** de la aplicación. En una aplicación de Vue, puedes usar el objeto "data" para definir las variables que deseas usar en tu plantilla. Estas variables se conocen como **"propiedades reactivas"**, lo que significa que cuando cambian su valor, la plantilla se actualiza automáticamente para reflejar esos cambios.

Luego, en tu plantilla, puedes usar la sintaxis de **interpolación de Vue** (que es **{{ }}**) para mostrar el valor de la propiedad reactiva:

```
<div id="app" template>
  <!-- muestra un mensaje en pantalla -->
  <p>{{ message }}</p>
</div>
```

Por ejemplo, si tienes una aplicación de Vue que muestra un mensaje en pantalla, podrías usar una **propiedad reactiva** llamada **"message"** para controlar el contenido del mensaje:

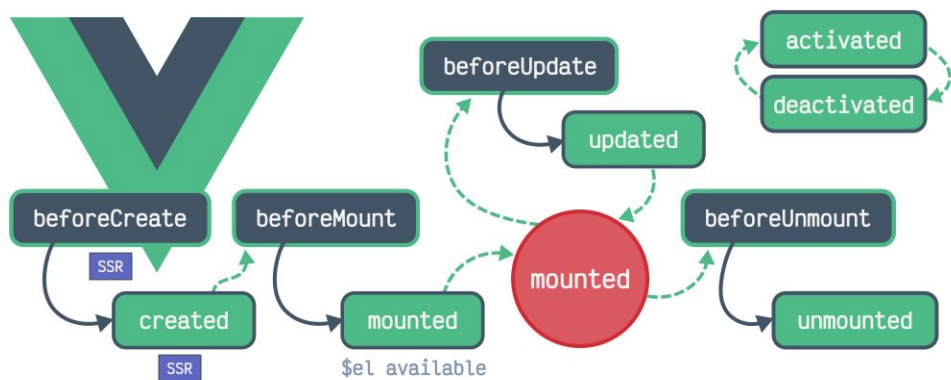
```
data() {
  return {
    message: '¡Hola, Mundo!'
  }
},
```

Ciclo de vida de Vue

El ciclo de vida de una aplicación de Vue se refiere a los diferentes momentos o etapas en los que ocurren eventos y acciones durante la vida útil de la aplicación. Vue tiene un conjunto de métodos de ciclo de vida que se ejecutan en diferentes momentos y que puedes usar para ejecutar código específico en esos momentos.

Puedes usar estos métodos de ciclo de vida para ejecutar código específico en diferentes momentos durante la vida útil de tu aplicación. Por ejemplo, puedes usar el método "mounted" para enlazar eventos de DOM una vez que la aplicación se haya montado en el DOM, o puedes usar el método "beforeDestroy" para limpiar cualquier evento o intervalo de tiempo que hayas definido en la aplicación antes de que se destruya.

Para más información sobre el ciclo de vida de Vue y cómo utilizar los métodos de ciclo de vida, puedes consultar la documentación oficial o buscar tutoriales y recursos en línea.



- **beforeCreate**: se ejecuta justo después de que se instancia una nueva instancia de Vue, pero antes de que se hayan creado las propiedades reactivas y se haya compilado la plantilla.
- **created**: se ejecuta justo después de que se hayan creado las propiedades reactivas y se haya compilado la plantilla. Es el momento adecuado para realizar operaciones asíncronas, como hacer peticiones a APIs.
- **beforeMount**: se ejecuta justo antes de que se monte la instancia de Vue en el DOM.
- **mounted**: se ejecuta justo después de que se monte la instancia de Vue en el DOM. Es el momento adecuado para realizar operaciones que requieran acceder al DOM, como enlazar eventos de DOM.
- **beforeUpdate**: se ejecuta justo antes de que se actualice la instancia de Vue con los cambios en el estado de la aplicación.
- **updated**: se ejecuta justo después de que se actualice la instancia de Vue con los cambios en el estado de la aplicación.
- **beforeUnmount**: se ejecuta justo antes de que se desmonte la instancia de Vue.
- **unmounted**: se ejecuta justo después de que se demonta la instancia de Vue.

Uso de Hooks (métodos del ciclo de vida)

Los hooks más usados son **created**, **mounted**, **unmounted** y **updated**.

En el ejemplo de abajo vamos a poder ver, que a nuestra instancia de Vue le agregamos un nuevo método (created). También podemos ver que estamos haciendo uso de una de las propiedades de nuestra instancia en este caso **message**, a la cual la llamamos utilizando **this** haciendo referencia a que quiero de esta instancia la propiedad message.

```
const { createApp } = Vue
createApp({
  data() {
    return {
      message: '¡Hola, Mundo!'
    }
  },
  created() {
    console.log(this.message);
  },
}).mount('#app')
```

Methods

En Vue, "**methods**" se refiere a un **objeto** que contiene funciones que puedes usar en tu aplicación. Puedes usar el objeto "methods" para definir funciones que manejen eventos, realicen cálculos o realicen cualquier otra tarea que necesites hacer en tu aplicación.

Por ejemplo, supongamos que tienes una aplicación que muestra un contador y un botón que permite incrementar el contador cuando se hace clic:

```
const { createApp } = Vue
createApp({
  data() {
    return {
      count: 0
    }
  },
  methods: {
    // define una función para incrementar el contador
    incrementCount() {
      this.count++;
    }
  }
}).mount('#app')
```

Luego, en tu plantilla, puedes usar la sintaxis de **enlace de eventos** de Vue (que es **@** o **v-on:**) para **enlazar** el **evento "click"** del botón a la función "incrementCount":

```
<div id="app">
  <button @click="incrementCount">Incrementar contador</button>
  <p>Contador: {{ count }}</p>
</div>
```

Cuando el usuario haga clic en el botón, se ejecutará la función "incrementCount" y el contador se incrementará en uno.

Es importante tener en cuenta que las funciones del objeto "methods" tienen acceso a las propiedades reactivas de la instancia de Vue a través del **"this"** keyword. Esto significa que puedes usar las propiedades reactivas dentro de las funciones de "methods" para leer o modificar el estado de la aplicación.

Computed

```
...  
data() {  
  return {  
    products: [  
      { name: 'Producto 1', price: 10 },  
      { name: 'Producto 2', price: 20 },  
      { name: 'Producto 3', price: 30 }  
    ],  
  },  
}  
  
computed: {  
  // define una propiedad computada para calcular el  
  // precio total  
  totalPrice() {  
    return this.products.reduce((total, product) => total  
      + product.price, 0);  
  }  
}  
...  

```

En Vue, **"computed"** es un **objeto** que contiene propiedades que son calculadas a partir de otras propiedades de la aplicación. Las propiedades computadas se conocen también como **"propiedades calculadas"**, y puedes usarlas para realizar cálculos o transformaciones de datos de forma declarativa.

Una de las **principales ventajas** de las propiedades computadas es que son reactivas, lo que significa que **se actualizan automáticamente cuando cambian las propiedades de las que dependen**. Esto puede ser muy útil cuando necesitas realizar cálculos complejos que involucren varias propiedades diferentes.

Por ejemplo, supongamos que tienes una aplicación que muestra una lista de productos y que deseas calcular el precio total de la lista:

Luego, en tu plantilla, puedes usar la sintaxis de **interpolación de Vue** (que es `{{ }}`) para mostrar el valor de la **propiedad computada**:

```
<div id="app">
  <p>Precio total: {{ totalPrice }}</p>
</div>
```

Cuando la aplicación se cargue, el mensaje "Precio total: 60" se mostrará en pantalla. Si cambias los precios de los productos en el objeto "products", el precio total también se actualizará automáticamente.

Es importante tener en cuenta que las propiedades computadas **son sólo para lectura**, lo que significa que no puedes modificar su valor directamente. Si necesitas modificar el valor de una propiedad computada, puedes hacerlo a través de una función del objeto "methods".

"v-model" es una directiva de Vue que se usa para crear un enlace bidireccional entre un elemento de formulario y una propiedad reactiva de la aplicación. Esto significa que "v-model" sincroniza el valor del elemento de formulario con la propiedad reactiva y viceversa.

La sintaxis de "v-model" es la siguiente:

```
<div id="app" template>  
  <input type="text" v-model="property">  
</div>
```

v-model

Donde "property" es la propiedad reactiva que se está enlazando al elemento de formulario.

En el siguiente ejemplo, supongamos que tienes una aplicación que permite a los usuarios introducir su nombre en un campo de texto y que deseas almacenar el nombre en una propiedad reactiva llamada "name":

```
...  
data() {  
  return {  
    name: ""  
  },  
  ...  
}
```

En tu archivo **HTML**, puedes usar "v-model" para enlazar el valor del campo de texto a la propiedad "name":

```
...  
<div id="app" template>  
  <label>Nombre:</label>  
  <input v-model="name">  
</div>  
...
```

Cuando el usuario introduce su nombre en el campo de texto, el valor de la propiedad "name" se actualizará automáticamente con el valor del campo de texto. Si cambias el valor de la propiedad "name" en el código JavaScript, el valor del campo de texto también se actualizará automáticamente.

"v-model" es una forma conveniente de crear enlaces **bidireccionales** entre elementos de formulario y propiedades reactivas.

v-bind

"**v-bind**" es una directiva de Vue que se usa para enlazar el valor de una expresión a un atributo de un elemento HTML. La sintaxis de "v-bind" es la siguiente:

```
<div id="app" template>  
  <element v-bind:attribute="expression">  
</div>
```

Donde "**attribute**" es el atributo del elemento al que se le está enlazando el valor y "**expresión**" es la expresión que se evalúa para obtener el valor que se enlazaré al atributo.

En el siguiente ejemplo, supongamos que tienes una aplicación que muestra un mensaje y que deseas cambiar el color del mensaje en función de una propiedad reactiva llamada "color":

```
. . .  
data() {  
  return {  
    message: 'Hola, mundo!',  
    color: 'red'  
  }  
},  
. . .
```

En tu archivo **HTML**, puedes usar "v-bind" para enlazar el atributo "style" del elemento "p" a la propiedad "color":

```
. . .  
<div id="app" template>  
  <p v-bind:style="{ color: color }">  
    {{ message }}  
  </p>  
</div>  
. . .
```

Cuando la aplicación se cargue, el mensaje se mostrará en rojo. Si cambias el valor de la propiedad "color" en el código JavaScript, el color del mensaje también se actualizará automáticamente.

v-for

"v-for" es una directiva de Vue que se usa para iterar sobre una colección de elementos y renderizar un elemento HTML por cada elemento de la colección. La sintaxis de "v-for" es la siguiente:

```
<div id="app" template>
  <element v-for="(item, index) in items" :key="index">
    {{ item }}
  </element>
</div>
```

Donde "items" es la colección que se está iterando y "item" es el elemento actual de la colección. "index" es opcional y representa el índice del elemento actual en la colección.

En el siguiente ejemplo, supongamos que tienes una lista de productos y quieres **renderizar** una lista de elementos "li" con el nombre de cada producto:

```
...  
data() {  
  return {  
    products: [  
      { name: 'Producto 1' },  
      { name: 'Producto 2' },  
      { name: 'Producto 3' }  
    ]  
  },  
  ...  
}
```

En tu archivo **HTML**, puedes usar "v-for" para **iterar** sobre la lista de productos y renderizar un elemento "li" por cada producto:

```
...  
<div id="app" template>  
  <ul>  
    <li v-for="product in products" :key="product.name">  
      {{ product.name }}  
    </li>  
  </ul>  
</div>  
...
```

v-if y v-else

"v-if" y "v-else" son directivas de Vue que se usan para controlar la renderización de elementos HTML en función de una condición. "v-if" renderiza el elemento si la condición es verdadera, mientras que "v-else" renderiza el elemento si la condición es falsa.

La sintaxis de "v-if" es la siguiente:

```
<div id="app" template>  
  <element v-if="expression">  
  </div>
```

Donde "expresión" es la expresión que se evalúa para determinar si el elemento se renderiza o no.

En el siguiente ejemplo, supongamos que tienes una aplicación que muestra un mensaje de bienvenida si el usuario está conectado y un mensaje de advertencia si no lo está:

```
. . .  
data() {  
  return {  
    isLoggedIn: true  
  },  
  . . .  
}
```

Cuando la aplicación se cargue, se mostrará el mensaje de bienvenida porque la propiedad "isLoggedIn" es verdadera. Si cambias el valor de "isLoggedIn" a false, se mostrará el mensaje de advertencia.

En tu archivo **HTML**, puedes usar "v-if" y "v-else" para controlar la renderización de los mensajes:

```
. . .  
<div id="app" template>  
  <p v-if="isLoggedIn">  
    Bienvenido!  
  </p>  
  <p v-else>  
    Por favor, inicia sesión para continuar.  
  </p>  
</div>  
. . .
```

Es importante tener en cuenta que "v-if" y "v-else" no solo controlan la renderización de los elementos, sino que también los agrega o elimina del DOM en función de la condición. Esto significa que "v-if" es más pesado que otras opciones como "v-show", que solo oculta el elemento en lugar de eliminarlo del DOM. Si necesitas mostrar y ocultar elementos con frecuencia, es recomendable usar "v-show" en lugar de "v-if".

¡Muchas gracias!

MIND
HUB.