



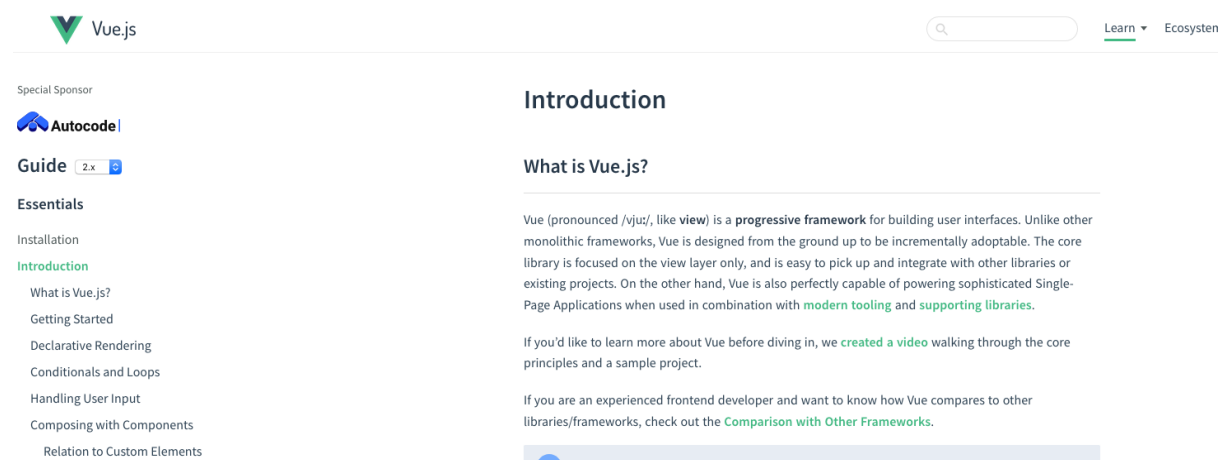
- Es un framework progresivo para desarrollar una interfaz de usuario.
- Muy ligero 19KB.
- Funciona también desde el lado del cliente.
- Implementado en JavaScript.

Arrancando una instancia

Lo primero es visitar su página <https://vuejs.org/> y pinchar en el get started.

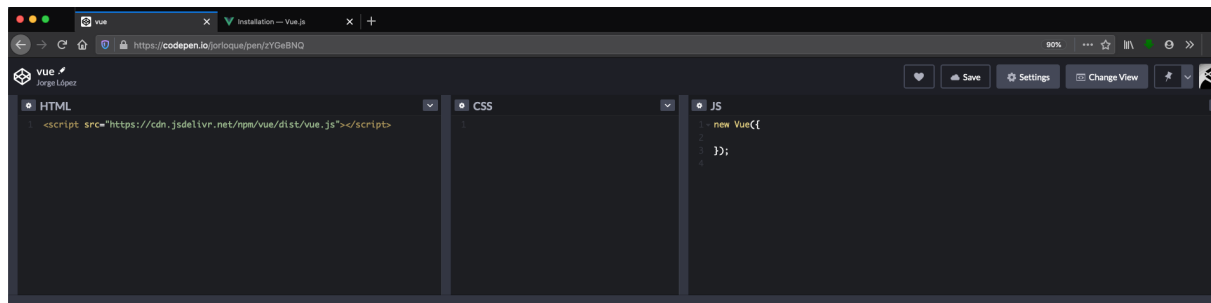


Y nos llevará aquí:

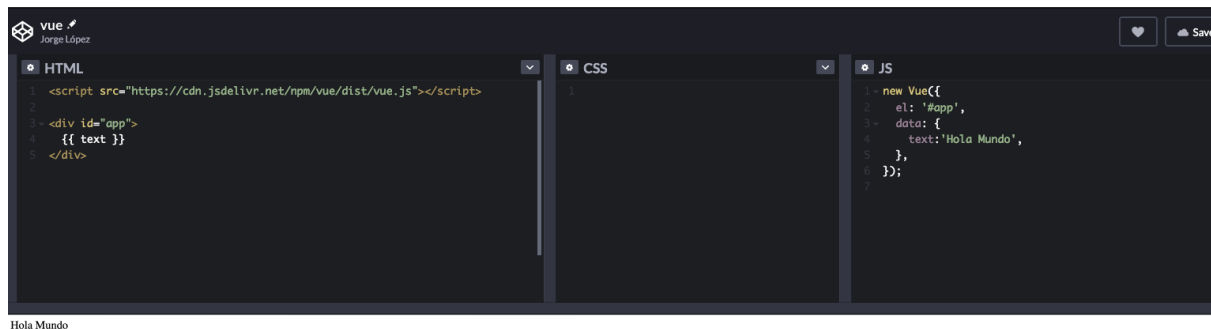


Nos manda a la documentación, iremos directamente a la instalación. Vamos a hacer dos pruebas, la primera será en la página de <https://codepen.io> y la segunda en local.

Para la primera utilizaremos la cuenta de GitHub para loguearnos y en el html emplearemos el CDN que tenemos en la página de Vue en el apartado de instalación.



Rellenaremos todo para que quede de la siguiente manera:



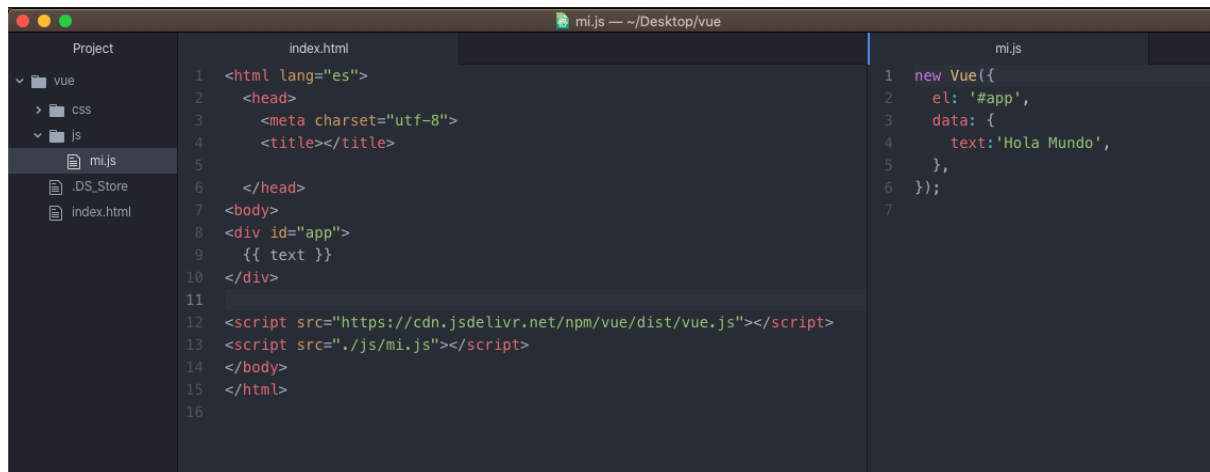
El html

```
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
//el id relaciona en el js el elemento
<div id="app">
//text es el nombre de los datos que va a mostrar
  {{ text }}
</div>
```

El JS

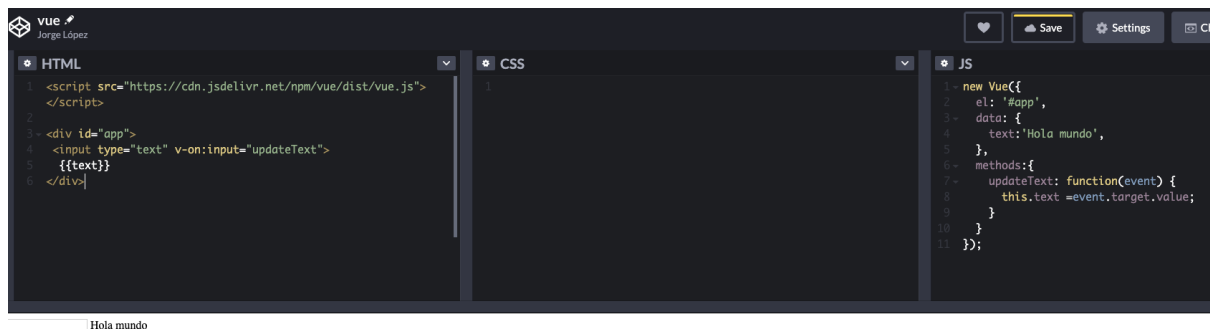
```
new Vue({
//instancia que creamos
//el elemento app lo relacionamos con el id en el html
  el: '#app',
//datos a mostrar
  data: {
    text: 'Hola Mundo',
  },
});
```

Utilizando un IDE:



Básicos de Vue

Ahora vamos a añadir un input pasándole un texto, así veremos lo qué es una directiva, éstas cuando ocurre un evento que reaccione.



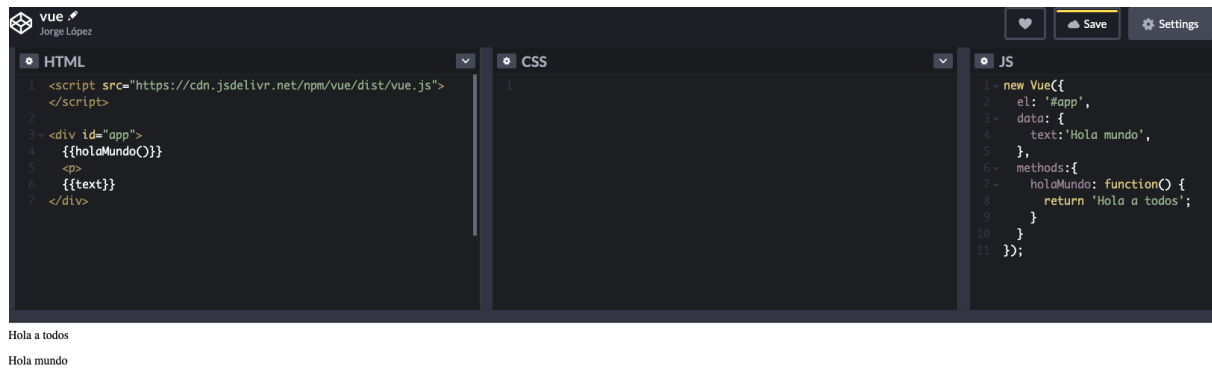
En HTML:

```
//la directiva input llamará al método updateText
<input type="text" v-on:input="updateText">
```

En JS:

```
methods: {
  //cómo llamamos al evento
  updateText: function(event) {
    //this hace referencia a la instancia Vue (de id = app) y el que ha disparado el evento.
    this.text = event.target.value;
  }
}
```

Ahora para añadir el típico ejemplo “Hola mundo”, es tan fácil como crear una función y llamarla desde el html.



The screenshot shows a web application running in a browser. The browser's address bar displays 'vue' and the user's name 'Jorge López'. The page content is divided into three sections: HTML, CSS, and JS. The HTML section contains a script tag for Vue.js and a div with an id of 'app' containing a holaMundo() function call and a text element. The CSS section is empty. The JS section contains a new Vue instance with a data property 'text' set to 'Hola mundo' and a holaMundo method that returns 'Hola a todos'. The browser's console shows the output 'Hola a todos' and 'Hola mundo'.

```
vue
Jorge López

HTML
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js">
2 </script>
3 <div id="app">
4   {{holaMundo()}}
5   <p>
6     {{text}}
7   </p>
8 </div>

CSS
1

JS
1 new Vue({
2   el: '#app',
3   data: {
4     text: 'Hola mundo',
5   },
6   methods: {
7     holaMundo: function() {
8       return 'Hola a todos';
9     }
10  }
11 });
```

Hola a todos
Hola mundo

Si lo que quisiera es devolver la propiedad en el **return**, entonces debería llamarla como **this.text**.

Interacción y Binding

Partiremos de una situación sencilla:

HTML: tenemos un párrafo con un texto.

CSS: dos clases red y blue que ponen un color de fondo.

JS: una instancia de Vue con un elemento data y una variable text y otra para enlazar a la clase CSS.



```
HTML
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js">
2 </script>
3
4 <div id="app">
5   <p v-bind:class="color"> Soy un parrafo </p>
6 </div>

CSS
1 p {
2   padding: 10px;
3 }
4
5 .azul {
6   background: blue;
7   color: green;
8 }
9
10 .red {
11   background: red;
12   color: green;
13 }

JS
1 new Vue({
2   el: '#app',
3   data: {
4     text: 'Hola mundo',
5     color: 'azul',
6   },
7 });
```

Soy un parrafo

Y si cambiamos a la clase rojo en el vue:



```
HTML
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js">
2 </script>
3
4 <div id="app">
5   <p v-bind:class="color"> Soy un parrafo </p>
6 </div>

CSS
1 p {
2   padding: 10px;
3 }
4
5 .azul {
6   background: blue;
7   color: green;
8 }
9
10 .rojo {
11   background: red;
12   color: green;
13 }

JS
1 new Vue({
2   el: '#app',
3   data: {
4     text: 'Hola mundo',
5     color: 'rojo',
6   },
7 });
```

Soy un parrafo

Para poder enlazar de una forma dinámica al **css** tenemos que hacer un **bind** a la propiedad de la instancia.

Ahora vamos a establecer una comunicación bidireccional porque hasta el momento sólo nos hemos comunicado desde el JS al HTML, para poder hacerlo bidireccional tenemos que añadir en el input la propiedad v-model, entonces lo que escribamos aquí se modificará en la instancia de Vue y a su vez en la variable data.



Llamamos a la propiedad `framework` desde el html y modificamos su valor en el vue. Con esta directiva podemos modificar nuestra instancia de Vue desde el html, podríamos decir que “vue = html”.

Las directivas vistas hasta ahora son:

- vi-bind: poder enlazar dinámicamente a un css.
- vi-on: podemos hacer reaccionar a eventos y también modificar el vue desde el html, cuando ocurra un evento realiza esta acción.
- v-model: conexión bidireccional.
- v-once: renderiza una sola vez el contenido.
- v-cloak: evita cargas de contenido.
- v-link: interpreta etiquetas html en el vue para pasarlas al html correctamente.
- v-text: reemplaza el contenido de un html pasándole una propiedad de la instancia de vue.



```
HTML
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4   <p v-text="text">Soy un párrafo</p>
5 </div>

CSS
1

JS
1 new Vue({
2   el: '#app',
3   data: {
4     text: 'Hola mundo',
5     color: 'rojo',
6   },
7 });
```

Hola mundo

Al utilizar la directiva v-text reemplaza el texto html por el valor de la propiedad que tenemos en el vue. Aunque si en vez de una propiedad quiero escribir un texto directamente lo pondré con comillas simples.



```
HTML
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4   <p v-text="Soy lo que hay aqui">Soy un párrafo</p>
5 </div>

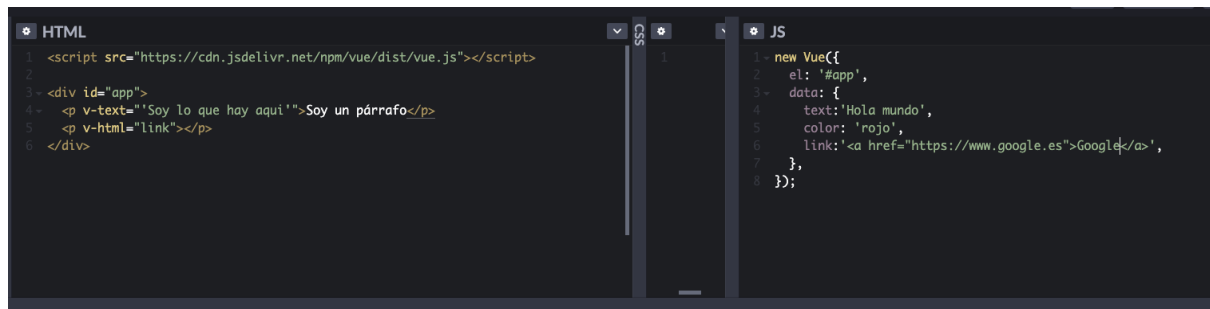
CSS
1

JS
1 new Vue({
2   el: '#app',
3   data: {
4     text: 'Hola mundo',
5     color: 'rojo',
6   },
7 });
```

Soy lo que hay aqui

Ahora si tuviéramos una lista que renderiza una lista de nombres de personas, cada vez que cargamos el contenido vue tiene una sobrecarga, para ello utilizaremos v-once, lo cargará sólo una vez si no tiene que volver a cargarlo más y se desvincula.

Si tenemos un atributo html dentro de nuestra instancia de vue no podemos pasarla porque no lo sabría interpretar, para ello utilizamos la directiva v-html.



```
HTML
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4   <p v-text="\"Soy lo que hay aqui\"">Soy un párrafo</p>
5   <p v-html="link"></p>
6 </div>

CSS
1

JS
1 new Vue({
2   el: '#app',
3   data: {
4     text: 'Hola mundo',
5     color: 'rojo',
6     link: '<a href="https://www.google.es">Google</a>',
7   },
8 });
```

soy lo que hay aqui

[Google](https://www.google.es)

La directiva v-cloak, le dice a vue: “no cargues hasta que todo el contenido que esté dentro de ti no haya cargado”. Para evitar ver trozos cargados y otros no.



```
HTML
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app" v-cloak>
4   <p>Hola, {{name}} Vue se ha cargado correctamente </p>
5 </div>
6
7 <button onclick="loadValue()">Cargar vue</button>

CSS
1 [v-cloak] > * {
2   display: none;
3 }
4
5 [v-cloak]::before {
6   content: "Aún no se ha cargado";
7 }

JS
1 function loadValue() {
2   new Vue({
3     el: '#app',
4     data: {
5       name: 'Fran',
6       text: 'Hola mundo',
7       color: 'rojo',
8       link: '<a href="https://www.google.es">Google</a>',
9     },
10   })
11 }
```

Aún no se ha cargado

Cargar vue

Cuando hagamos click en el botón como ya le habrá dado tiempo más que de sobra para cargar entonces mostrará el mensaje de que se ha cargado correctamente.



```
HTML
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app" v-cloak>
4   <p>Hola, {{name}} Vue se ha cargado correctamente </p>
5 </div>
6
7 <button onclick="loadValue()">Cargar vue</button>

CSS
1 [v-cloak] > * {
2   display: none;
3 }
4
5 [v-cloak]::before {
6   content: "Aún no se ha cargado";
7 }

JS
1 function loadValue() {
2   new Vue({
3     el: '#app',
4     data: {
5       name: 'Fran',
6       text: 'Hola mundo',
7       color: 'rojo',
8       link: '<a href="https://www.google.es">Google</a>',
9     },
10   })
11 }
```

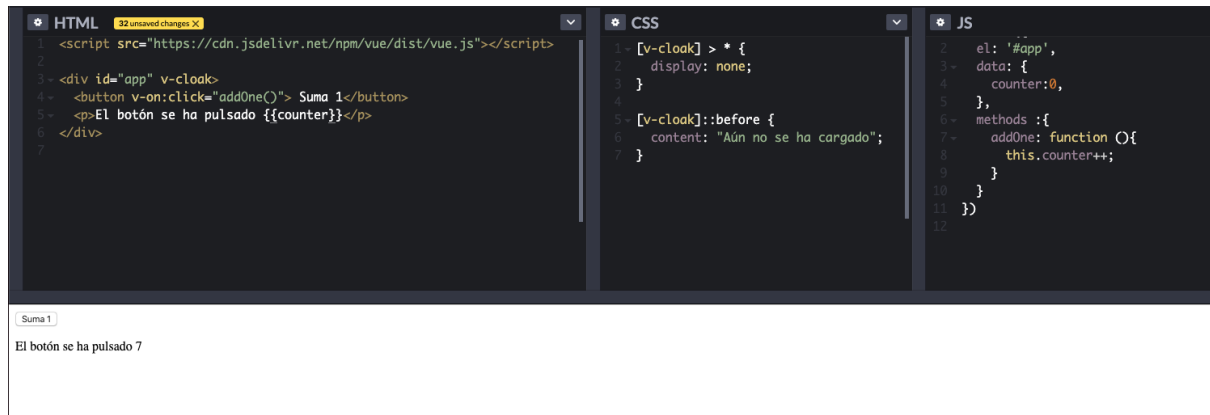
Hola, Fran Vue se ha cargado correctamente

Cargar vue

Eventos

Modificadores de eventos

Cuando marcamos un evento modificamos nuestra instancia.



```
HTML
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app" v-cloak>
4   <button v-on:click="addOne()"> Suma 1</button>
5   <p>El botón se ha pulsado {{counter}}</p>
6 </div>
7
CSS
1 [v-cloak] > * {
2   display: none;
3 }
4
5 [v-cloak]::before {
6   content: "Aún no se ha cargado";
7 }
8
JS
1 el: '#app',
2 data: {
3   counter: 0,
4 },
5 methods: {
6   addOne: function () {
7     this.counter++;
8   }
9 }
10
11 })
12
```

Suma 1

El botón se ha pulsado 7

Cada vez que hagamos click modificaremos el valor de la propiedad **counter** y la mostraremos en el html.

Vamos a definir dónde está el cursor en un cierto momento utilizando eventos.



```
HTML
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4   <button v-on:click="addOne()">Suma 1</button>
5   <p>El botón se ha pulsado {{counter}} veces</p>
6   <p>La posición horizontal del cursor es: {{cursorPositionX}} <b></b></p>
7 </div>
8
9
CSS
1 [v-cloak] > * {
2   display: none;
3 }
4
5 [v-cloak]::before {
6   content: "Aún no se ha cargado";
7 }
8
JS
1 new Vue({
2   el: '#app',
3   data: {
4     counter: 0,
5     cursorPositionX: 'desconocida',
6   },
7   methods: {
8     addOne: function(event){
9       this.counter++;
10      this.cursorPositionX = event.x;
11    }
12  }
13 });
14
```

Suma 1

El botón se ha pulsado 0 veces

La posición horizontal del cursor es: desconocida

Y cuando hagamos click nos determinará la posición x.

HTML

```

1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4   <button v-on:click="addOne">Suma 1</button>
5   <p>El botón se ha pulsado {{counter}} veces</p>
6   <p>La posición horizontal del cursor es: {{cursorPositionX}} <b></b></p>
7 </div>
8
9

```

CSS

JS

```

1 new Vue({
2   el: '#app',
3   data: {
4     counter: 0,
5     cursorPositionX: 'desconocida',
6   },
7   methods: {
8     addOne: function(event) {
9       this.counter++;
10      this.cursorPositionX = event.x;
11    }
12  }
13 });
14

```

Suma 1

El botón se ha pulsado 4 veces

La posición horizontal del cursor es: 18

Captura el evento en la posición x, cuando éste ocurre. Ahora a lo mejor si quisiéramos pasarle un parámetro al evento, vamos a añadir 10 clics al botón cada vez que lo pulsemos. \$event: capturamos todos los eventos originales del dom para poder pasarlos al método de nuestra instancia de vue.

HTML

```

1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4   <button v-on:click="addOne(10,$event)">Suma 1</button>
5   <p>El botón se ha pulsado {{counter}} veces</p>
6   <p>La posición horizontal del cursor es: {{cursorPositionX}} <b></b></p>
7 </div>
8
9

```

CSS

JS

```

1 new Vue({
2   el: '#app',
3   data: {
4     counter: 0,
5     cursorPositionX: 'desconocida',
6   },
7   methods: {
8     addOne: function(addition, event) {
9       this.counter+=addition;
10      this.cursorPositionX = event.x;
11    }
12  }
13 });
14

```

Suma 1

El botón se ha pulsado 40 veces

La posición horizontal del cursor es: 32

Ahora si quisiéramos modificadores de eventos y teclado modificaremos nuestro código de la siguiente forma:

HTML

```

1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4   <form v-on:submit.prevent="addItem">
5     Añade un objeto a la lista
6     <input type="text" v-model="listItem"/>
7     <button type="submit" class="btn">Enviar</button>
8   </form>
9   <br/>
10  Lista
11  {{list}}
12 </div>
13
14

```

CSS

JS

```

1 new Vue({
2   el: '#app',
3   data: {
4     listItem: '',
5     list: [],
6   },
7   methods: {
8     addItem: function() {
9       this.list.push(this.listItem);
10      this.listItem = '';
11    }
12  }
13 });
14

```

Añade un objeto a la lista

potatas

Enviar

Lista []

Con el **prevent** no recargará la página, y así cada vez que hagamos un **submit** sólo recargará el valor **list** NO recargará la página, porque si hiciera la recarga la lista se quedaría vacía y restaurar todo.

```
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4   <form v-on:submit.prevent="addItem">
5     Añade un objeto a la lista
6     <input type="text" v-model="listItem"/>
7     <button type="submit" class="btn">Enviar</button>
8   </form>
9   <br/>
10  Lista
11  {{list}}
12 </div>
13
14
15 new Vue({
16   el: '#app',
17   data: {
18     listItem:'',
19     list: [],
20   },
21   methods: {
22     addItem: function() {
23       this.list.push(this.listItem);
24       this.listItem = '';
25     },
26   },
27 });
```

Añade un objeto a la lista Enviar

Lista ["patatas", "peras", "limones"]

Modificadores de teclas:

Hablando de modificadores de teclas, si quisiéramos que al apretar la tecla "t", enviar un ítem a la lista, para ello, tal y como lo tenemos cada vez que pulsamos una tecla ésta se añadiría a la lista:

```
3 <div id="app">
4   <form v-on:submit.prevent="addItem">
5     Añade un objeto a la lista
6     <input type="text" v-model="listItem" v-on:keyup="addItem"/>
7     <button type="submit" class="btn">Enviar</button>
8   </form>
9   <br/>
10  Lista
11  {{list}}
12 </div>
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
```

Añade un objeto a la lista Enviar

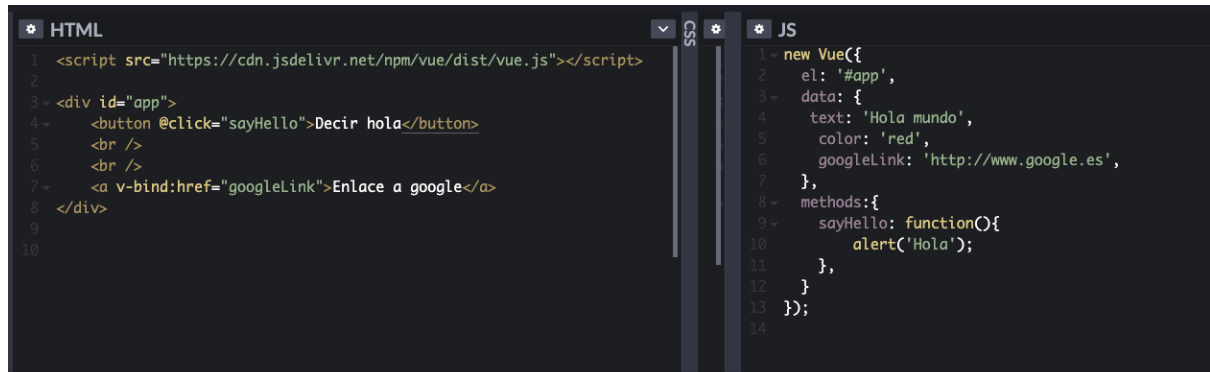
Lista ["a", "s", "d", ""]

Pero si lo que queremos es que fuera una en particular (en concreto el enter), funciona por código de tecla, y se añade:

```
em" v-on:keyup.13="addItem"/>  
Enviar</button>
```

Los códigos de tecla tienen unos atajos, por ejemplo `keyup.enter`.

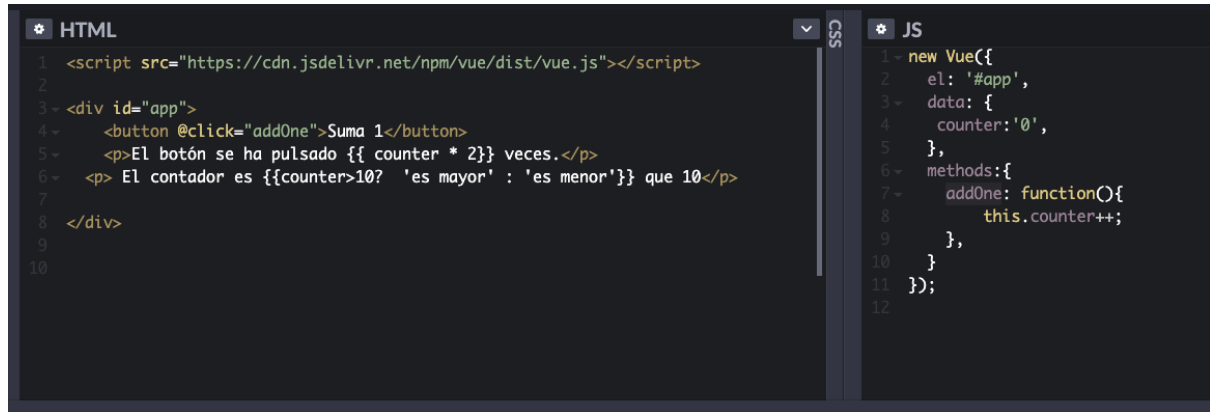
Un pequeño atajo que podemos utilizar, que nos ofrece Vue es `@click` que substituye a `v-on:click` :



```
HTML  
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>  
2  
3 <div id="app">  
4   <button @click="sayHello">Decir hola</button>  
5   <br />  
6   <br />  
7   <a v-bind:href="googleLink">Enlace a google</a>  
8 </div>  
9  
10  
CSS  
JS  
1 new Vue({  
2   el: '#app',  
3   data: {  
4     text: 'Hola mundo',  
5     color: 'red',  
6     googleLink: 'http://www.google.es',  
7   },  
8   methods: {  
9     sayHello: function() {  
10       alert('Hola');  
11     },  
12   },  
13 });  
14
```

Plantillas con JS

Los **templates** en JS se refieren a embeber código JS en nuestra plantilla html dentro de las llaves.



The screenshot shows a code editor with two panels. The left panel is labeled 'HTML' and contains the following code:

```
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4   <button @click="addOne">Suma 1</button>
5   <p>El botón se ha pulsado {{ counter * 2 }} veces.</p>
6   <p> El contador es {{counter>10? 'es mayor' : 'es menor'}} que 10</p>
7 </div>
8
9
10
```

The right panel is labeled 'JS' and contains the following code:

```
1 new Vue({
2   el: '#app',
3   data: {
4     counter: '0',
5   },
6   methods: {
7     addOne: function() {
8       this.counter++;
9     },
10  }
11 });
12
```

Suma 1

El botón se ha pulsado 0 veces.

El contador es es menor que 10

Por ejemplo, la definición de variables no funciona, tienen que ser cosas muy sencillas.

Propiedades computadas

Nos sirve para limpiar un poco la sintaxis del template, dar mayor facilidad de lectura. Por ejemplo, el ternario que habíamos hecho en el ejercicio anterior lo podríamos pasar a una propiedad computada en nuestra instancia de vue.



```
HTML
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4   <button @click="addOne">Suma 1</button>
5   <p>El botón se ha pulsado {{ counter * 2 }} veces.</p>
6   <p> El contador es {{ resultadoComputado }} que 10</p>
7
8 </div>
9
10
JS
1 new Vue({
2   el: '#app',
3   data: {
4     counter: '0',
5   },
6   methods: {
7     addOne: function() {
8       this.counter++;
9     },
10  },
11  computed: {
12    resultadoComputado: function() {
13      return this.counter > 10 ? 'mayor' : this.counter;
14    }
15  }
16 })
```

Suma 1

El botón se ha pulsado 0 veces.

El contador es 0 que 10

Aunque también podríamos hacerlo como un método, las propiedades computadas son evaluadas cada vez que la propiedad cambia, mientras que el método siempre se ejecuta, da igual que haya cambiado o no.

Watchers

Estas propiedades están pensadas para funcionar de manera asíncrona, están pendientes de unas propiedades de nuestra instancia, constantemente la está vigilando. Vamos a simular la búsqueda en un carrito de la compra



```
HTML
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4   Buscar en tu carrito
5   <input type="text" v-model="searchCartItem"/>
6
7   <p v-if="searching">Buscando ...</p>
8   <ul v-else class="collection">
9     <li class="collection-item" v-for="item in shopList">
10      {{ item }}</li>
11    </ul>
12 </div>
13
14
JS
1 new Vue({
2   el: '#app',
3   data: {
4     searchCartItem: '',
5     shopList: [],
6     searching: false,
7   },
8   watch: {
9     searchCartItem: function() {
10      setTimeout(() => {
11        this.shopList = ['Tomates', 'Queso', 'Leche'];
12        this.searching = false;
13      }, 500);
14    }
15  }
16 });
17
```

Buscar en tu carrito

Partiremos de la siguiente situación: el ítem que añadimos tiene que ser el mismo que en el **watch** y dentro definimos un **timeout** en el que simulamos una llamada a una API que tarda un poquito. Y en el html metemos un if-else, si **searching** es verdadero que aparezca Buscando y, sino que aparezca la lista de la compra.

Si yo introduzco un valor en la caja de text, entonces como ha cambiado cargará los valores de la nueva lista.

```
3 <div id="app">
4   Buscar en tu carrito
5   <input type="text" v-model="searchCartItem"/>
6
7   <p v-if="searching">Buscando ...</p>
8   <ul v-else class="collection">
9     <li class="collection-item" v-for="item in shopList"> {{item}} </li>
10  </ul>
11  <pre>{{data}}</pre>
12 </div>
13
14
```

Buscar en tu carrito

- Tomates
- Queso
- Leche

```
{
  "searchCartItem": "a",
  "shopList": [
    "Tomates",
    "Queso",
    "Leche"
  ],
  "searching": false
}
```

Como el valor ha cambiado vue sabía que tenía que hacer ciertas cosas como por ejemplo actualizar la lista de valores de la lista.

CSS dinámico

Vamos a añadir clases dinámicas a nuestros elementos html dependiendo de cómo están nuestros elementos en la instancia de Vue. Si yo hago click en el primer cuadrado veré como va cambiando el valor de la variable **isCircle**.

```
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4   <pre>{{data}}</pre>
5   <div class="element" @click="isCircle = !isCircle"></div>
6   <div class="element"></div>
7   <div class="element"></div>
8 </div>
9
10
```

```
1 * {
2   padding: 0px;
3   margin: 0px;
4 }
5
6 .element {
7   border: 1px solid;
8   width: 100px;
9   height: 20px;
10  float: left;
11  padding: 10px;
12  margin: 10px;
13
14
15 }
```

```
1 new Vue({
2   el: '#app',
3   data: {
4     isCircle: false,
5   },
6 });
7
```

"isCircle": true

Ahora le añadiremos la clase dinámica a nuestro elemento, primero tendremos que hacer un bind y si isCircle es verdadero que le añada la clase **circle**.

```
HTML
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4   <pre>{{data}}</pre>
5   <div class="element" @click="isCircle = !isCircle" :class="{circle:
6     isCircle}">dfsfs</div>
7   <div class="element"></div>
8   <div class="element"></div>
9 </div>
10
```

```
CSS
1
2 padding: 0px;
3 margin: 0px;
4 }
5
6 .element {
7   border: 1px solid;
8   width: 100px;
9   height: 20px;
10  float: left;
11  padding: 10px;
12  margin: 10px;
13
14
15 }
16
17 .circle {
18   background-color: #ddd;
19   border-radius: 100%;
20 }
```

{ "isCircle": true }

dfsfs

Clases CSS

¿Qué pasa si queremos añadir una propiedad y la otra propiedad no se cumple? si no es círculo que sea la otra clase. Simplemente la añadiremos a continuación del bind.

```
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4   <pre>{{data}}</pre>
5   <div class="element" @click="isCircle = !isCircle"
6     :class="{circle: isCircle, roundCorners: !isCircle}"></div>
7   <div class="element"></div>
8   <div class="element"></div>
9 </div>
10
11
```

```
6- .element {
7   border: 1px solid;
8   width: 100px;
9   height: 20px;
10  float: left;
11  padding: 10px;
12  margin: 10px;
13 }
14
15- .circle {
16   background-color: #ddd;
17   border-radius: 100%;
18 }
19
20- .roundCorners {
21   border-radius: 10px;
22 }
```

"isCircle": false



Para mejorar esto, podemos incluir una computed property.

```
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4   <pre>{{data}}</pre>
5   <div class="element" @click="isCircle = !isCircle"
6     :class="borderRadius"></div>
7   <div class="element"></div>
8   <div class="element"></div>
9 </div>
10
11
```

```
1- new Vue({
2   el: '#app',
3   data: {
4     isCircle: false,
5   },
6   computed: {
7     borderRadius: function() {
8       return {circle: this.isCircle,
9         roundCorners: !this.isCircle
10      }
11    }
12  }
13 });
14
```

"isCircle": false

