UD09 - Introducción y elementos básicos en Vue. Directivas.

1. Introducción

El uso de un framework nos facilita enormemente el trabajo a la hora de crear una aplicación. Vue es un framework progresivo para la construcción de interfaces de usuario y aplicaciones desde el lado del cliente. Lo de framework "progresivo" significa que su núcleo es pequeño pero está diseñado para crecer: su núcleo está enfocado sólo en la capa de visualización pero es fácil añadirle otras bibliotecas o proyectos existentes (algunos desarrollados por el mismo equipo de Vue) que nos permitan crear complejas SPA.

Su código es *opensource* y fue creado por el desarrollador independiente Evan You, lo que lo diferencia de los otros 2 frameworks más utilizados, *Angular* desarrollado por Google y *React* desarrollado por Facebook.

Vue tiene una curva de aprendizaje menor que otros frameworks y es extremadamente rápido y ligero.

Este material está basado en la guía oficial de Vue y veremos además los servicios de vue-router, axios y pinia (sustituto de vuex en Vue3) entre otros.

¿Qué framework es mejor?
Depende de la aplicación a desarrollar y de los gustos del programador. Tenéis algunos enlaces al respecto:

- Comparativa VueJs
- Comparativa Openwebminars
- Openwebminars: Vue vs Angular
- Carlos Azaustre: Vue vs Angular (vídeo)
- Angular vs React vs Vue: Which Framework to Choose in 2021
- ...

Las razones de que veamos Vue en vez de Angular o React son, en resumen:

- **Sencillez**: aunque Angular es el framework más demandado hoy en el mercado su curva de aprendizaje es muy pronunciada. Vue es mucho más sencillo de aprender pero su forma de trabajar es muy similar a Angular por lo que el paso desde Vue a Angular es relativamente sencillo
- **Uso del framework**: React es también muy sencillo ya que es simplemente Javascript en el que podemos codificar la vista con JSX, pero la forma de trabajar de Vue es más parecida a otros frameworks, especialmente a Angular por lo que lo aprendido nos será de gran ayuda si queremos pasar a ese framework
- **Rendimiento**: Vue hace uso del concepto de *Virtual DOM* igual que React por lo que también ofrece muy buen rendimiento

2. Usar Vue

Para utilizar Vue sólo necesitamos enlazarlo en nuestra página desde un CDN. Si queremos usar la nueva versión Vue 3 usaremos cualquier CDN como:

```
<script src="https://unpkg.com/vue@next"></script>

<script src="https://unpkg.com/vue@3.2.21/dist/vue.global.prod.js">
</script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/3.2.21/vue.cjs.js"
integrity="sha512-
2e2aXOh4/FgkCAUyurkjk0Uw4m1gPcExFwb1Ai4Ajjg97se/FEWfrLG1na4mq8cgOzouc8qLIq
sh0EGksPGdqQ==" crossorigin="anonymous" referrerpolicy="no-referrer">
</script>
```

Esta no es la forma más recomendable de trabajar por lo que más adelante usaremos la herramienta `vue-cli` para crear un completo *scaffolding* que nos facilitará enormemente la creación de nuestras aplicaciones (donde podremos incluir otras herramientas, trabajar con componentes o construir una SPA de forma sencilla).

Nosotros estamos usando *VSCode* como editor. Para que reconozca correctamente los ficheros *.vue* debemos instalar el *plugin* **Vetur**. Si vamos a usar Vue3 con Typescript podemos instalar el *plugin* **Volar**.

3. Estructura de una aplicación Vue

Vamos a crear la aplicación con Vue que mostrará un saludo. En el HTML necesitamos enlazar la librería de Vue y creamos un elemento (en nuestro caso un DIV) que contendrá la aplicación. En Vue 3:

```
<div id="app">
  {{ message }}
</div>

<footer>
  <p>
    <span class="copy-left">©</span>
    <span>DWEC - Vue3</span>
  </p>
</footer>

<script src="https://unpkg.com/vue@next"></script>
```

```
Vue.createApp({
  data() {
    return {
      message: 'Hello Vue.js!'
    }
  }
}).mount('#app');

```

3.1 HTML

En el HTML debemos vincular los scripts de la librería de Vue y de nuestro código.

Vue se ejecutará dentro de un elemento de nuestra página (al que se le suele poner como id *app*) que en este caso es un `<div>`.

Dentro de ese elemento es donde podemos usar expresiones de Vue (fuera del mismo se ignorarán). En este ejemplo se usa el *moustache* (**{{ ... }}**) que muestra en la página la variable o expresión Javascript que contiene.

3.2 Javascript

En el fichero JS debemos crear un nuevo objeto Vue que recibe como parámetro un objeto con varias propiedades.

En **Vue 3** la sintaxis para crear la aplicación es ligeramente diferente:

```
var app = Vue.createApp({
  data() {
    return {
      message: 'Hello Vue!'
    }
  }
})
app.mount('#app')
```

Las principales diferencias son:

- la instancia se crea con el método `createApp` en vez de con el constructor de Vue
- el elemento en que se montará la aplicación no se incluye como una propiedad del objeto que se pasa al crear la aplicación sino que se indica en el método `mount`

- la propiedad *data* no es un objeto sino una función que devuelve ese objeto (esto sucede igual en los componentes de Vue2 como veremos más adelante).

4. La instancia *Vue*

La instancia que hemos creado (al igual que cada componente) recibe un objeto de opciones con las siguientes propiedades:

- **data**: objeto (o función) donde definiremos todas las variables que vamos a usar en la vista. Las variables que sólo se usan en javascript las definiremos con **let** en el método donde vayamos a usarlas. Todas las variables definidas en *data* son accesibles desde la vista poniendo **{{ su_nombre }}** y desde el código JS poniendo `this.su_nombre`
- **computed**: son variables cuyo valor hay que calcularlo usando una función. Por ejemplo:

```
data: {
    nombre: 'David',
    apellido: 'Espert',
},
computed: {
    nombreCompleto() {
        return this.nombre + ' ' + this.apellido;
    }
}
```

- **methods**: objeto con métodos que también podemos llamar desde la vista
- ***hooks*** (eventos del ciclo de vida de la instancia): para ejecutar código en determinados momentos: **'created'**, **'mounted'**, **'updated'**, **'destroyed'**. Ej.:

```
created() {
    console.log('instancia creada');
}
```

5. *Binding* de variables

En la Guía de la documentación oficial de Vue tenemos un tutorial guiado donde podemos probar cada una de las funcionalidades de Vue. En la parte superior izquierda nos pregunta por nuestras preferencias: de momento escogeremos **Options** y **HTML**.

Para probar el funcionamiento de los ejemplos de este tutorial conviene que nos descarguemos los ficheros y los abramos en local.

Fichero HTML:

```
<!DOCTYPE html>
<html>
<head>
  <title>Hello world</title>
</head>
<body>

  <div id="app">
    <p>{ { message }}</p>
  </div>

  <script src="https://unpkg.com/vue"></script>
  <script src="01-HelloWorld.js"></script>
</body>
</html>
```

Nuestro código debemos cargarlo después de cargar la librería y de crear el elemento HTML que contenga la aplicación.

Fichero JS en Vue3:

```
var miApp = Vue.createApp({
  data() {
    return {
      message: 'Hello Vue.js!'
    }
  }
}).mount('#app');
```

5.1 Enlace unidireccional: interpolación {{...}}

Hemos creado una variable *miApp* que contiene nuestro objeto Vue y que podemos ver y manipular desde la consola. Si cambiamos el valor de la variable *message*

```
miApp.message = "Hola Vue!";
```

vemos que cambia lo que muestra nuestra página.

Esto es porque Vue (al igual que Angular o React) enlazan el DOM y los datos de forma que cualquier cambio en uno se refleja automáticamente en el otro.

5.2 Enlazar a un atributo: v-bind

Para mostrar un dato en el DOM usamos la interpolación **{{ }}** pero si queremos nostrarlo como atributo de una etiqueta debemos usar `v-bind`:

```
<p v-bind:title="message">
  Hover your mouse over me for a few seconds
  to see my dynamically bound title!
</p>
```

Vue incorpora estos '*atributos*' que podemos usar en las etiquetas HTML y que se llaman **directivas**. Todas las directivas comienzan por **`v-`** (en Angular es igual pero el prefijo es *ng-*). Como la directiva `v-bind` se utiliza mucho se puede abreviar símplemente como `:` (el carácter 'dos puntos'). El siguiente código es equivalente al de antes:

```
<p :title="message">
```

5.3 Enlace bidireccional: v-model

Tanto **{{ }}** como `v-bind` son un enlace unidireccional: muestran en el DOM el valor de un dato y reaccionan ante cualquier cambio en dicho valor.

Tenemos además está la directiva `v-model` que es un enlace bidireccional que enlaza un dato a un campo de formulario y permite cambiar el valor del campo al cambiar el dato pero también cambia el dato si se modifica lo introducido en el input.

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

Fichero HTML:

```
<div id="app">
  <p>El valor actual de la variable 'message' es <strong>{{ message }}
</strong></p>
  <label>Si cambias el valor del input se cambia la variable:</label>
  <input v-model="message">
</div>

<footer>
  <p>
    <span class="copy-left">©</span>
    <span>DWC - Vue JS</span>
  </p>
</footer>
<script src="https://unpkg.com/vue@next"></script>
```

Fichero JS en Vue3:

```
var miApp = Vue.createApp({
  data() {
    return {
      message: 'Hello Vue.js!'
    }
  }
}).mount('#app');
```

Vemos que al escribir en el *input* automáticamente cambia lo mostrado en el primer párrafo. Esta característica nos permite ahorrar innumerables líneas de código para hacer que el DOM refleje los cambios que se producen en los datos.

NOTA: toda la aplicación se monta en el elemento *app* por lo que las directivas o interpolaciones que pongamos fuera del mismo no se interpretarán.

6. Otras utilidades

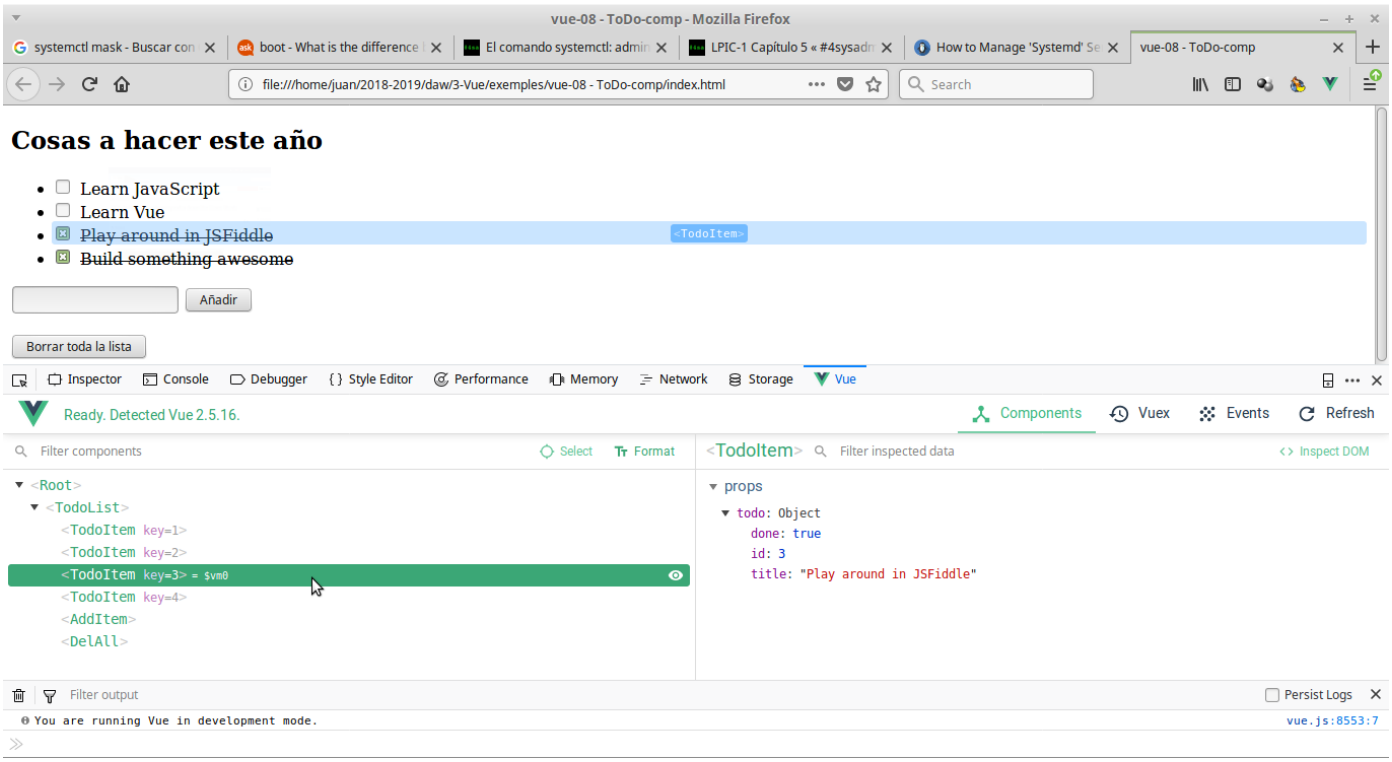
6.1 [Vue devtools]

Es una extensión para Chrome y Firefox que nos permite inspeccionar nuestro objeto Vue y acceder a todos los datos de nuestra aplicación. Es necesario instalarlo porque nos ayudará mucho a depurar nuestra aplicación, especialmente cuando comencemos a usar componentes.

Podemos buscar la extensión en nuestro navegador o acceder al enlace desde la documentación de Vue .

Si tenemos las DevTools instaladas en la herramienta de desarrollador aparece una nueva opción, *Vue*, con 4 botones:

- Componentes: es la vista por defecto y nos permite inspeccionar todos los componentes Vue creados (ahora tenemos sólo 1, el principal, pero más adelante haremos componentes hijos)
- Pinia/Vuex: es la herramienta de gestión de estado para aplicaciones medias/grandes
- Eventos: permite ver todos los eventos emitidos
- Refrescar: refresca la herramienta

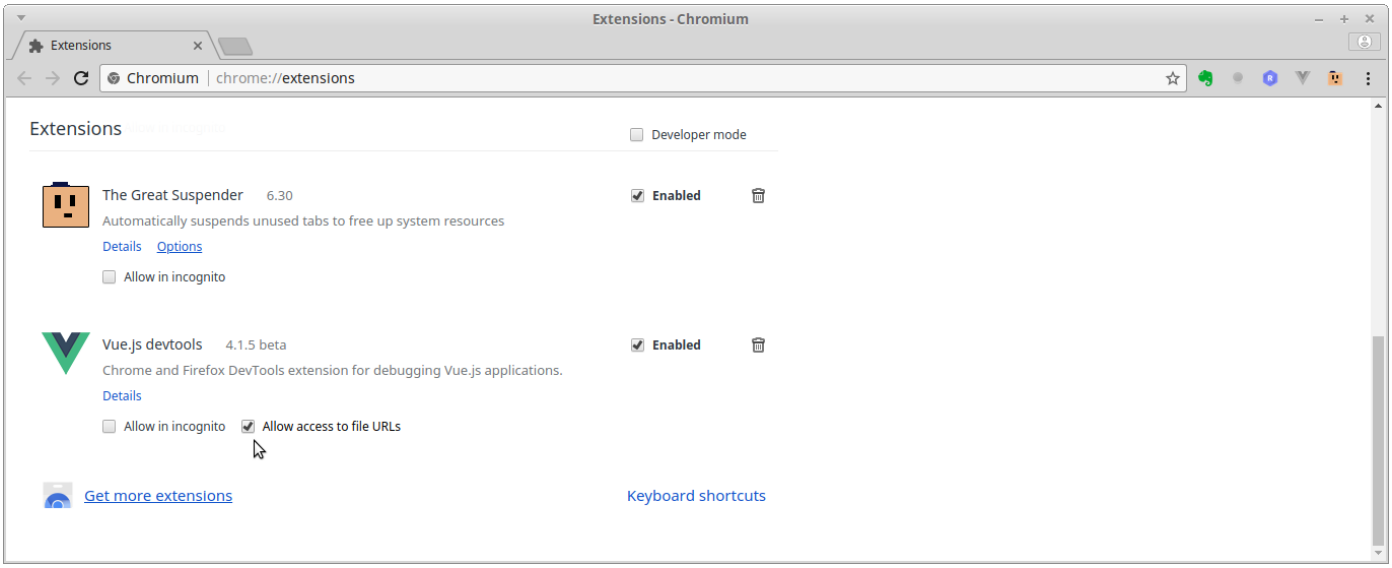


Junto al componente que estamos inspeccionando aparece **= $vm0** que indica que DevTools ha creado una variable con ese nombre que contiene el componente por si queremos inspeccionarlo desde la consola.

Cuando inspeccionamos nuestros componentes, bajo la barra de botones aparece otra barra con 3 herramientas:

- Buscar: permite buscar el componente con el nombre introducido aquí
- Seleccionar componente en la página: al pulsarlo (se dibuja un punto en su interior) hace que al pulsar sobre un componente en nuestra página se seleccione en la herramienta de inspeccionar componentes
- Formatear nombre de componentes: muestra los nombres de componentes en el modo *camelCase* o *kebab-case*

NOTA: Si por algún motivo queremos trabajar sin servidor web (desde file://...) hay que habilitar el acceso a ficheros en la extensión.



6.2 Extensiones para el editor de código

Cuando empecemos a trabajar con componentes usaremos ficheros con extensión **.vue** que integran el HTML, el JS y el CSS de cada componente. Para que nuestro editor los detecte correctamente es conveniente instalar la extensión para Vue.

En el caso de *Visual Studio Code* esta extensión se llama **Volar** (sustituye en *Vue 3* a la extensión **Vetur** que se usa con *Vue 2*). En *Sublime Text* tenemos el plugin **Vue Syntax Highlight**.

6.3 Otras utilidades

Vue 3 permite utilizar directamente *Typescript* en nuestros componentes simplemente indicándolo al definir el SFC (lo veremos al llegar allí).

Respecto a los *tests* se recomienda usar *Jest* para los test unitarios y *Cypress* para los E2E, como se indica en la documentación oficial.

6.4 Cursos de Vue

Podemos encontrar muchos cursos en internet, algunos de ellos gratuitos. Por ejemploe, los creadores de Vue tienen la web Vue Mastery donde podemos encontrar desde cursos de iniciación (gratuitos) hasta los más avanzados.

7. Directivas básicas

Las directivas son atributos especiales que se ponen en las etiquetas HTML y que les dan cierta funcionalidad. Todas comienzan por **v-**.

Las más comunes son:

- `v-text`: es equivalente a hacer una interpolación (**{{ ... }}**). Muestra el valor en la etiqueta
- `v-once`: igual pero una vez renderizado no cambia lo mostrado en la vista aunque cambie el valor de la variable
- `v-html`: permite que el texto que se muestra contenga caracteres HTML que interpretará el navegador (al usar la interpolación las etiquetas HTML son escapadas). Internamente hace un `.innerHTML` del elemento mientras que `v-text` (y `{\{...}}`) hacen un `.textContent`
- `v-bind`: para asignar el valor de una variable a un atributo de una etiqueta HTML (no entre la etiqueta y su cierre como hace la interpolación). Por ejemplo si tenemos la variable *estado* cuyo valor es *error* y queremos que un *span* tenga como clase ese valor haremos:

```
<span v-bind:class="estado">...
```

El resultado será: `<span class="error">`. La directiva *v-bind:* se puede abreviar simplemente como `:` (`<span :class="estado">`)

- `v-model`: permite enlazar un input a una variable (la hemos visto en el capítulo anterior). Tiene 3 modificadores útiles.
  - `.lazy`: em lugar de actualizar el valor al pulsar cada tecla (*onInput*) lo hace al perder el foco (*onChange*)
  - `.number`: convierte el contenido a Number
  - `.trim`: elimina los espacios al principio y al final del texto
- `v-if`: renderiza o no el elemento que la contiene en función de una condición
- `v-show`: similar al *v-if* pero siempre renderiza el elemento (está en el DOM) y lo que hace es mostrarlo u ocultarlo (`display: none`) en función de la condición. Es mejor si el elemento va a mostrarse y ocultarse a menudo porque no tiene que volver a renderizarlo cada vez
- `v-for`: repite el elemento HTML que contiene esta etiqueta para cada elemento de un array
- `v-on`: le pone al elemento HTML un escuchador de eventos (ej `<button v-on:click="pulsado">Pulsa</button>`. La directiva `v-on:` se puede abreviar como `@`, por ejemplo `<button @click="pulsado">Pulsa</button>`.

Lo que enlazamos en una directiva o una interpolación puede ser una variable o una expresión javascript. Ej.:

```
<p>{ { name }}</p>
<p>{ { 'Cómo estás ' + name }}</p>
<p>{ { name=='root'?'Hola Administrador':'Hola ' + name }}</p>
```

7.1 Condicionales: v-if

Esta directiva permite renderizar o no un elemento HTML en función de una variable o expresión.

El checkbox está enlazado a la variable *marcado* (a la que al inicio le hemos dado el valor true, por eso aparece marcado por defecto) y los párrafos se muestran o no en función del valor de dicha variable.

La directiva `v-else` es opcional (puede haber sólo un `v-if`) pero si la ponemos el elemento con el `v-else` debe ser el inmediatamente siguiente al del `v-if`.

```
const app = Vue.createApp({
  data() {
    return {
      marcado: true,
    }
  }
})

app.mount('#app');
```

```
<div id="app">
  <input type="checkbox" v-model="marcado"> Marca y desmarca
  <p v-if="marcado">
    El checkbox esstá marcado
  </p>
  <p v-else>
    El checkbox esstá desmarcado
  </p>
</div>
```

> **PRUEBA 01_UD9**: Prueba este código y entiende su funcionamiento de manera correcta.

También se pueden enlazar varios con `v-else-if`:

```
<div v-if="type === 'A'">
  A
</div>
<div v-else-if="type === 'B'">
  B
</div>
<div v-else-if="type === 'C'">
  C
</div>
<div v-else>
  Not A/B/C
</div>
```

7.2 Bucles: v-for

Esta directiva repite el elemento HTML en que se encuentra una vez por cada elemento del array al que se enlaza.

```
Vue.createApp({
  data() {
    return {
      todos: [
        {
          id: 1,
          title: 'Aprender JS',
          done: false,
         },
        {
          id: 2,
          title: 'Aprender jQuery',
          done: false,
         },
        {
          id: 3,
          title: 'Aprender Vue',
          done: false,
         },
      ]
    }
  }
}).mount('#app')
```

```
<div id="app">
  <p>
    Cosas a hacer este año:
  </p>
  <ul>
    <li v-for="elem in todos">
      {{ elem.title}}
    </li>
  </ul>
</div>
```

PRUEBA 02_UD9: Prueba este código de la directiva v-for y entiende su funcionamiento de manera correcta.

La directiva v-for recorre el array *todos* y para cada elemento del array crea una etiqueta <li> y carga dicho elemento en la variable *elem* a la que podemos acceder dentro del <li>.

Además del elemento nos puede devolver su índice en el array: `v-for="(elem,index) in todos" ...`.

Vue es más eficiente a la hora de renderizar si cada elemento que crea *v-for* tiene su propia clave, lo que se consigue con el atributo *key*. Podemos indicar como clave algún campo único del elemento o el índice:

```
<... v-for="(elem,index) in todos" :key="index" ...>
```

Pasar una *key* en cada *v-for* es recomendable ahora pero será obligatorio al usarlo en componentes así que conviene usarlo siempre.

También podemos usar `v-for` para que se ejecute sobre un rango (como el típico `for (i=0;i<10;i++)`):

```
<span v-for="n in 10" :key="n">{{ n }}</span>
```

NOTA: No se recomienda usar `v-for` y `v-if` sobre el mismo elemento. Si se hace siempre se ejecuta primero el `v-if`.

7.3 Eventos: v-on

Esta directiva captura un evento y ejecuta un método como respuesta al mismo.

```
Vue.createApp({
  data() {
```

```
    return {
      todos: [
        {
          id: 1,
          title: 'Aprender JS',
          done: false,
         },
        {
          id: 2,
          title: 'Aprender jQuery',
          done: false,
         },
        {
          id: 3,
          title: 'Aprender Vue',
          done: false,
         },
      ]
    }
  },
  methods: {
    delTodos() {
      if (confirm('¿Deseas borrar la lista de cosas a hacer?')) {
        this.todos=[];
      }
    }
  }
}).mount('#app')
```

```
<div id="app">
  <p>
    Cosas a hacer este año:
  </p>
  <ul>
    <li v-for="elem in todos">
      {{ elem.title}}
    </li>
  </ul>
  <button v-on:click="delTodos">
    Borrar lista
  </button>
</div>
```

> **PRUEBA 03_UD9**: Prueba este código para escuchar eventos y entiende su funcionamiento de manera correcta.

El evento que queremos capturar se pone tras el carácter `:` y se indica el método que se ejecutará.

Fijaos en el método *delTodos()* que para hacer referencia desde el objeto Vue a alguna variable o método se le antepone ***this.***

Se puede pasar un parámetro a la función escuchadora:

```
<button v-on:click="pulsado('prueba')">Pulsa</button>
```

```
  pulsado(valor) {
    alert(valor);
  }
```

Esta directiva se usa mucho así que se puede abreviar con `@`. El código equivalente sería:

```
<button @click="pulsado('prueba')">Pulsa</button>
```

7.3.1 Modificadores de eventos

A un evento gestionado por una directiva *v-on* podemos añadirle (separado por .) un modificador. Alguno de los más usados son:

- ***.prevent***: equivale a hacer un preventDefault()
- ***.stop***: como stopPropagation()
- ***.self***: sólo se lanza si el evento se produce en este elemento y no en alguno de sus hijos
- ***.once***: sólo se lanza la primera vez que se produce el evento (sería como hacer un *addEventListener* y tras ejecutarse la primera vez hacer un *removeEventListener*)

Ejemplo:

```
<form @submit.prevent="enviaForm">
```