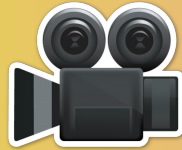




Clase 2. Vue JS

Proyecto CDN y primeras directivas

RECORDÁ PONER A GRABAR LA CLASE





OBJETIVOS DE LA CLASE

- Crear un proyecto Vue CDN
- Analizar y ejecutar las directivas data binding

CRONOGRAMA DEL CURSO

Clase 1



Inducción y configuración de herramientas



CONFIGURANDO GIT Y GITHUB



PROGRAMA CON ECMAScript EN NODE

Clase 2



Proyecto CDN y primeras directivas



PROYECTO CON HTML, CSS Y JS



USO DE DATA BINDING



CONTADOR

Clase 3



Directivas estructurales, condicionales y de atributo



BOTÓN PARA OCULTAR Y MOSTRAR



TABLAS DINÁMICAS

CREAR UN PROYECTO CON VUE CDN

PRIMER PROYECTO EN VUE

¡Vamos a construir nuestro primer proyecto en Vue! 🙌



PRIMER PROYECTO EN VUE

Para ello, utilizaremos el **CDN**, el cual debemos declarar dentro del apartado **<head>** de cada documento HTML que conforme nuestro proyecto. En la [guía oficial de Vue](#), encontrarás referencia al CDN que debes utilizar.

→ *Crea nuevo proyecto en VS Code, y agrega un documento HTML.*

PRIMER PROYECTO EN VUE

```
...
<!-- development version, includes helpful console warnings -->
<script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
</head>
```

La guía oficial detalla dos versiones de CDN: su **versión Development**, sin minificar y con advertencias a través de la Consola JS, y su **versión para Producción**, con `vue.js` 100% optimizado.

La versión Development será la que utilizaremos, para así identificar en la Consola JS los posibles errores al escribir la sintaxis de Vue.

VERSIONES DE VUE

Como puedes ver en la línea de código que referencia al CDN, Vue JS indica qué versión de framework estamos utilizando. En nuestro caso, corresponde a la **versión 2**: `.../npm/vue@2/di...`

Si por alguna razón necesitamos volver a una versión anterior, solo debemos cambiar el número de esta URL, y listo.

INSTANCIA DE VUE Y DATA

INSTANCIA DE VUE Y DATA

Creemos a continuación la aplicación más sencilla posible, conocida por todos como: **¡Hola Mundo!**



PRIMER PROYECTO EN VUE

- Doblearemos el código de nuestra app: referenciamos en el documento HTML un `<script>` llamado **app.js** (o el nombre que gustes). Allí guardaremos nuestro código JS.
- El código de nuestro documento HTML lo relacionaremos con **app.js** mediante el **id** de un tag HTML.

PRIMER PROYECTO EN VUE

Ejemplo
en vivo



```
<> index.html M x JS app.js U
<> index.html > html > body > div#app
1  <!DOCTYPE html>
2  <html lang="es">
3    <head>
4      <meta charset="UTF-8">
5      <title>Primer proyecto con Vue CDN</title>
6      <meta name="viewport" content="width=device-width">
7      <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
8      <script defer src="js/app.js"></script>
9    </head>
10   <body>
11     <div id="app">
12       |
13     </div>
14   </body>
15 </html>
```

1. En el apartado `<body>` definimos una etiqueta `div` con un `id`. Allí desplegaremos el código de Vue en el documento HTML.
2. Pulsemos **Ctrl + clic** sobre el script definido para crearlo.



CLASE VUE Y DATA

```
var app = new Vue({  
  el: '#app',  
  data: {  
  
  }  
})
```

👉 Aquí tenemos el código base de cualquier aplicación Vue. En él, instanciamos la clase **Vue()** que recibe como parámetro un objeto JSON el cual Vue puede interpretar.

La propiedad **el** refiere al **Elemento HTML** donde Vue renderiza. **data** almacena cualquier dato (*variables, constantes, arrays*) que declaramos, para utilizar luego en nuestra aplicación.

CLASE VUE Y DATA

Ejemplo
en vivo



```
JS app.js ×  
data: {  
  message: 'Hola Coder. Hola Mundo!'  
}
```

Agreguemos en el objeto **data**, una propiedad con un valor definido, tal como vemos aquí. 🖱️

En el **documento HTML** referenciamos **message**, para visualizar su contenido. Encerremos esta propiedad en llaves doble.

```
<> index.html M ×  
<div id="app">  
  {{ message }}  
</div>
```

A TESTEAR NUESTRO PROYECTO

¡Estamos en condiciones de probar nuestra primera App Vue!

Si bien lo realizado es muy básico, nos permite adquirir conocimientos reiterativos para toda aplicación que construyamos utilizando Vue CDN.

👉 Configuremos **Live Server** para probarlo.



INICIAR PROYECTO EN LIVE SERVER

Ejemplo
en vivo



```
index.html — HelloWorld
EXPLORER
  HELLO...
  js
    app.js
    index.html M
  index.html > html
    1 <!DOCTYPE html>
    2 <html lang="es">
    3   <head>
    4     <meta charset="UTF-8">
    5     <title>Primer proyecto con Vue
    6     <meta name="viewport" content=
    7     <script src="https://cdn.jsde
    8     <script defer src="js/app.js"
    9   </head>
   10   <body>
   11     <div id="app">
   12       {{ message }}
   13     </div>
   14   </body>
   15 </html>
```

→ Si no has instalado **Live Server** en VS Code, descárgalo desde su web oficial: [Live Server](https://live-server.net) | [VSCode Extension](https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer) (ritwickdey.github.io)

→ Para ejecutar tu proyecto, pulsa sobre el documento HTML con el botón secundario del mouse, y selecciona del menú **Open with Live Server**.

CODER HOUSE

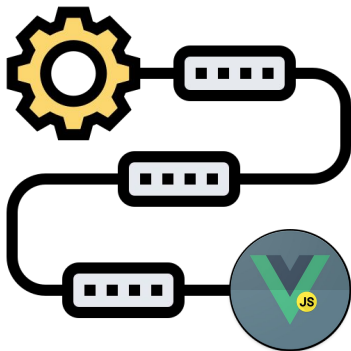
METHODS



METHODS

`methods{}` es un objeto asociado a la instancia de Vue.

Dentro de este definimos las funciones que necesitamos, las cuales interactuarán con nuestros elementos HTML, manejando diferentes eventos mediante las directivas **v-on**, para ejecutar acciones determinadas.





DEFINIR METHODS

```
var app = new Vue({  
  el: '#app',  
  data: {  
    message: ...  
  },  
  methods: {  
  
  }  
})
```

👉 Agregamos **methods** inmediatamente después del objeto **data{}**. Debemos separar uno del otro por una coma al igual que las funciones que declaremos luego, dentro de este.



AGREGAR MÉTODOS

```
methods: {  
  imprimirEnConsola() {  
    console.log('Hola, coders!')  
  },  
  calculoMatematico() {  
    let a = 21  
    let b = 3  
    console.log(a * b)  
  }  
}
```

Solo resta comenzar a agregar todos los métodos (o *funciones*) que consideremos necesarias para darle vida a nuestra aplicación Vue.

👉 Como mencionamos antes, deberás separar cada uno de ellos por una coma.



Métodos, Funciones... ¿ 😞 ?

Tal vez sea un confuso definir un objeto llamado `methods`, que contiene un array de funciones a las cuales también las llamamos métodos. 🙏 Este último nombre es el correcto para definir a dichas funciones, ya que las mismas están englobadas dentro del objeto **`Vue()`**, por lo tanto, **toda función creada dentro de un objeto pasa a ser un método.**

INTERACCIÓN ENTRE LOS MÉTODOS Y DATA



INTERACCIÓN MÉTODOS-DATA

Si, por ejemplo, tuviésemos las variables declaradas en `data` y debemos realizar una operación matemática con ellas, a través de un método, podemos invocarlas desde este último, anteponiendo la palabra `this`.

```
data: {  
  message: 'Hola Coders!',  
  a: 21,  
  b: 3  
},  
methods: {  
  calculoMatematico() {  
    console.log(this.a * this.b)  
  }  
...  
}
```




MÉTODOS Y COMPONENTES

Podemos también reflejar el resultado de un método en un elemento HTML. Para ello, modifiquemos el método en cuestión, usando `return` en lugar de `console.log()`.

Luego, agregamos un elemento HTML y, dentro de este, el nombre del método encerrado entre dos llaves (o *doble mustaches*).

```
...
methods: {
  calculoMatematico() {
    return this.a * this.b
  },
  ...
}
```

<> index.html M ✕

```
<div id="app">
  {{ message }}
  <p>Resultado: {{ calculoMatematico() }}</p>
  ...
</div>
```

PROPIEDADES COMPUTADAS

PROPIEDADES COMPUTADAS



Toda variable creada en **data** no puede asumir valores de otra variable creada en ese mismo espacio pero, por suerte, las **propiedades computadas** resuelven este problema.

Éstas **son propiedades asociadas a componentes**, a las que se le pueden aplicar cálculos o transformaciones.



PROPIEDADES COMPUTADAS

Siempre **deben retornar un valor**.

Trabajan de forma similar a los **getter** de otros lenguajes de programación y, para definirlas, lo hacemos a continuación de `data` o `methods`.

Si bien son funciones, cuando las utilizamos **siempre las escribimos como si fuesen una propiedad o una variable**.

PROPIEDADES COMPUTADAS

Ejemplo
en vivo



```
computed: {  
  duplicar() {  
    ...  
  },  
  raizCuadrada() {  
    ...  
  }  
}
```

Una vez definido el objeto `computed`, tal como hicimos en `methods`, creamos dentro de este todas las funciones que necesitemos, separadas por comas.

Tengamos presente que, las propiedades computadas, **no pueden recibir un parámetro por afuera de la función.**

De necesitar pasarle un valor, debemos crear un método para tal fin.

PROPIEDADES COMPUTADAS

Ejemplo
en vivo



Y así como recurrimos a `this` para referenciar una propiedad declarada en `data`, también debemos utilizar `this` si queremos acceder a un método.

```
...  
  
computed: {  
  duplicar() {  
    this.imprimirEnConsola()  
    return this.a * 2  
  }  
}
```

Ejemplo
en vivo



Propiedades computadas

Finalmente, declaramos nuestra propiedad computada en el documento HTML, tratándola como si fuese una variable.

Así obtendremos el resultado esperado. 💪

```
<div id="app">
  {{ message }}
  <p>Resultado: {{ calculoMatematico() }}</p>
  <p>Valor duplicado de {{ a }}: {{ duplicar }}</p>
</div>
```

Hola Coders!

Resultado: 63

Valor duplicado de 21: 42

CODER HOUSE



REACTIVIDAD Y CACHÉ

Cuando un **método** devuelve algo almacenado en **data**, éste sólo se ejecutará la primera vez y, si **data** cambia su valor, el método no reaccionará a dicho cambio.

El caché de las **propiedades computadas** hace que estas se ejecuten una sola vez, y cuando el valor de **data** cambie, ésta actualizará la información de la pantalla.



PROYECTO VUE CDN

Crea tu primer proyecto con Vue CDN.

Tiempo estimado: 15 minutos.

CODER HOUSE

SUBTÍTULO DEL DESAFÍO

Desafío
generico



Crea un proyecto que incluya HTML, estilízalo con CSS y por supuesto Vue CDN, integrando un archivo JavaScript donde escribirás tu código Vue.

Agrega en éste al menos 3 propiedades en el apartado **data**, y represéntalas luego en la Vista.

Tiempo estimado: 15 minutos.

CODER HOUSE

EXPRESIONES



EXPRESIONES Y RENDERIZADO

Al momento, aprendimos cómo Vue JS incorpora el valor de las variables en un documento HTML mediante el uso de **{{ doble mustaches }}**.

Este proceso es conocido como interpolación y es similar a las expresiones contenidas en marcadores, que se usan en JS **\${valor}**.

Además de esto, Vue cuenta con una serie de etiquetas y atributos de comportamiento específico que permiten renderizar cualquier otro objeto JS al formato de HTML nativo.



INTERPOLACIONES

Interpolación, es la posibilidad de convertir modelos de datos en cadenas de texto para que estos puedan mostrarse en pantalla.

Ejemplos de esto 🙌 son los que usamos hasta ahora.

```
<div id="app">
  {{ message }} <!-- propiedad almacenada en data -->
  <p>Resultado: {{ calculoMatematico() }}</p> <!-- Método declarado en Methods-->
  <p>Valor duplicado de 21: {{ duplicar }}</p> <!-- Una propiedad computada -->
</div>
```

INTERPOLAR EXPRESIONES



INTERPOLAR EXPRESIONES

Si bien no es lo más apropiado o recomendado recurrir a muchas de estas opciones, también es posible interpolar algunas expresiones JavaScript dentro del doble moustache. Vue sabrá cómo decodificarlas.

```
<p> {{ calculoMatematico + 1 }}</p>
```

```
<!-- método interpolado con una suma -->
```

```
<p>{{ ok ? 'Sí' : 'No' }}</p>
```

```
<!-- operador ternario JS -->
```

```
<p>{{ message.split('').reverse().join('') }}</p> <!-- Métodos de string JS -->
```

Ejemplo
en vivo



¡VAMOS A PRACTICAR LO VISTO!

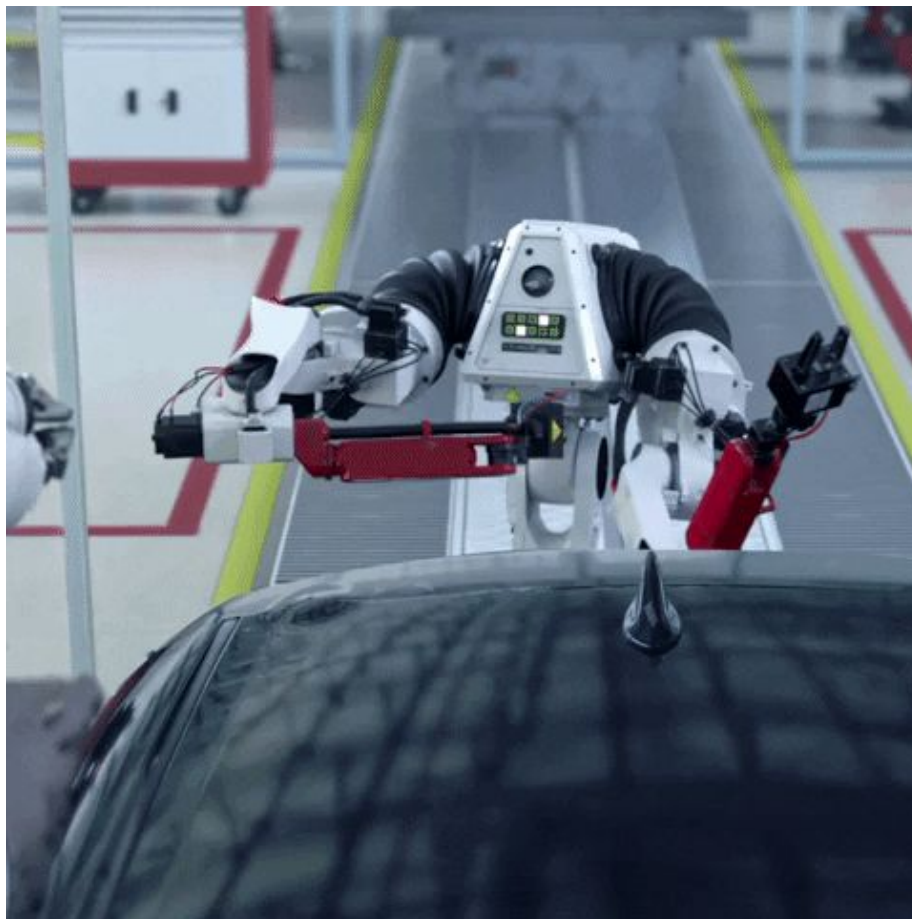
CODER HOUSE



BREAK

¡5/10 MINUTOS Y VOLVEMOS!

DIRECTIVAS



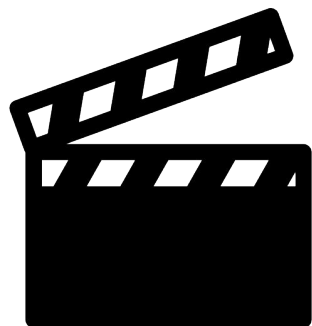
DIRECTIVAS



Las directivas son atributos que simplifican un montón de acciones que, en JavaScript puro, llevarían varias líneas de código.

Se identifican mediante el prefijo **v-**(seguido de la acción), y se agregan a un tag **HTML**, como un atributo más de éste.

DIRECTIVAS



Están pensadas para realizar acciones dinámicas que usualmente no se pueden realizar de forma normal, desde el ámbito de un documento HTML.

👁️ Veamos, a continuación, una infografía de cómo se compone una directiva.



- **Directiva:** el nombre de la directiva; a veces abreviable a un solo caracter.
- **Argumento:** ciertas directivas requieren que se indique un argumento.
- **Modificador:** en algunos casos es posible modificar su comportamiento.
- **Valor:** espera que definamos un valor, tal como lo hacemos con los atributos HTML

TIPOS DE DIRECTIVAS

TIPO DE DIRECTIVA	EJEMPLOS	FUNCIÓN
Directivas básicas	v-pre, v-once, v-model	Realizan tareas simples.
Directivas condicionales	v-show, v-if, v-else, v-elseif	Realizan acciones según condiciones.
Directivas de bucles	v-for	Iteración sobre arrays.
Directivas avanzadas	v-bind, v-on, v-slot	Realizan tareas muy específicas.
Directivas personalizadas		Directivas que podemos definir nosotros como desarrolladores.

V-TEXT, V-HTML, V-BIND, V-ON, V-MODEL



DIRECTIVA V-TEXT

```
<p> {{ nombreDeUsuario }} </p>  
  
<!-- alternativa 👉 a la línea de arriba -->  
  
<p v-text="nombreDeUsuario"></p>
```

A través de la directiva `v-text` podemos renderizar en pantalla: variables, métodos, propiedades, entre otras cosas. Es una alternativa al *doble moustache* que usamos anteriormente.



DIRECTIVA V-HTML

Si incluyéramos HTML dentro de una variable o propiedad en data, ésta se mostraría tal cual lo escribimos.

```
data: {  
  htmlCard: `

</div>`  
  ...  
}


```

Para resolverlo, Vue cuenta con la etiqueta `v-html`, la cual renderiza todo contenido HTML que incluya la propiedad o variable indicada.

```
<div v-html="htmlCard"></div>
```



DIRECTIVA V-BIND

La directiva **v-bind** permite enlazar una variable o propiedad, con un atributo específico de un tag HTML.

```
portada: "images/movies/jobs.jpg",  
descripcion: "Biopic de Steve Jobs basada en su libro biográfico."
```

Así, colocamos como valor de un atributo HTML el contenido que tengamos almacenado en una o más variables de JavaScript.

```
<div class="card black white-text">  
    
</div>
```



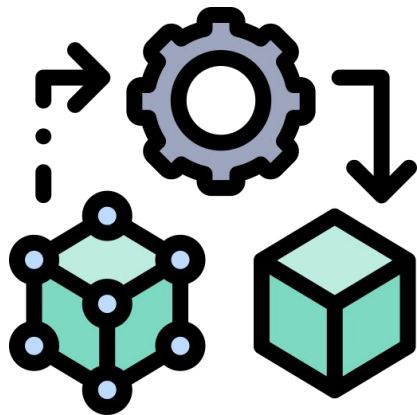
DIRECTIVA V-ON

v-on es utilizada para gestionar los eventos del **DOM** desde Vue, haciendo más cómodo y práctico su utilización.

Tenemos la misma cantidad de opciones de cada **on-event** de Vanilla JavaScript, o si manejas JS moderno, a los **addEventListener()** que usamos para detectar el evento de un elemento HTML puntual.

```
<button v-on:click="iniciarSesion">LOGIN</button>
```

DIRECTIVA V-MODEL



Esta directiva permite crear un Modelo de Datos bidireccional entre un elemento HTML y una variable de Vue, sincronizando el contenido de ambos.

¿Qué significa esto?



Veámoslo en un ejemplo...



DIRECTIVA V-MODEL

```
app.js — HelloWorld
index.html M  app.js M x  ...
js > JS app.js > [e] app > data
1  var app = new Vue({
2    el: '#app',
3    data: {
4      |
5      message: 'Hola Coders!',
6      a: 21,
7      b: 3,
8      usuario: '',
9      htmlCard: `<div class="card black
10     white-text></div>`,
12     popcorn: {
13       portada: "images/movies/jobs.
14       jpg",
15       descripcion: "Biopic de Steve
16       Jobs basada en su libro
```

Definimos una variable y la asociamos a un elemento HTML **<p>** y a un elemento HTML **<input>**. En este último, utilizando **v-model**.

Modificando el dato del elemento input, **v-model** aplica el modelo bidireccional, reflejando el cambio en la etiqueta HTML.

SIMPLIFICAR DIRECTIVAS

SIMPLIFICAR DIRECTIVAS

La integración del prefijo **v-** en las directivas de Vue tiene el objetivo de integrar una señal visual que permita identificar atributos específicos de Vue respecto al resto de atributos propios de los Elementos HTML.

Pero, a su vez, Vue propuso algunas abreviaturas para sus dos directivas más utilizadas: **v-bind** y **v-on**.

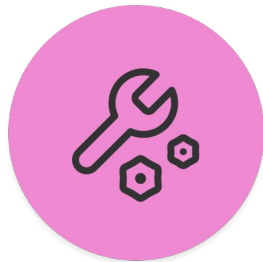
Veamos, a continuación, cómo aplicarlas.



SIMPLIFICAR DIRECTIVAS

```
<!-- directiva original -->  
<a v-bind:href="url">...</a>  
  
<!-- abreviatura -->  
<a :href="url">...</a>  
  
<!-- directiva original -->  
<a v-on:click="iniciarSesion">...</a>  
  
<!-- abreviatura -->  
<a @click="iniciarSesion">...</a>
```

En el bloque de código contiguo puedes apreciar una diferencia entre la directiva original o completa y, debajo, su equivalente abreviada.



DATA-BINDING

Implementa funcionalidades en HTML con data-binding.

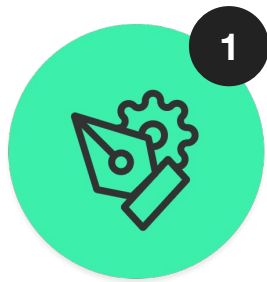


DATA-BINDING

Añadir al proyecto creado en el desafío anterior, un elemento de entrada, o input, cuyo valor ingresado se almacene en una variable de la instancia de Vue, utilizando doble data binding con **v-bind** y **v-on** (manual) y, luego, con **v-model** (automática).

Representar esa variable usando expresiones.

Tiempo estimado: 15 minutos.



CONTADOR

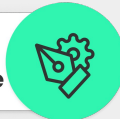
Implementa un contador con Vue JS.

CONTADOR

Formato: Utiliza el nombre “Nombre del Desafío + Apellido”.

Sugerencia: N/A.

Desafío
entregable



>> Consigna: Implementa con Vue CDN un contador ascendente-descendente.

El ejercicio deberá tener dos botones y un tag, o input (del tipo read-only), donde se refleje el valor numérico del contador.

Hasta el momento no integramos ningún Framework CSS, así que te invitamos a integrar Bootstrap de forma convencional y maquetar el proyecto con estilos acordes a la propuesta sugerida.

Sube el proyecto a través de tu perfil de la plataforma Coderhouse.

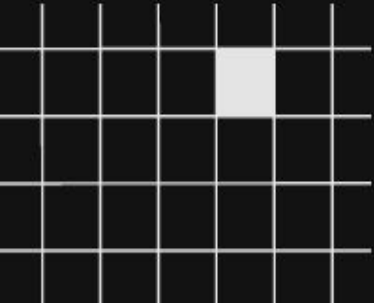
¿PREGUNTAS?





¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Integrar Vue CDN en un proyecto web
 - Expresiones
 - Interpolaciones
 - Directivas
 - Directivas simplificadas
 - Data Binding
- 



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN

CODER HOUSE