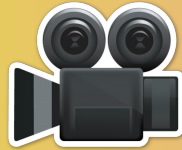




Clase 14. Vue JS

Presentación de Vue Cli 3

RECORDÁ PONER A GRABAR LA CLASE





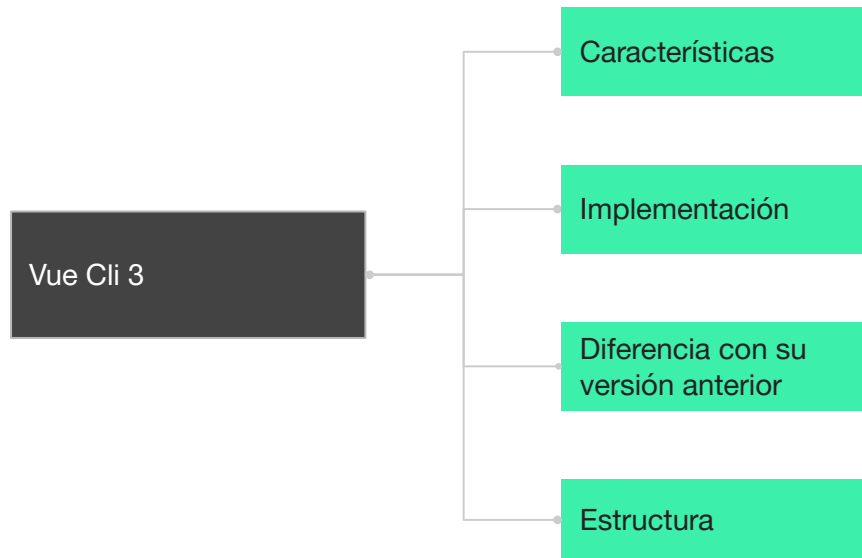
OBJETIVOS DE LA CLASE

- Analizar lo novedoso de la nueva versión de Vue CLI 3 y las ventajas de uso.

MAPA DE CONCEPTOS

MAPA DE CONCEPTOS CLASE 14

¡Para
recordar!



CRONOGRAMA DEL CURSO

Clase 13



Vue en Producción



BUILD DE UN PROYECTO
VUE CLI



EJEMPLO EN VIVO



INTEGRAR PROYECTO DE
PRODUCCIÓN EN SERVIDOR
NODE



ENTREGA INTERMEDIA

Clase 14



Vue CLI 3



ESTRUCTURA DE
PROYECTO VUE CLI 3



EJEMPLO EN VIVO



CREAR COMPONENTES
CON VUE CLI 3

Clase 15



Vue CLI 3 - características avanzadas



EJEMPLO DE USO DE
COMPOSITION API



EJEMPLO EN VIVO



FORMULARIO VUELIDATE
CON VALIDACIONES SIMPLES
Y PERSONALIZADAS



DE VUE 2 A VUE 3

VUE 3

VUE 3

¡Llegó el momento!

Como “*no hay 2 sin 3*”, es el turno de saltar a la última versión de Vue y conocer así sus características, similitudes, y diferencias respecto a su anterior versión: **Vue Cli 2.**

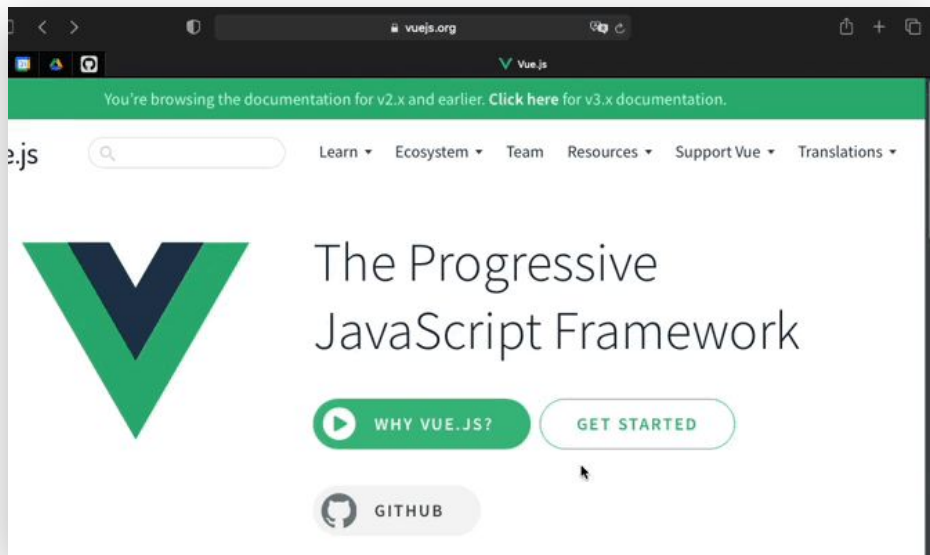


VUE 3

Ten presente que, al momento de elaborar este curso, **Vue 3** no tenía su documentación oficial traducida al español. Si manejas inglés, puedes ayudar a traducir parte de la misma.



<https://v3.vuejs.org/>



VUE 3

Vue 3 cuenta con nuevas características notables y con una ligera diferencia de lo que Vue 2 nos tiene acostumbrados, por lo tanto:

¿tengo que aprender todo de nuevo, olvidando lo visto con Vue/Cli 2?



La respuesta es ¡no!

Todas las características de **Vue 2** siguen presentes en **Vue 3**, por retro-compatibilidad. Además, se agregaron nuevas funcionalidades.



VUE 3

Algo que suele suceder en librerías y frameworks es que, con cada nueva versión, su tamaño crece lo cual conlleva a una menor performance respecto a su versión anterior.

Por suerte, **Vue 3** fue reescrito desde cero lo que ayudó a que no incrementará más de **19 kb** su tamaño, respecto a su versión anterior.

Vue 2

Asset	Size	Chunks
js/ui-c0266b6bd5a26196d002.js	413 KiB	0 [em:
js/ui-c0266b6bd5a26196d002.js.br	91.5 KiB	[em:
js/ui-c0266b6bd5a26196d002.js.gz	109 KiB	[em:
js/ui-c0266b6bd5a26196d002.js.map	1.58 MiB	0 [em:
js/ui-c0266b6bd5a26196d002.js.map.br	335 KiB	[em:
js/ui-c0266b6bd5a26196d002.js.map.gz	407 KiB	[em:

Vue 3

Asset	Size	Chunks
js/ui-34caf8d12584e4e16014.js	432 KiB	0 [em:
js/ui-34caf8d12584e4e16014.js.br	96.4 KiB	[em:
js/ui-34caf8d12584e4e16014.js.gz	116 KiB	[em:
js/ui-34caf8d12584e4e16014.js.map	1.8 MiB	0 [em:
js/ui-34caf8d12584e4e16014.js.map.br	372 KiB	[em:
js/ui-34caf8d12584e4e16014.js.map.gz	460 KiB	[em:

COMPOSITION API

COMPOSITION API

Si eres tecnólogo de la programación, seguramente habrás leído de Vue 3 en muchos rincones de Internet.

Y es muy probable que, en todos ellos, te hayas topado con la flamante característica denominada:

Composition API

Pero, al fin y al cabo, ¿qué es *Composition API*?



COMPOSITION API

Vue 3 propone una nueva forma de exponer el funcionamiento de su Core basado, por supuesto, en APIs.

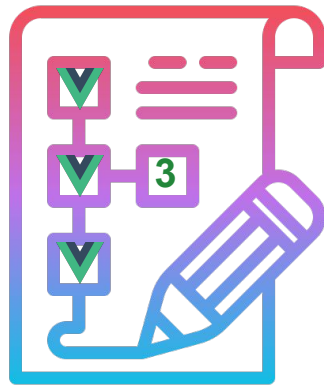
Esta nueva filosofía propuesta fue bautizada como Composition API y se encarga, entre otras cosas, de mejorar cualquier punto débil que las aplicaciones de gran escala pueden llegar a sufrir cuando son construidas.



COMPOSITION API

Dentro de la reestructuración que Vue llevó a cabo en su versión 3, aprovechó a mejorar su Core y Performance facilitando al máximo la organización y reutilización de código.

A lo largo de esta clase, veremos algunas de estas particularidades que Vue 3 propone, para reutilizar a pleno la lógica entre distintos componentes Vue, de la mano de simples funciones.



COMPOSITION API

Al elaborarse su Core prácticamente desde cero, se aprovechó esta tarea para no solo optimizarlo, sino también para mejorar su performance y compatibilidad con el lenguaje de programación **TypeScript**.

Vue ya cuenta desde su versión 2 con esta compatibilidad, pero en su versión 3 mejoró mucho más dado que el **Core** de Vue fue escrito íntegramente con este lenguaje.



COMPOSITION API

Tengamos presente que, más allá de que Composition API nace junto a Vue 3, esta herramienta en sí cuenta con una dependencia pensada para implementarse en aplicaciones Vue 2.

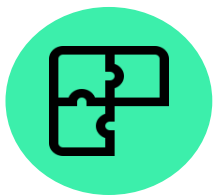
Por lo tanto, si quieres llevar una o más características de Composition API a tu ecosistema Vue 2, instala vía **NPM** el plugin dedicado para tal fin:

[@vue/composition-api](https://github.com/vuejs/composition-api)



INSTALACIÓN





INSTALACIÓN

El primer paso para ir a la nueva versión de Vue es verificar que las herramientas base que tenemos instaladas, cumplan con la versión mínima necesaria.

Ejecutemos en la Terminal los siguientes comandos:

```
$> node --version
```

Y, para verificar la versión de vue/cli:

```
$> vue --version
```

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE

MacBook-Air:vue fernando$ node --version
v14.16.1
MacBook-Air:vue fernando$ vue --version
@vue/cli 4.5.15
MacBook-Air:vue fernando$
```

La versión de Node debe ser superior a la 10.0 mientras que la de Vue/Cli, debe ser superior a la 4.5.

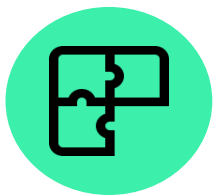


INSTALACIÓN

Si encuentras diferencias con las versiones de **Node** y/o **Vue/Cli**, siendo una o las dos menores a las versiones base mencionadas, entonces procede a actualizar Node en primera instancia y luego Vue.

Cuando estés por actualizar Vue, ejecuta el siguiente comando en la ventana Terminal 🖱️

```
$> vue upgrade
```



INSTALACIÓN

```
package.json > ...  
  "build": "vue-cli-service build",  
  "lint": "vue-cli-service lint"  
},  
"dependencies": {  
  "core-js": "^3.6.5",  
  "vue": "^2.6.11"  
},
```



```
package.json > ...  
  "build": "vue-cli-service build",  
  "lint": "vue-cli-service lint"  
},  
"dependencies": {  
  "core-js": "^3.6.5",  
  "vue": "^3.0.0"  
},
```

Si realizas este último proceso con un proyecto Vue/Cli editado en VS Code, encontrarás al finalizar el proceso, que el mismo fue migrado de la versión **2.x** a la **3.0**.

Ten precaución de cerrarlo previamente, si no quieres actualizar un proyecto Vue 2.



INSTALACIÓN

Dentro de las particularidades de **Vue 3**, encontraremos que muchas de las dependencias deben ser actualizadas de forma manual, ya que no se actualizan automáticamente con el proceso de actualización de Vue.

Vue-Router es una de ellas, por lo tanto, si queremos actualizarla, debemos ejecutar el siguiente comando en la ventana Terminal:

```
$> npm install vue-router@4
```

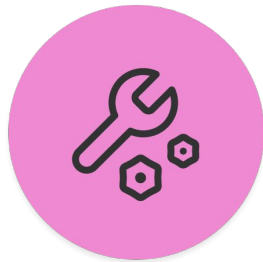


INSTALACIÓN

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  bash + - [ ] [ ] ^ x
MacBook-Air:vue3-app fernando$
```

Con todo correctamente verificado y/o actualizado, ya podemos crear nuestra primera aplicación Vue 3, desde el cliente Vue/Cli.





PROYECTO VUE CLI 3

Crea la estructura base de un proyecto Vue-Cli 3.



PROYECTO VUE CLI 3

Ya adquirimos los lineamientos base para utilizar proyectos Vue Cli 3.

Con esta idea clara de todo lo que hay que verificar y/o actualizar, pongamos manos a la obra y creemos un proyecto utilizando la ventana Terminal, que responda a la versión 3 de Vue.

Una vez que tengas la estructura base del mismo, modifica algún componente, texto o elemento predeterminado y valida dicho cambio ejecutando el proyecto en el navegador web.

Cuentas con **10 minutos** para realizar esta actividad.

VUE 3: CARACTERÍSTICAS

VUE 3: CARACTERÍSTICAS

Veamos entonces qué buenas nuevas nos trae Vue 3 respecto a su anterior versión:

Los cambios son varios y uno muy importante es que Vue 3 permite modular aún más las funcionalidades propias eligiendo, **importación mediante**, solo aquellas que vamos a utilizar previo a implementarlas en nuestras aplicaciones.



APP VUE 3



CARACTERÍSTICAS: APP VUE 3

```
import Vue from 'vue'
import App from './App.vue'

Vue.config.productionTip = false

new Vue({
  render: h => h(App),
}).$mount('#app')
```

Vue 2

En Vue 2 Instanciamos el objeto `Vue()` parametrizándolo con el contenido a renderizar para, finalmente, montarlo en `#app`.
En la estructura de `main.js` de Vue 3, podemos visualizar un cambio notorio. 🙌

Vue 3 reemplaza la instanciación por el método `createApp`, el cual debemos importarlo previamente a utilizarlo, enviándole a este como parámetro el objeto `App` para, finalmente, invocar el método `mount()`.

```
import { createApp } from 'vue'
import App from './App.vue'

createApp(App).mount('#app')
```

Vue 3



CARACTERÍSTICAS: APP VUE 3

```
import { createApp } from 'vue'
import router from './router'
import App from './App.vue'

const appR = createApp(App)
appR.use(router)
appR.mount('#appR')

//Aquí no necesito ruteo.
const appB = createApp(App)
appB.mount('#appB')
```

Esta técnica nos permite definir elementos determinados a incorporar en la instancia de Vue, como ser router, Vuex o el llamado a alguna API y que estos se apliquen específicamente sobre la instancia en cuestión.

createApp()

Como podemos apreciar en el último slide, `createApp()` marca la diferencia con esta nueva modalidad de implementación.

Así lograremos, por ejemplo, que nuestra aplicación Vue 3 no deba cargar objetos innecesarios en determinados apartados de la misma, optimizando notablemente su performance.

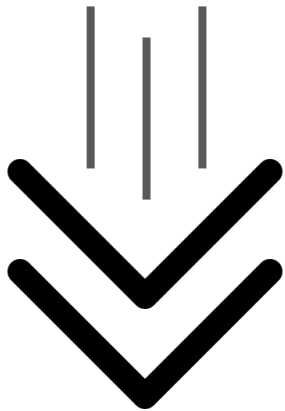
TREE SHAKING

CARACTERÍSTICAS: TREE SHAKING



Haciendo una analogía directa con el trabajo de agitar el árbol para cosechar los productos maduros, en el mundo de la programación el **tree-shaking** es una técnica que se utiliza para **eliminar código muerto**, la cual se aplica para optimizar las aplicaciones en general.

CARACTERÍSTICAS: TREE SHAKING



Por lo tanto, aquí, en Vue 3, la característica denominada `tree shaking` refiere a “*la reestructuración*” que mencionamos al inicio de esta clase.

Varias APIs globales expuestas a través del objeto Vue ya no están accesibles de forma directa, sino mediante su importación previa.

👁️ Veamos, a continuación, cuáles son:



CARACTERÍSTICAS: TREE SHAKING

- `nextTick`
- `observable`
- `version`
- `compile`
- `set`
- `delete`

👉 Este es el listado de elementos que ahora requieren una importación previa dentro de nuestra aplicación Vue, para luego recién poder utilizarlos.

El siguiente ejemplo es aplicable a cada una de ellas, simplemente reemplazando su nombre entre las llaves.

```
import { nextTick } from 'vue'
```

CARACTERÍSTICAS: TREE SHAKING



El aporte de la metodología **Tree Shaking** lo veremos reflejado al llevar el proyecto a producción.

El mismo estará mucho más “*optimizado*”, habiendo eliminado durante el proceso de “*build*” todo aquello que no es requerido.

SETUP



CARACTERÍSTICAS: SETUP

```
setup (props, context) {  
  context.attrs  
  context.slots  
  context.parent  
  context.root  
  context.emit  
}
```

Es un nuevo método que encontramos dentro de los componentes vue. El mismo debe establecerse justo antes de que el componente sea creado, más luego después de haber establecido el valor de **props**.

Pensando en detalle esta particularidad, notamos que la instancia del componente aún no está creada, por lo cual no dispone de acceso a **data**, **methods**, o **propiedades computadas** que se hayan definido.



CARACTERÍSTICAS: SETUP

Por lo tanto, lo que setup retorna estará disponible en todo el componente vue. Es decir, si aquí definimos (`computed properties`, `methods`, `lifeCycle functions`, `data`, `templates`, etcétera) todo será accesible de forma global dentro de dicho componente.

Una de sus ventajas es la de **contar con más control de lo que se expone para con nuestro template.**

```
src/components/miComponente.vue
```

```
export default {  
  props: {  
    character: {  
      type: String,  
      default: 'Coderhouse'  
    }  
  },  
  setup(props, context) {  
    console.log(props)  
    return {}  
  }  
  // Aquí, la definición  
  del componente  
}
```



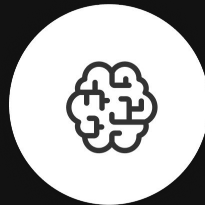
CARACTERÍSTICAS: SETUP

Además de que `setup` puede devolver datos para que estos se visualicen en el template, también contamos con la posibilidad de que haga un retorno del renderizado del propio componente.

```
import { h } from 'vue'

export default {
  props: {
    character: {
      type: String,
      default: 'Coderhouse'
    }
  },
  setup() {
    return () => h('div',
      [props.character])
  }
  // ...
}
```


REFS



¡PARA PENSAR!

Es hora de ejercitar un poco la mente.

¿Se acuerdan cuál era la función de `$refs`?

SI LA RECUERDAS, CUÉNTALA
BREVEMENTE EN EL CHAT DE ZOOM.



CARACTERÍSTICAS: REFS

```
import { ref } from 'vue'
```

Las referencias `$refs` son un objeto en el cual se almacenan los elementos DOM que cuentan con el atributo `ref`, dentro del template.

CARACTERÍSTICAS: REFS

En Vue 3, **ref** está disponible como un método de Composition API el cual se ocupa de hacer que una variable sea reactiva. Por lo tanto, ahora contamos con la posibilidad de crear objetos reactivos y que no estén asociados a un componente en particular.

```
import { ref } from 'vue'

export default {
  setup() {
    const season = ref(0)
    // expone en el
    // template
    return {
      season
    }
  }
}
```



CARACTERÍSTICAS: REFS



ref se encarga de tomar el argumento en cuestión, retornándolo junto a un objeto con un valor como propiedad, el cual puede utilizarse para acceder o mutar el valor de una variable reactiva.

CARACTERÍSTICAS: REFS

pass by reference



fillCup()

pass by value



fillCup()

Retornar valores dentro de un objeto puede ser muchas veces innecesario pero requerido a su vez, ya que mantiene su comportamiento unificado a lo largo de los diferentes tipos de datos en JavaScript.

Más aún, teniendo presente que en JS un tipado primitivo como **number** o **string** suele ser pasado como valor y no como referencia.

VUE ROUTER



CARACTERÍSTICAS: VUE ROUTER

```
import Vue from 'vue'  
import Router from 'vue-router'
```

```
Vue.use(Router)
```

```
export default new Router({  
  mode: 'history'  
  router: [...]  
})
```

Vue 2

Como mencionamos anteriormente, **Vue Router** cambió su lógica de uso original.

Aquí 🙌, un ejemplo de Router en Vue 2.

Ahora se invoca con el método `createRouter()`.

Debemos definir el tipo de historial a usar, entre `webHashHistory`, `createMemoryHistory` y `createWebHistory`.

```
import {createRouter, createWebHistory} from  
'vue-router'
```

```
const router = createRouter({  
  history: createWebHistory(),  
  redirect: '/',  
  router: [...]  
})
```

```
export default router
```

Vue 3

CARACTERÍSTICAS: VUE ROUTER

Veamos, a continuación, una descripción de las características de ruteo de la nueva versión de Vue:

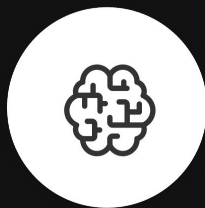
<i>HISTORIAL</i>	<i>DESCRIPCIÓN</i>
webHashHistory	Vue espera en la URL Del navegador, encontrar el caracter # (hash) y, en base a esto, determinar la ruta de navegación correcta.
createMemoryHistory	Mantendrá la SPA como tal, desarrollando la navegación por las diferentes rutas de la aplicación, sin generar un historial accesible desde el navegador web.
createWebHistory	Esta opción habilita el historial web, tal como lo conocemos: www.miurl.com/ruta1 , www.miurl.com/ruta2 , etcétera. Además habilita poder navegar mediante los botones de la barra de herramientas del navegador web.



BREAK

¡5/10 MINUTOS Y VOLVEMOS!

LIFE CYCLE



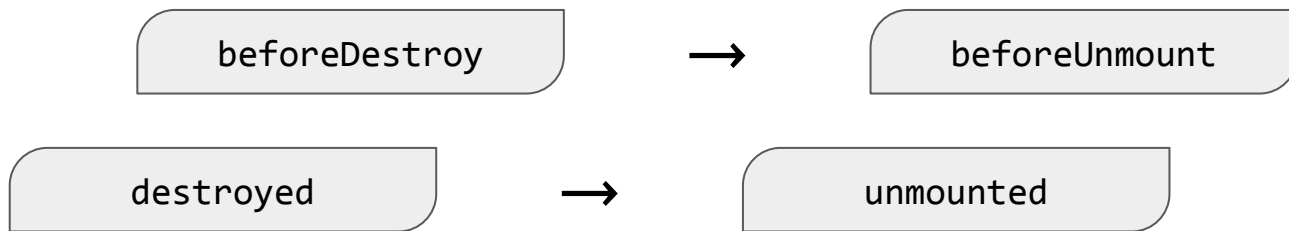
¡PARA PENSAR!

*Quando tratamos **Life Cycle Hooks** vimos que era un tema de complejidad media, pero sumamente interesante y útil.*

*¿Recuerdan cuáles eran algunos de estos eventos que aplicaban con otro nombre a partir de **Vue 3**?*

MENCIÓNALOS A TRAVÉS DEL CHAT DE ZOOM.

LIFE CYCLE



`beforeDestroy` y `destroyed` son los eventos life cycle hooks de **Vue 2** que cambian su nombre por `beforeUnmount` y `unmounted`, respectivamente, a partir de la versión 3 del framework.

LIFE CYCLE



Los eventos **Life Cycle** aprendidos en Vue 2, ayudaban a distribuir tareas claves de nuestra aplicación, a lo largo de su inicio y finalización: (*renderizado, peticiones API, guardar estados, entre otros*).

LIFE CYCLE

Dichos eventos siguen estando disponibles en Vue 3 aunque, en su última versión, contamos con la posibilidad de integrarlos dentro de la función `setup()`, concatenándolos con el prefijo `on`.

- `onBeforeMount`
- `onMounted`
- `onBeforeUpdate`
- `onUpdate`
- `onBeforeUnmount`
- `onUnmounted`
- `onActivated`
- `onDeactivate`
- `onErrorCaptured`



LIFE CYCLE

```
import { onBeforeMount, onMounted, onBeforeUpdate, onUpdated,  
onBeforeUnmount, onUnmounted, onActivated, onDeactivated,  
onErrorCaptured } from 'vue'
```

Tengamos en cuenta también que debemos importarlos previo a utilizarlos en nuestra aplicación.

Podremos, por supuesto, importar sólo aquellos eventos que consideremos necesario implementar.



LIFE CYCLE

Y su forma de implementarlos dentro de `setup()` es tan simple tal como vimos en los ejemplos abordados con Vue 2.

```
export default {  
  setup() {  
    onBeforeMount(() => {  
      // ...  
    })  
    onMounted(() => {  
      // ...  
    })  
    onBeforeUpdate(() => {  
      // ...  
    })  
    onUpdated(() => {  
      // ...  
    })  
    onBeforeUnmount(() => {  
      // ...  
    })  
    onUnmounted(() => {  
      // ...  
    })  
    onActivated(() => {  
      // ...  
    })  
    onDeactivated(() => {  
      // ...  
    })  
    onErrorCaptured(() => {  
      // ...  
    })  
  }  
}
```

LIFE CYCLE: ONE MORE THING...

Como broche adicional, life cycle incluye dos nuevos métodos a su set de control de ciclo de las aplicaciones Vue:

(on)renderTracked: el cual se invoca al acceder por primera vez a una dependencia reactiva en la función de render.

(on)renderTriggered: el cual es invocado al producirse un nuevo render del componente, permitiendo conocer qué lo disparó.

TELEPORT

TELEPORT



Teleport es una nueva funcionalidad de Vue que nace para organizar y reutilizar todos los bloques de código que puedan aprovecharse, en pos de reducir el código al máximo posible y así alcanzar una mayor performance, de cara a implementar el proyecto en producción.



TELEPORT

```
<template>
  <teleport to="#carreras-ch">
    <p>Frontend Developer</p>
    <p>Backend Developer</p>
    <p>Producto</p>
    <p>Data & Analytics</p>
  </teleport>
</template>
```

Su implementación se realiza a través de un tag dentro de `<template>`, al cual le definimos a través del atributo `to`, el destino de la porción de `template` que deseamos repetir en otra u otras secciones de nuestra aplicación.



TELEPORT

```
<div id="#app">  
  <h1>Las carreras más demandadas</h1>  
  <div id="carreras-ch"></div>  
  ...  
</div>
```

De esta forma, podemos trasladar rápidamente a cualquier parte del DOM la porción de template enmarcada bajo el tag `<teleport>`, simplemente referenciando en el origen cuál es el `id` de destino.

SUSPENSE

SUSPENSE



Esta etiqueta nos habilita la visualización de un contenido específico durante el período de tiempo que una operación asincrónica se esté ejecutando, hasta tanto finalice y se puedan visualizar los resultados de esta.



SUSPENSE

👉 Por un lado tenemos definido un componente padre, `miComponente`, el cual dispone internamente de una operación asincrónica para obtener datos remotos.

Todo esto, se verá reflejado en un `<template>`.



```
export default {  
  name: 'MiComponente',  
  async setup () {  
    const result = await fetch(URL)  
    return {  
      result  
    }  
  }  
}  
  
.....  
.  
  
<template>  
  <div>  
    {{ result }}  
  </div>  
</template>
```



SUSPENSE

El componente padre, cuenta con el siguiente `template`.

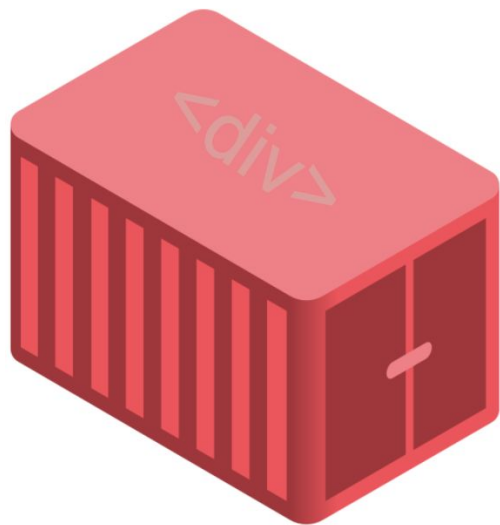


El apartado definido como `#fallback` mostrará el mensaje o animación que tenga definido durante el período de tiempo que demore la operación asincrónica, reemplazando la Vista por el resultado de esta apenas finalice.

```
<Suspense>
  <template #default>
    <MiComponente />
  </template>
  <template #fallback>
    Cargando...
  </template>
</Suspense>
```

MÚLTIPLES ELEMENTOS ROOT

MÚLTIPLES ELEMENTOS ROOT



Finalmente, una gran mejora que esperábamos era la de no tener que utilizar elementos `<div>` dentro de los templates como contenedores de dos elementos que podíamos poner contiguos o sueltos.



MÚLTIPLES ELEMENTOS ROOT

Si bien, podemos seguir utilizando DIVs en aquellos templates complejos que integran varios elementos HTML, ya no es requisito incluirlos para determinadas cosas simples, como el siguiente ejemplo brindado:

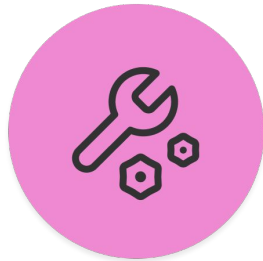
```
<template>
  <h1>Elementos Root</h1>
  <p>Vue 3 ya no exige DIVs para contenernos! 🙌 </p>
  <h2>Vue 3</h2>
</template>
```

Ejemplo
en vivo



¡VAMOS A PRACTICAR!

CODER HOUSE



CREANDO NUEVOS COMPONENTES

En el proyecto base construido en la práctica anterior, agrega y visualiza componentes Vue.

Tiempo estimado: 20 minutos



CREANDO NUEVOS COMPONENTES

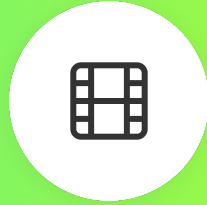
Reciclando el proyecto base anteriormente construido, agrega al menos dos componentes implementando en estos las nuevas características de Vue 3, como ser: **setup()**, **Suspense**, **Teleport**, o alguna otra de las aprendidas hoy.

Ejecuta luego el proyecto para validar que todo funciona correctamente.

Cuentas con **20 minutos** para realizar esta actividad.

¿PREGUNTAS?





***¿QUIERES SABER MÁS? TE DEJAMOS
MATERIAL AMPLIADO DE LA CLASE***

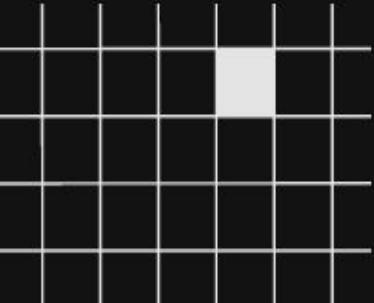


- [Composition API para Vue 2](#) | **Vue**
- [Vue 3: web oficial](#) | **Vue**
- [Vue 3 vía CDN \(*para prototipado*\)](#) | **Vue**



¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Vue 3
 - Composition API
 - Ciclos de vida
- 



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN

CODER HOUSE