



Diseño y Programación Web

Materia:

Aplicaciones para
Dispositivos Móviles

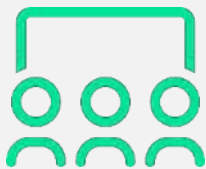
Docente contenidista: GARCIA, Mabel

Revisión: Coordinación

Contenido

¿En qué consiste la materia?	4
¿Qué es Vue?	5
Características principales de Vue	6
Framework Progresivo	6
Fácil de implementar	7
Liviano	7
Sistema reactivo	7
DOM Virtual	7
Representación Declarativa	8
Componentes	8
Adaptabilidad	8
Patron MVVM (Modelo Vista - Vista Modelo)	8
¿Quién usa Vue?	9
Herramienta para debug	11
Incorporar Vue a un documento HTML	16
Instancia de Vue	17
Interpolación	20
Directivas Básicas	22
v-cloak	22
v-once	23
v-show	24
v-html	27
v-bind	28
Manipular clases y estilos	30
Bibliografía	35

Clase 1



¡Te damos la bienvenida a la materia
Aplicaciones para Dispositivos Móviles!

En esta clase vamos a ver los siguientes temas:

- ¿Qué es Vue?
- Características de Vue.
- ¿Quién usa Vue?
- Herramienta para debug.
- Incorporar Vue a un documento html.
- Instancia de Vue.
- Interpolación.
- Directivas básicas (v-cloak, v-once, v-html, v-bind).
- Manipular clases y estilos.

¿En qué consiste la materia?

La materia se enfoca en el uso de Vue, un framework javascript que nos permite crear aplicaciones/sitios web interactivos y dinámicos.

A lo largo de la cursada, aprenderemos los fundamentos de Vue, cómo son los componentes, las directivas, los eventos y las transiciones, etc.

También veremos cómo integrar Vue con otras herramientas.

Al finalizar la materia, seremos capaces de desarrollar sitios web con Vue que se comuniquen con una base de datos y ofrezcan una experiencia de usuario fluida y atractiva, con un enfoque offline.

Introducción a Vue

¿Qué es Vue?

Vue.js es una biblioteca de JavaScript para crear interfaces web interactivas y reactivas. Creada en 2014 por Evan You, ex-ingeniero de Google, Vue.js se inspira en frameworks como Angular y React, pero introduce sus propias características y convenciones.

Fue diseñado para ser simple, flexible y fácil de aprender, con una API intuitiva y una curva de aprendizaje baja. Además, Vue.js se enfoca en la capa de vista, lo que te da mayor libertad y compatibilidad con otras herramientas y bibliotecas.

Hay que reconocer que ganó popularidad rápidamente gracias a su documentación clara, su comunidad activa y su ecosistema de complementos y herramientas. También recibió elogios de desarrolladores influyentes y empresas como Laravel, Alibaba y GitLab.

A diferencia de otros frameworks que surgieron de grandes empresas, como Angularjs creado por Google en 2010 y usado en sus productos como Gmail, Google Cloud, etc., o React creado por Facebook en 2013 y empleado en Instagram, Facebook, etc., Vue cuenta con el apoyo de empresas patrocinadoras que respaldan el proyecto.

Vue.js se ha convertido en uno de los marcos de JavaScript más utilizados y queridos, con más de 200 mil de estrellas en GitHub y más de 1,6 millones de sitios web que lo utilizan. También ha evolucionado para ofrecer soluciones más avanzadas, como el enrutador Vue, Vue CLI, entre otras.

Características principales de Vue

Dentro de todas las características que posee Vue, mencionaremos sólo algunas para ir comprendiendo lo básico y entender este framework, ya iremos profundizando a medida que sigamos viendo y probando sus características. Cabe destacar por ejemplo:

- Es un **framework progresivo**.
- Posee **fácil implementación**.
- Es **liviano**.
- Trabaja como un **sistema reactivo**.
- Se enfoca en el **virtual DOM**.
- Usa una **representación declarativa**.
- Implementa **componentes**.
- Es **adaptable**.
- Usa el **patrón MVVM**.

Framework Progresivo

Hay que comprender que cuando hablamos de **framework**, nos estamos refiriendo a una herramienta que ayuda a los desarrolladores a crear rutinas de programación de forma rápida y sencilla, para que el programador no tenga que escribir todo el código desde cero y pueda implementar o modificar comportamientos, partiendo de cierta estructura o procedimientos que hagan que el código resultante sea más ordenado y consistente.

Cuando decimos que es **progresivo**, significa que se puede usar de forma gradual, empezando por lo más básico e ir añadiendo funcionalidades más complejas según las necesidades del proyecto. Lo que nos permite que un proyecto sea escalable a futuro.

Además es de código abierto, por lo tanto, no requiere pago de licencias para su libre uso.

Fácil de implementar

Según la documentación oficial de Vue, **podemos comenzar a usarlo teniendo conocimientos en HTML, CSS y Javascript.**

En primera instancia, esto nos permite un primer contacto con el framework de forma más amena, sólo necesitamos agregar la etiqueta **<script>** en nuestro HTML y comenzar a profundizar en sus funcionalidades.

Liviano

El **archivo core que posee todas las funcionalidades básicas de Vue3** sólo **pesa 521kb**, si deseamos tenerlo dentro de nuestros recursos físicos cuando estamos en modo desarrollo.

Sistema reactivo

Utiliza un sistema de reactividad basado en observadores, eventos y permite actualizar la vista automáticamente cuando cambian los datos y viceversa. Esto hace que la interfaz sea más dinámica y fluida.

DOM Virtual

El DOM virtual es una **representación en memoria de los elementos de la página**. Esto permite hacer cambios en el DOM mejorando el rendimiento y la velocidad de la aplicación.

Representación Declarativa

Utiliza una sintaxis declarativa, **extiende el HTML estándar** con una sintaxis de plantilla que nos permite describir declarativamente la salida HTML según el estado de JavaScript.

Componentes

Permite **crear componentes reutilizables y personalizados que facilitan la organización y el mantenimiento del código**. Los componentes son bloques de código que contienen la lógica, los datos y el diseño de una parte de la interfaz.

Adaptabilidad

Se puede **añadir a aplicaciones o sitios web ya existentes** y aprovecharlo sin mayores problemas de instalación.

Patron MVVM (Modelo Vista - Vista Modelo)

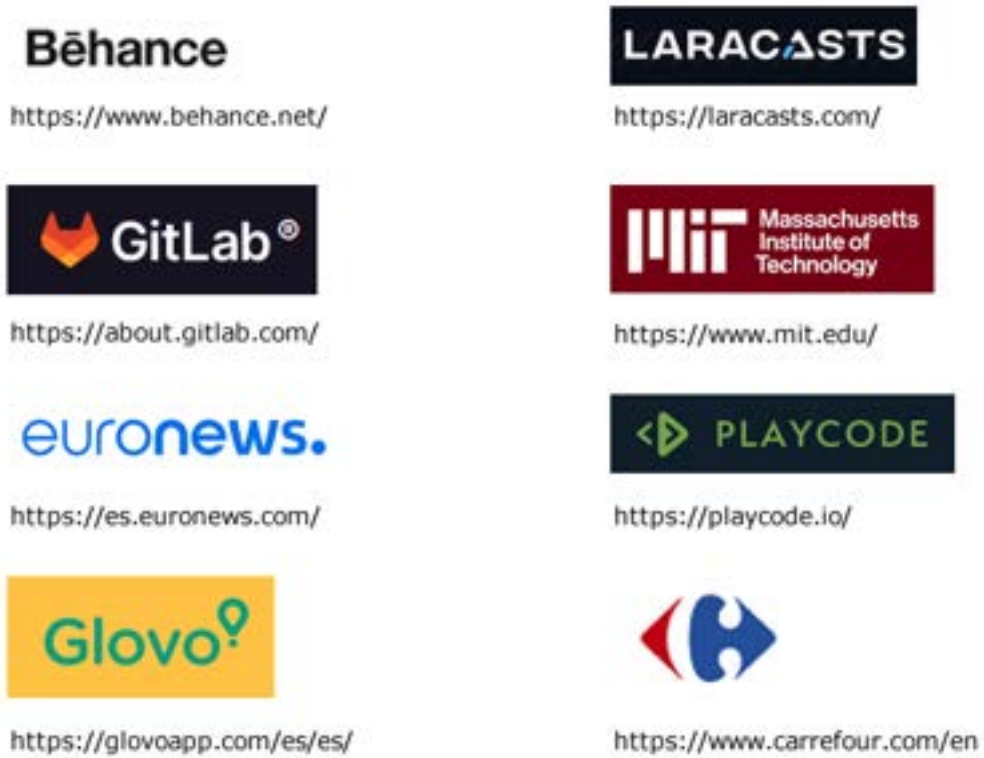
Es un patrón de diseño o modelo de abstracción usado para definir y **estructurar los componentes necesarios en el desarrollo de software**.

El **modelo representa la capa de datos y/o lógica** de negocio, contiene información.

La **vista debe representar la información** a través de los elementos visuales que la componen (HTML).

¿Quién usa Vue?

Actualmente muchos sitios webs usan en sus desarrollos este framework, por ejemplo:



<https://www.behance.net/> :

Es una plataforma de redes sociales, propiedad de Adobe desde el año 2020 que se enfoca en mostrar y descubrir trabajo creativo. Ofrece herramientas y tiene una gran comunidad.

<https://laracasts.com/> :

Es una plataforma de cursos online sobre programación y tecnologías relacionadas, especialmente PHP y Laravel.

<https://about.gitlab.com/>:

Es una herramienta para gestionar proyectos de software de forma colaborativa, usando el sistema de control de versiones Git.

<https://www.mit.edu/>:

El Instituto Tecnológico de Massachusetts, famosa universidad privada de Estados Unidos.

<https://es.euronews.com/> :

Es un canal de televisión que posee su portal de noticias.

<https://playcode.io/> :

Es un editor de código web que permite escribir y ejecutar código con vista previa en vivo.

<https://glovoapp.com/es/es/> :

Es una plataforma de entrega a domicilio que ofrece múltiples servicios a sus usuarios.

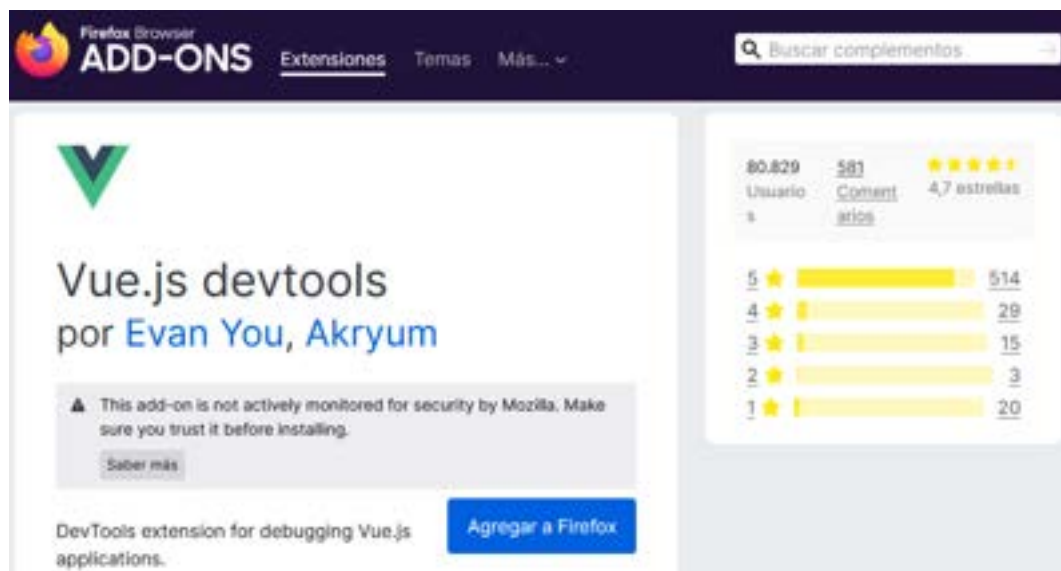
<https://www.carrefour.com/en> :

Carrefour es una empresa francesa de supermercados que opera en más de 30 países, incluyendo Argentina.

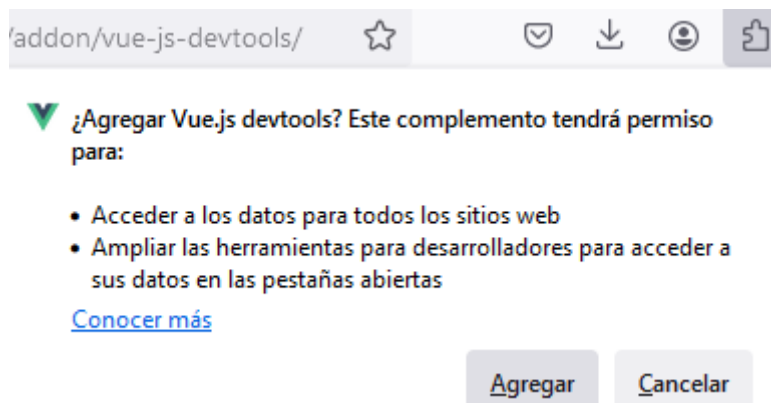
Herramienta para debug

La herramienta que usaremos se llama **Vue.js devtools**. Es una **extensión para los navegadores Chrome y Firefox** que nos permitirá depurar, analizar e inspeccionar el código de nuestras aplicaciones.

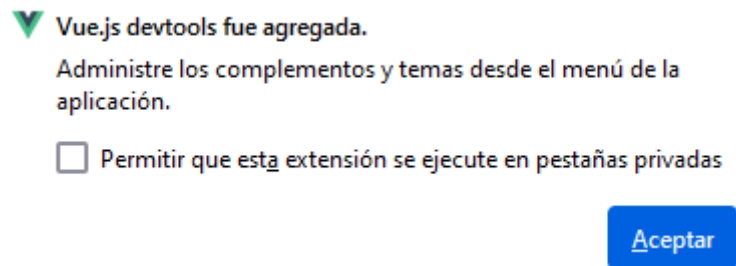
Si queremos instalarla en Firefox, abriremos el navegador de Firefox e iremos a la página web de los add-ons (<https://addons.mozilla.org/es/firefox/addon/vue-js-devtools/>) aquí nos encontraremos con la siguiente pantalla:



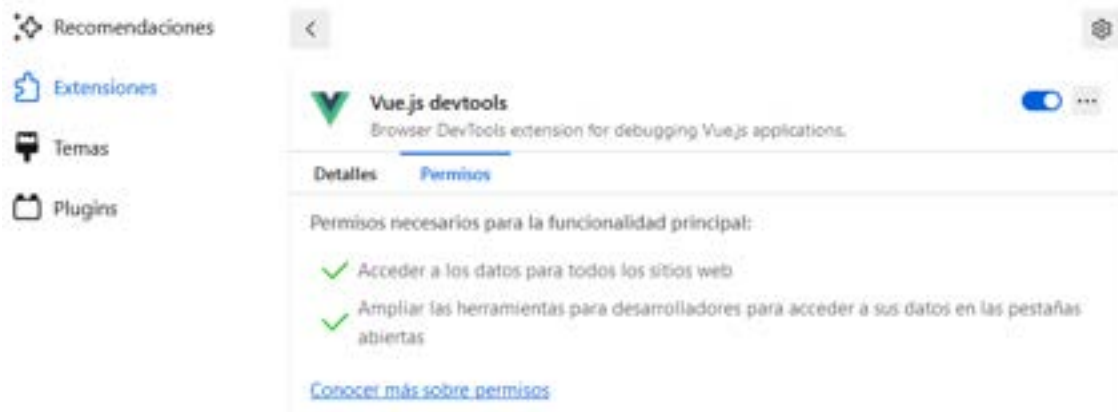
Le haremos click al botón azul **"Agregar a Firefox"** y se nos mostrará en el icono de las extensiones de la barra de navegación una confirmación para que sepamos el alcance de la extensión. Daremos click a **"Agregar"**.



Una vez agregada la extensión, nos avisará que administraremos los complementos desde el menú de la aplicación. Daremos click en **"Aceptar"**.



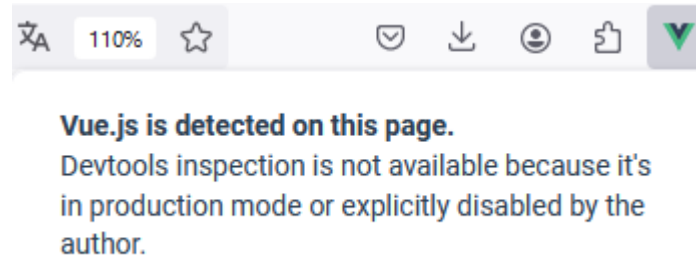
Nos dirigimos al icono de las **extensiones** y en **Administrar extensiones** revisaremos que tenga los permisos necesarios.



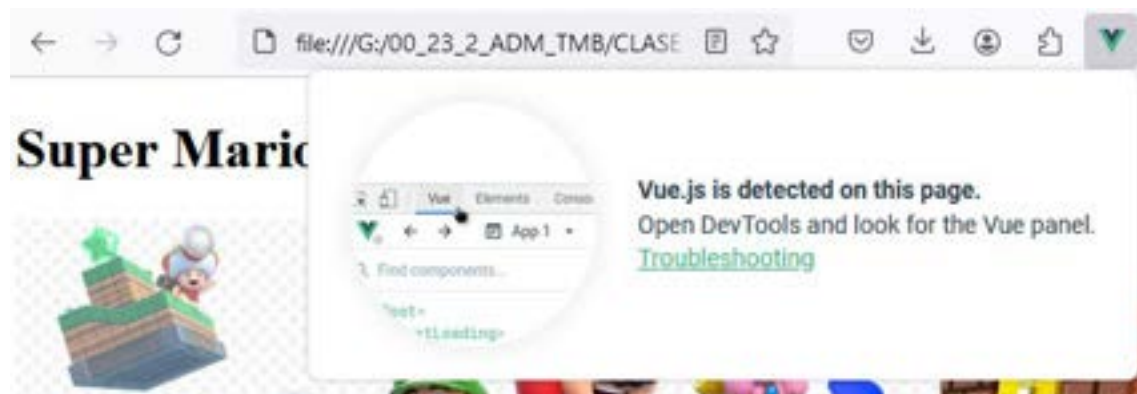
Cuando visitemos la web de un sitio que posea Vue, veremos el logo tal cual lo estamos viendo en la imagen anterior (color verde). Por ejemplo, si visitamos la página del MIT veremos lo siguiente:



Si hacemos click en la extensión de Vue, nos aparecerá un mensaje indicando que el sitio está usando Vue, pero se encuentra deshabilitada la inspección desde la consola, esto es lo normal, ya que nadie debería poder examinar el código implementado.



Cuando trabajemos con Vue en los desafíos de la semana, prácticas o con los ejemplos de la cursada, estaremos en modo desarrollo y deberemos controlar aspectos de nuestra aplicación, gracias al debug que haremos por medio de esta herramienta.



En este caso, estamos ejecutando un archivo HTML con Vue.js en modo local. Al activar la extensión, ésta detecta Vue.js en la página y nos informa al respecto. Para acceder al panel de Vue.js, debemos abrir la consola del navegador.

Dentro del panel, encontramos diversas herramientas que exploraremos a medida que avancemos en los conceptos básicos del framework.

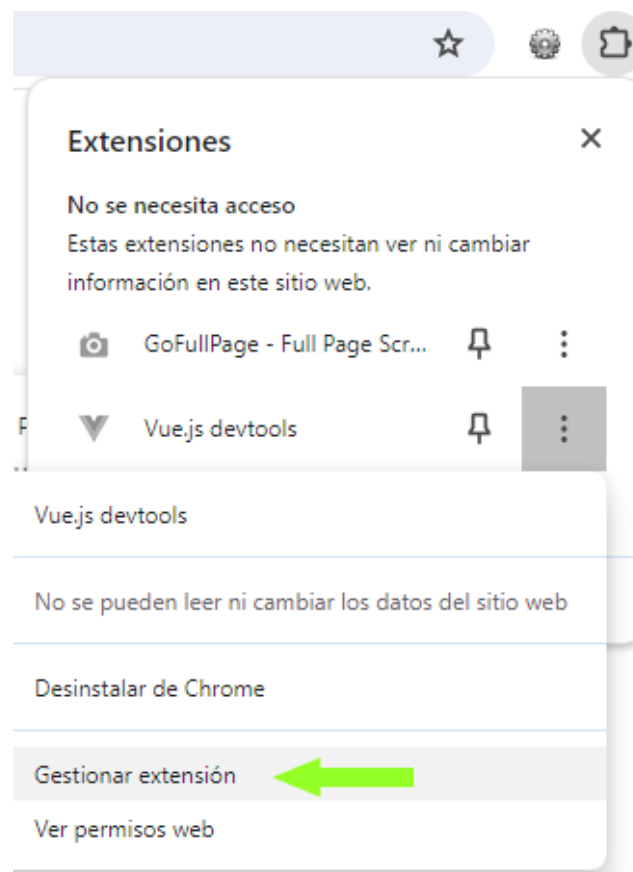
Tener en cuenta que si queremos instalar la extensión en Chrome, debemos seguir los mismos pasos:

1. Abrir Google Chrome.
2. ir al store https://chromewebstore.google.com/detail/vuejs-devtools/nhdogjmejiglipccpnnnanhbledajbpd?hl=es&utm_source=ext_sidebar

3. Añadir a Chrome:

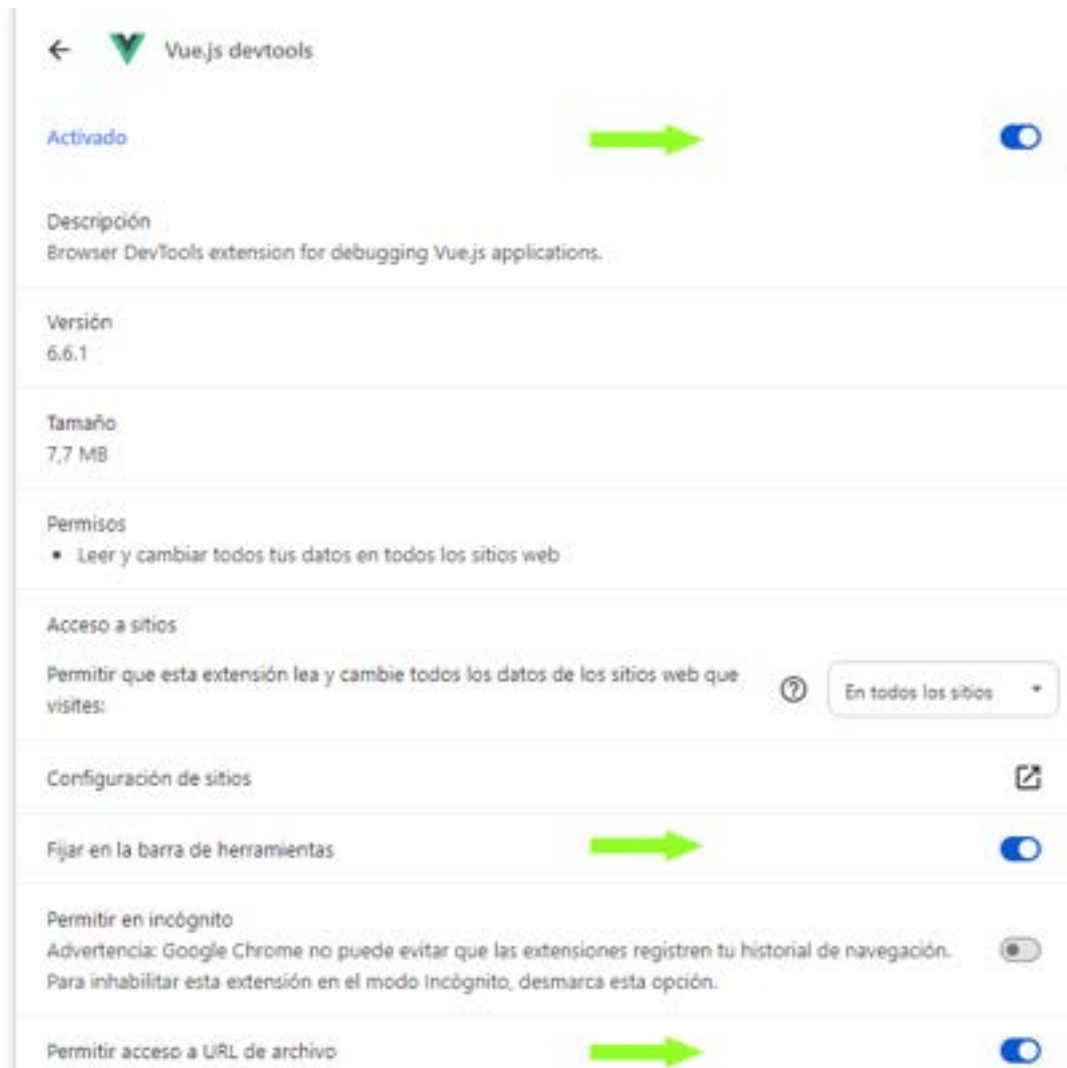


4. Añadir extensión.
5. Ir al ícono de extensiones.



6. Hacer click en **Gestionar extensión**.

7. Verificar que estos 3 ítems estén encendidos en la configuración:



Incorporar Vue a un documento HTML

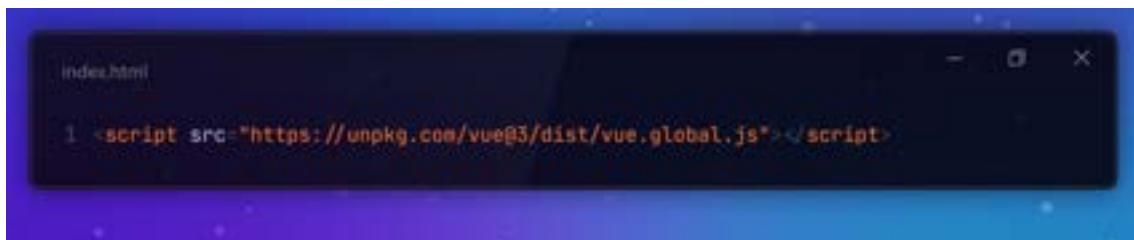
En nuestro primer acercamiento a Vue, vamos a trabajar por medio de un archivo javascript que contiene el core con sus funcionalidades.

Si vamos a la documentación veremos que hay 2 versiones. La versión 2.x y la versión 3. La versión 2.x ya no obtendrá más actualizaciones y se nos recomienda que usemos la versión 3.

Si bien muchas funcionalidades de la versión 2 se mantienen, iremos viendo a medida que avancemos en la cursada las diferencias que podemos encontrar.

Nos enfocaremos en la **Api Options** que posee esta versión.

En nuestro **archivo HTML vamos a usar vue directamente por CDN** a través de una etiqueta **<script>**.



Si queremos tener este **archivo de forma local**, iremos al navegador, pegaremos la url del **src** de la etiqueta **script** y seleccionaremos el código de este archivo para poder copiarlo.

Una vez copiado, abriremos nuestro editor de código y crearemos un **archivo de tipo js, por ejemplo vue3.js** y **pegaremos el código**. Finalmente vinculamos este recurso local a nuestro HTML.

Lo ideal es que este archivo se encuentre dentro de una carpeta por cuestiones organizativas.

De esta forma, no dependemos de nuestra red para que nuestro archivo pueda funcionar con Vue y no tendremos errores si nuestra red posee algún comportamiento errático.



```
index.html
1 <script src="scripts/vue3.js"></script>
```

Ahora necesitamos **crear otro archivo .js, en donde empezaremos a desarrollar sintaxis de Vue para poder manipular las bondades que nos brinda este framework.**

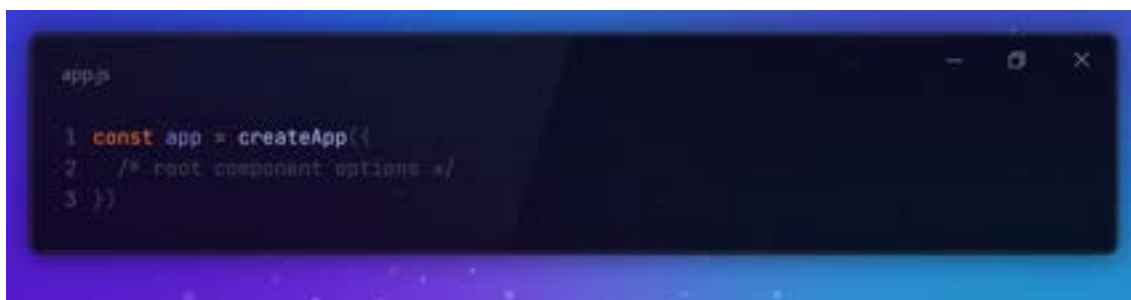
Instancia de Vue

Como mencionamos anteriormente, Vue trabaja con el patrón MVVM.

Por lo tanto tendremos un archivo js con la lógica y datos que Vue manipulará (Modelo) y por otro lado necesitamos de la vista (nuestro HTML) en donde se renderizan estos datos y actuarán los métodos declarados en la lógica).

Habitualmente se suele crear un archivo llamado **app.js**, que poseerá todo lo concerniente al Modelo.

Teniendo en claro esto, debemos tener presente que toda aplicación que use Vue precisa **la creación de una instancia principal, raíz o también llamada instancia root** y esto se logra con la función **createApp**.



```
app.js
1 const app = createApp({
2   /* root component options */
3 })
```

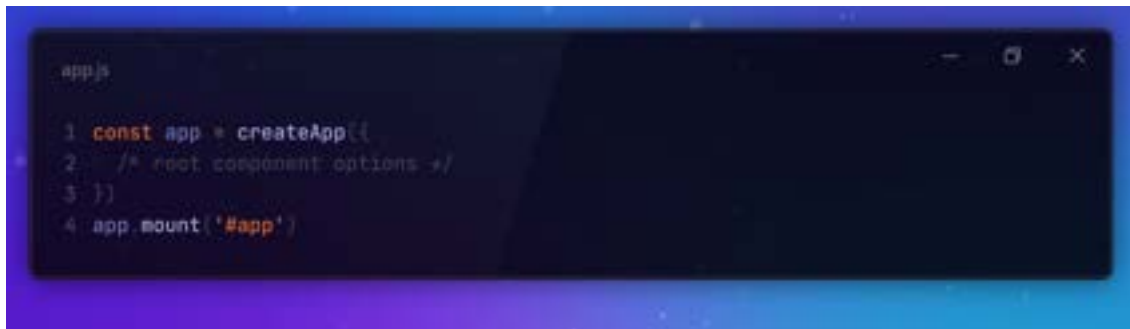
La función **createApp** retorna, de hecho, un componente que se almacena en la constante **app**. Cada aplicación requiere un "componente raíz", que pueda contener otros componentes como sus hijos.

Más adelante profundizaremos en estos conceptos.

El objeto que espera **la instancia root de Vue, definirá las opciones** de configuración (variables, métodos, etc).

Luego tendremos que establecer **en donde va a trabajar Vue**, y para ello, necesitamos usar un **método llamado .mount()**.


El argumento que espera que le pasemos es un **string con sintaxis CSS que apunte a su respectivo id, class o selector**. Este elemento, generalmente llamado **"#app"** debe existir en nuestro HTML, aunque puede tener otro nombre.

A screenshot of a code editor with a dark background and blue and purple accents. The code is written in JavaScript and shows the creation of a Vue.js application. It starts with 'app.js' at the top left. The code consists of four lines: 1. 'const app = createApp({', 2. ' /* root component options */', 3. '})', and 4. 'app.mount('#app')'. The code is syntax-highlighted with colors like orange for keywords and blue for comments.

La primera opción que veremos dentro del objeto **createApp** será la **función data**. Esta opción es un lugar donde Vue guarda todas las variables locales a la instancia/componente. Se define como una **function** que devuelve, retorna siempre un **Object**.

Dicho objeto tiene, en su interior, una **colección de propiedades que son las variables de Vue que usaremos en nuestra instancia**.

Por ejemplo, vamos a **inicializar una variable dentro de la instancia principal**, para que luego la podamos mostrar en el **HTML**:



```
app.js
1 const app = Vue.createApp({
2   data() {
3     return {
4       mensaje: 'Aprendiendo Vue'
5     }
6   }
7 });
```

En nuestro **index** tendremos un contenedor (div, section, etc) que tendrá el **id="app"**. Desde donde empieza esta etiqueta **<div>** hasta su cierre, será el ámbito de acción para Vue y resolverá sus procedimientos y sintaxis.

Cualquier código de Vue que implementemos debe estar dentro de este contenedor, caso contrario, Vue no ejecutará nuestro código.



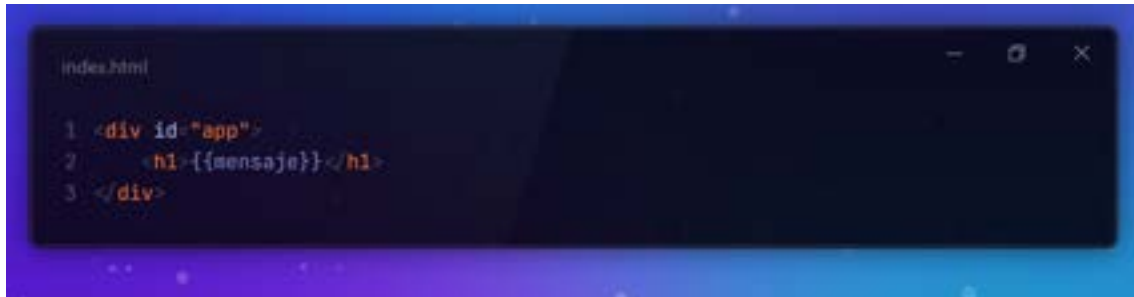
```
index.html
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8" />
5   <title>Introducción a Vue</title>
6 </head>
7
8 <body>
9   <div id="app">
10    +!--código html que se verá afectado por vue-->
11
12   </div>
```

Interpolación

Para mostrar el contenido de esta variable **"mensaje"** vamos a usar una **interpolación, también llamada "enlace de datos"**.

Se escribe con **llaves dobles**, también denominado **"mustache"**.

A medida que vayamos avanzando iremos profundizando en estos nuevos conceptos.

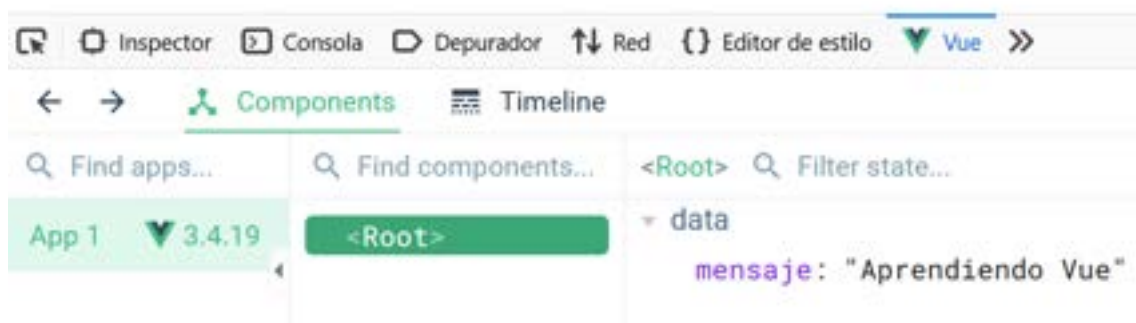


```
index.html
1 <div id="app">
2   <h1-{{mensaje}}</h1>
3 </div>
```

En el **navegador** veremos el **contenido** de esta **variable**, que Vue se encargó de procesar.

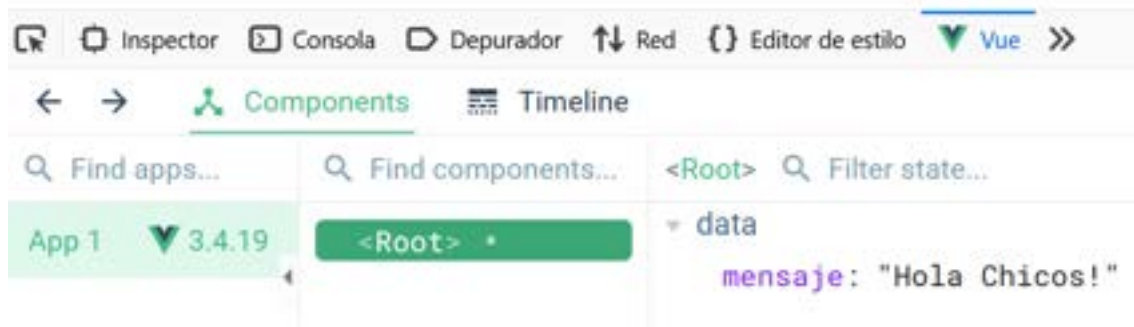
Si abrimos la consola, podremos ver al hacer click en el icono de Vue cómo esta información se ve inicializada en la instancia root y veremos si actualizamos su contenido, cómo la vista (HTML) reacciona al cambio, debido a que ambos elementos están vinculados, es decir, son reactivos.

Aprendiendo Vue



Le haremos click al contenido de la propiedad "mensaje" para poder editarlo. Ahora el **valor de la propiedad "mensaje" es "Hola Chicos"** y en el navegador vemos reflejado esa actualización.

Hola Chicos!



Directivas Básicas

Las directivas son una característica de Vue que otorga funcionalidades reactivas al DOM representado sobre las etiquetas en las cuales se aplican.

Son atributos especiales para un elemento con prefijo “v-”, seguido de un nombre que indica su propósito.

Podemos tener varias directivas aplicadas a una etiqueta del HTML y cada directiva otorgará determinados comportamientos a ejecutar cuando Vue la procese.

Para ejemplificar, vamos a empezar por directivas sencillas para comprender qué comportamientos podemos encontrar en ellas.

v-cloak

Esta directiva sirve para ocultar el contenido no compilado hasta que la instancia del root esté montada. De esta manera, se evita que el usuario vea el parpadeo del contenido con llaves de las interpolaciones durante la carga de la página. [Para ocultar el contenido no compilado, se marca el elemento con v-cloak.](#)

A screenshot of a code editor window titled 'index.html'. The code is as follows:

```
1 <div id="app" v-cloak>
2   <h1-{{mensaje}}</h1>
3 </div>
```

Y en nuestra hoja de estilos CSS, la [regla para ocultar este contenido hasta que la compilación termine.](#)

A screenshot of a code editor window titled 'estilos.css'. The code is as follows:

```
1 [v-cloak]{
2   display: none;
3 }
```

v-once

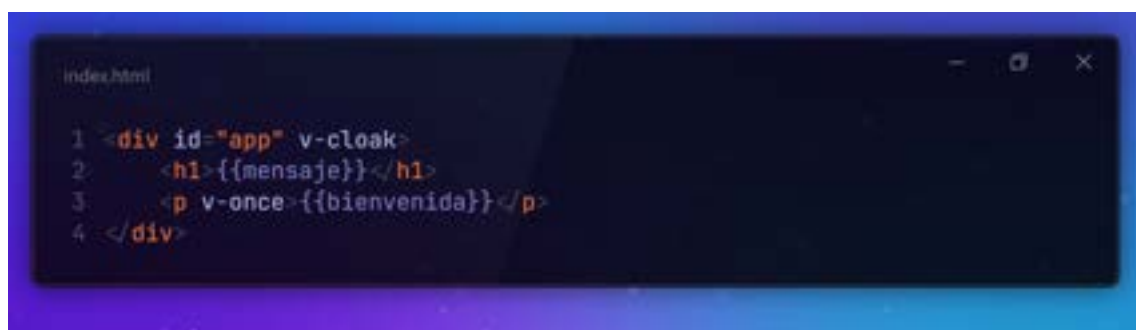
Esta directiva sirve para renderizar el contenido de un elemento sólo una vez y evitar que se vuelva a renderizar en las actualizaciones posteriores.

De esta manera, se puede mejorar el rendimiento de la aplicación al reducir el número de operaciones de renderizado necesarias.

Por ejemplo, si tuviéramos la propiedad **"bienvenida"** inicializada dentro de nuestra instancia root y su valor fuera **"Primera clase de Vue"**, al querer modificar el contenido de la propiedad **"bienvenida"** desde el panel de vue devtools, no se reflejará este cambio en la vista, por estar usando en nuestro HTML la directiva **v-once** en el párrafo que contiene la interpolación.



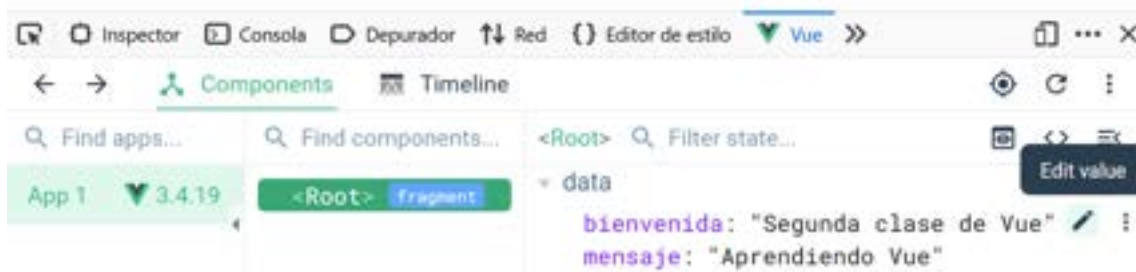
```
1 const app = Vue.createApp({
2   data() {
3     return {
4       mensaje: "Aprendiendo Vue",
5       bienvenida: "Primera clase de Vue"
6     }
7   },
8 });
9 app.mount('#app');
```



```
1 <div id="app" v-cloak>
2   <h1-{{mensaje}}</h1>
3   <p v-once-{{bienvenida}}</p>
4 </div>
```

Aprendiendo Vue

Primer clase de Vue



v-show

Esta directiva permite mostrar de modo condicional un elemento. Si el valor de la variable es **true**, mostrará el elemento, si es **false** le aplicará la propiedad CSS **display:none** y lo ocultará.

Por ejemplo, vamos a agregar otra propiedad a nuestra instancia, en este caso le pondremos de nombre **"visible"** y su valor inicial en **true**.



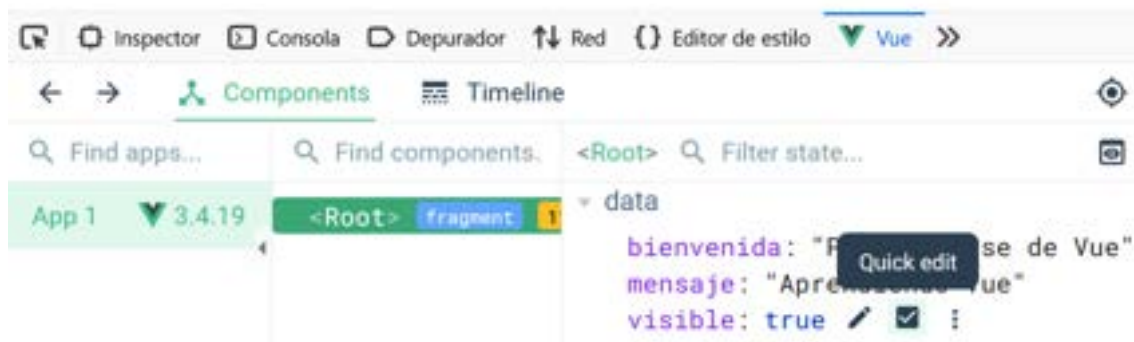
En nuestro **index.html**, creamos un nuevo párrafo con la directiva **v-show** y le pasaremos el nombre de esta variable. El texto que contiene el párrafo se verá mientras el contenido de la variable **"visible"** siga siendo **true**, si llegara a cambiar por algún motivo, Vue se encargará de ocultarla y le aplicará **display:none;** a ese párrafo.

```
index.html
1 <div id="app" v-cloak>
2   <h1>{{mensaje}}</h1>
3   <p v-once>{{bienvenida}}</p>
4   <p v-show="visible">Esto se verá si la variable "visible" posee
   valor true</p>
5 </div>
6
```

Aprendiendo Vue

Primer clase de Vue

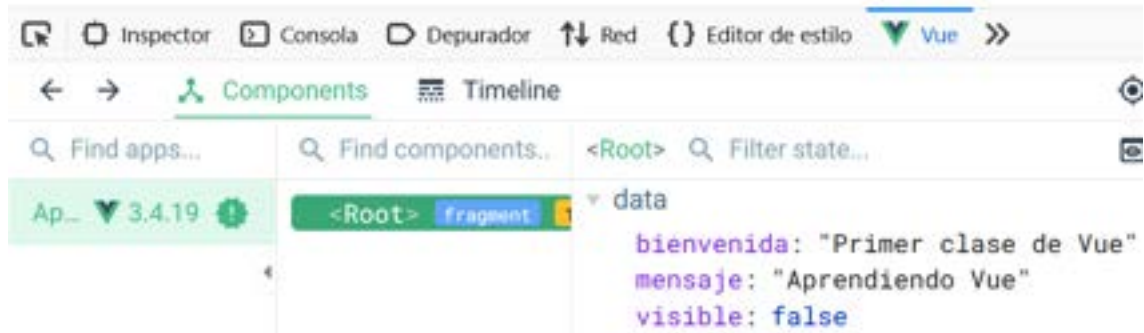
Esto se verá si la variable "visible" posee valor true



Si abrimos la consola podemos cambiar rápidamente su valor a **"false"** y observar que este párrafo deja de ser mostrado al usuario.

Aprendiendo Vue

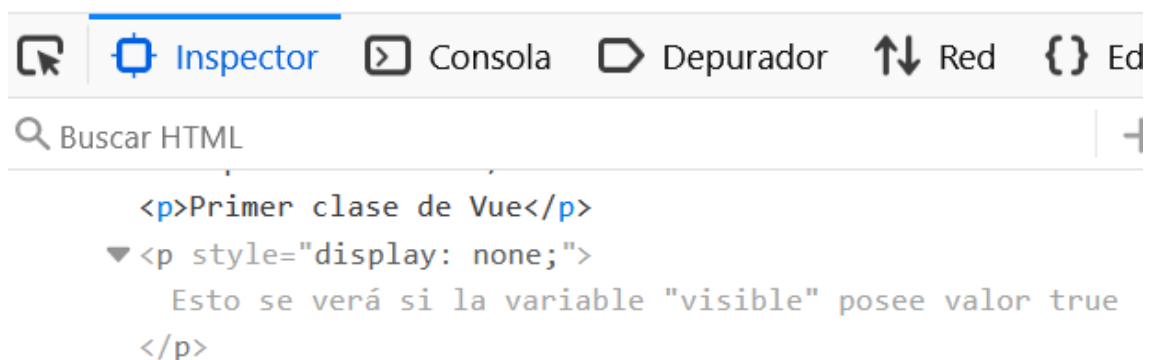
Primer clase de Vue



Si vamos a la solapa **"Inspector"** en la consola veremos que a ese texto Vue le aplica mediante el atributo style de CSS, la propiedad display:none;

Aprendiendo Vue

Primer clase de Vue



v-html

Algo que tenemos que tener presente es que las interpolaciones interpretan los datos que inicializamos como texto plano, no etiquetas HTML.

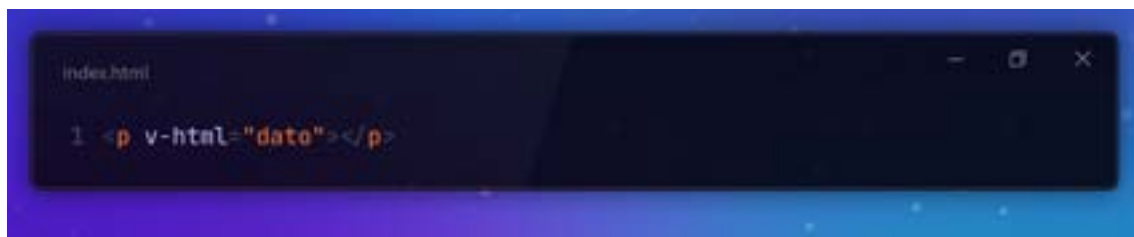
Si queremos que una propiedad en la lógica contenga una etiqueta HTML tenemos que usar la directiva v-html.

Por ejemplo, si queremos pasarle a la propiedad **"dato"** una etiqueta HTML que sea un **span** para renderizarla dentro de un párrafo en nuestro index podríamos tener el siguiente código:



```
app.js
1 const app = Vue.createApp({
2   data() {
3     return {
4       dato: '<span>Soy un texto dentro de un span</span>'
5     }
6   },
7 });
8 app.mount('#app');
```

En este caso nuestro html tendrá la siguiente sintaxis:



```
index.html
1 <p v-html="dato"></p>
```

La consola nos mostrará efectivamente que es añadido como un span dentro del párrafo donde asignamos la variable dato a la directiva v-html

```
▼ <p>
  <span>Soy un texto dentro de un span</span>
</p>
```

Hay que ser muy cuidadosos con este tipo de directiva. Si el párrafo tuviera algún texto adicional, sería sobrescrito por este span. Además, la consola nos mostraría un warning por esto.

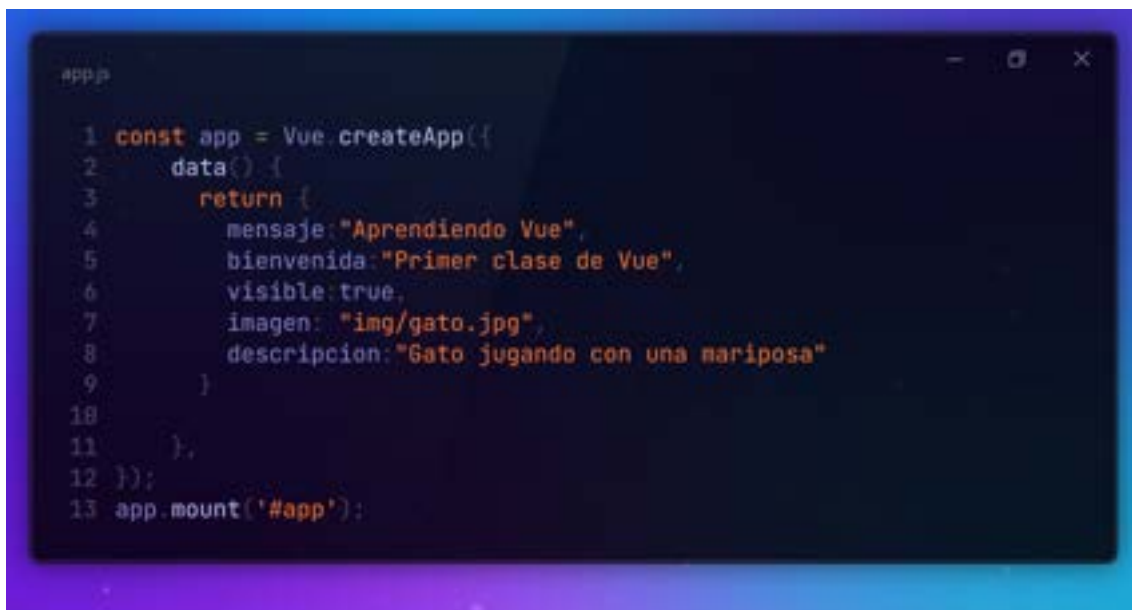
Más adelante veremos otra forma para manipular bloques de etiquetas html desde la lógica hacia la vista.

v-bind

Esta directiva sirve para enlazar o vincular una variable de Vue con un atributo HTML. La usaremos mucho a medida que vayamos profundizando en el uso de este framework.

Por ejemplo, podemos tener en nuestra lógica, en nuestro archivo app.js una variable **"imagen"**, cuyo valor será la ruta hacia una imagen que tenemos guardada en nuestros recursos locales dentro de la carpeta **"img"**. Esa ruta la vincularemos con el atributo **src** de la etiqueta ****.

Además vamos a tener la variable **"descripcion"**, que será el texto alternativo de esa imagen para poder referenciarla desde el atributo **alt** desde la etiqueta ****



```
app.js
1 const app = Vue.createApp({
2   data() {
3     return {
4       mensaje: "Aprendiendo Vue",
5       bienvenida: "Primer clase de Vue",
6       visible: true,
7       imagen: "img/gato.jpg",
8       descripcion: "Gato jugando con una mariposa"
9     }
10  }
11 });
12 app.mount('#app');
```

En el HTML tendremos la siguiente sintaxis:

```
index.html
1 
```

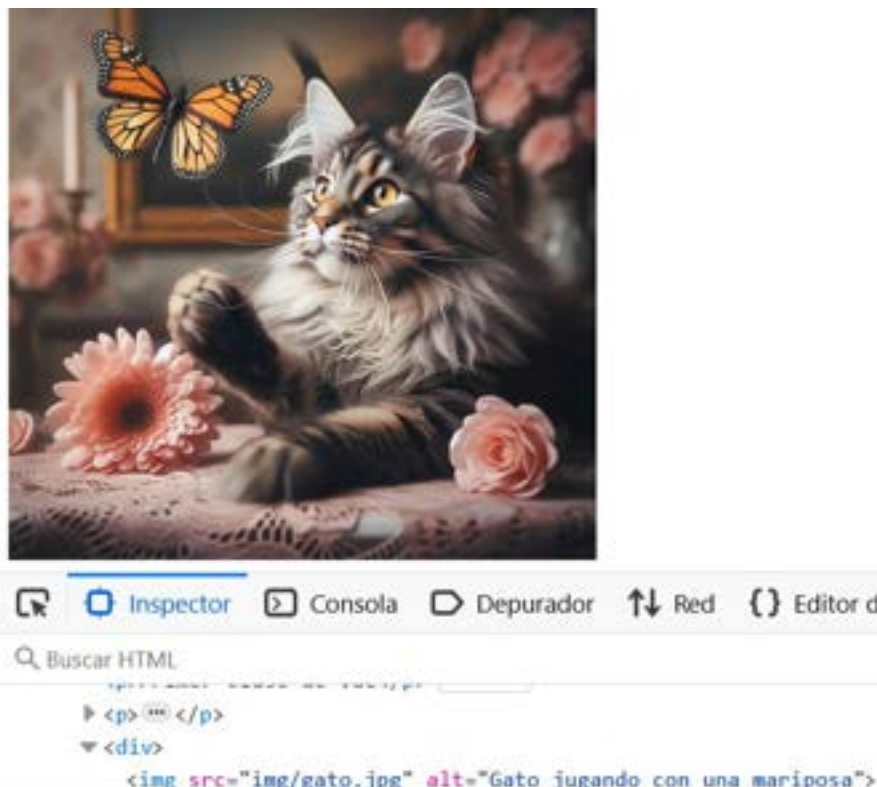
La directiva **v-bind** acepta una sintaxis abreviada, que es eliminar el texto v-bind que está por delante de los dos puntos.

":atributo=variable".

Por lo tanto, el siguiente código es igual que el anterior:

```
index.html
1 
```

En el navegador veremos la imagen del gato y la consola mostrará los valores de los atributos **src** y **alt**, como si hubiéramos escrito esta etiqueta sin la participación de Vue, de forma estática. Sin embargo, este tipo de sintaxis nos va a permitir crear elementos dinámicos que dependen de los datos o la lógica de nuestra instancia.



Manipular clases y estilos

Hay escenarios en donde queremos cambiar el aspecto de un elemento HTML, según los datos o la lógica de nuestra instancia o componente. Por ejemplo, queremos cambiar el color, el tamaño o la forma de un botón. Para hacer esto, podemos usar la **directiva v-bind**, que nos permite usar valores variables para los atributos **class** y **style** del elemento. Así, manipularemos cómo se ve el elemento sin tener que escribir string largos y complicados.

Vue nos facilita esto, permitiéndonos **usar objetos o matrices para definir los valores de clase y estilo, además de cadenas de texto**.

Por ejemplo, podemos pasar un objeto para asignar clases a un elemento del HTML.

Agreguemos otra propiedad a la instancia, en este caso se llamará **"booleano"** y su valor será **true**.



```
1 const app = Vue.createApp({
2   data() {
3     return {
4       mensaje: "Aprendiendo Vue",
5       bienvenida: "Primer clase de Vue",
6       visible: true,
7       imagen: "img/gato.jpg",
8       descripcion: "Gato jugando con una mariposa",
9       booleano: true
10    }
11  },
12  },
13 });
14 app.mount('#app');
```

En nuestro index, tendremos la directiva **v-bind** para poder cambiar la clase CSS de un **div** acorde al valor de esta variable **"booleano"**. Mientras sea **true** aplicará la clase y cuando sea **false** se la quitará.

Puede que nuestro contenedor tenga una clase CSS, si su valor es **true** lograríamos trabajar con clases combinadas de forma dinámica.

```
index.html

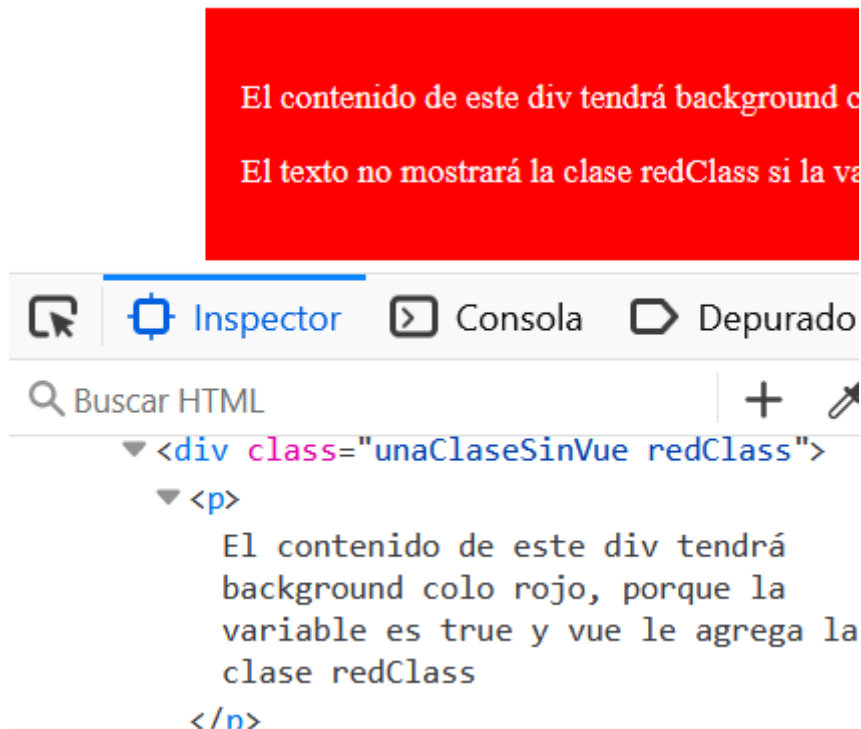
1 <div class="unaClaseSinVue" v-bind:class="{ redClass: booleano }" >
2   <p>El contenido de este div tendrá background color rojo, porque la
3     variable es true y Vue le agrega la clase redClass</p>
4   <p>El texto no mostrará la clase redClass si la variable es false
5     pero si muestra la clase estática que presenta desde css en el
6     html, sin que intervenga Vue</p>
7 </div>
```

Lógicamente estas clases deben estar definidas en nuestro archivo CSS

```
estilo.css

1 .unaClaseSinVue{
2   font-size: 1.2em;
3   padding: 1em;
4 }
5 .redClass{
6   background: red;
7   color: white;
8   width: fit-content;
9   margin-inline: auto;
10 }
```


Si examinamos la consola veremos que se están combinando ambas clases:



Si bien ahora estamos sólo inicializando variables con true o false, más adelante cuando vayamos profundizando veremos que la interacción del usuario es fundamental para lograr cambios visuales.

Además de lo mencionado, también podemos trabajar con el atributo **style** para manipular estilos CSS.

Por medio de la directiva **v-bind** podemos enlazar objetos de javascript con propiedades visuales para afectar a elementos del HTML de forma muy sencilla.

En nuestro archivo app.js podemos declarar las propiedades con valores de CSS.

```
app.js

1 const app = Vue.createApp({
2   data() {
3     return {
4       colorEncabezado: 'green',
5       tamañoFuente: 43,
6       itálica: 'italic'
7     }
8   },
9 });
10 app.mount('#app');
```

En contraparte, desde nuestro HTML tendremos alguna etiqueta que queremos afectar a nivel visual y le asignaremos un objeto con propiedades CSS (color, font-size, font-style), cuyos valores sean las propiedades declaradas en la lógica.

```
index.html

1 <h2 :style="{ color: colorEncabezado, fontSize: tamañoFuente + 'px',
   fontStyle: itálica }">Encabezado con style desde Vue</h2>
```

Aunque se recomienda escribir las **claves** en **camelCase**, **:style** también admite **claves de propiedades CSS** en **kebab-case** (corresponde a cómo se usan en el CSS real) en tal caso, iría con comillas:

```
index.html

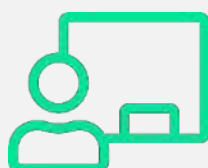
1 <h3 :style="{ 'background-color': backgroundColor }">Soy un h3 con
   fondo naranja</h3>
```

*Con esta introducción empezaremos a comprender Vue,
¡Pongamos en práctica lo que vimos hasta ahora!*



Hemos llegado así al final de esta clase en la que vimos:

- ¿Qué es Vue?
- Características de Vue.
- ¿Quién usa Vue?
- Herramienta para debug.
- Incorporar Vue a un documento html.
- Instancia de Vue.
- Interpolación.
- Directivas básicas (v-cloak, v-once, v-html, v-bind).
- Manipular clases y estilos.



Te esperamos en la **clase en vivo** de esta semana.
No olvides realizar el **desafío semanal**.

¡Hasta la próxima clase!

Bibliografía

Vue.js. (s.f.). Documentación Oficial introductoria.

Recuperado de:

<https://vuejs.org/guide/introduction.html>

Vue.js. (s.f.). Documentación Oficial sobre directivas.

Recuperado de:

<https://vuejs.org/api/built-in-directives.html#built-in-directives>

Vue.js Devtools. (s.f.). Guía de instalación.

Recuperado de:

<https://devtools.vuejs.org/guide/installation.html>