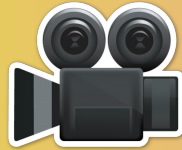




Clase 4. Vue JS

Componentes Web

RECORDÁ PONER A GRABAR LA CLASE





OBJETIVOS DE LA CLASE

- Dar los primeros pasos en la creación y uso de un componente de vista en Vue CDN con pasaje de parámetros por props.

CRONOGRAMA DEL CURSO

Clase 3



Directivas estructurales, condicionales y de atributo



BOTÓN PARA
MOSTRAR Y OCULTAR
HTML



TABLAS DINÁMICAS

Clase 4



Componentes



CREANDO COMPONENTES



PASAJES DE
PARÁMETROS



PROYECTO VUE CDN

Clase 5



VueJS en NodeJS



USANDO VUE CLI 2

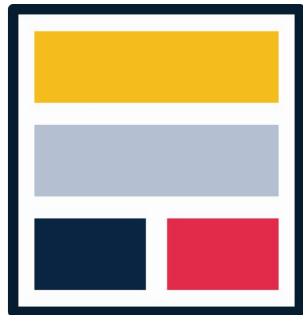


SERVIDOR DE DESARROLLO
AUTOMÁTICO

COMPONENTES WEB EN VUE JS

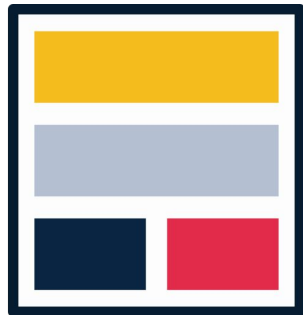
¿QUÉ SON LOS COMPONENTES WEB?

COMPONENTES WEB



Un **componente web** es básicamente un **elemento** que **cuenta con una Vista, Estilos y Lógica propia**, que puedes reutilizar todas las veces que sean necesarias dentro de un proyecto web.

COMPONENTES WEB



Para hacer una comparativa con el ecosistema web, un componente web equivale a un elemento HTML que utilizamos en una página homónima, como serían: `<h1>`, `<div>`, `<image>`, etcétera.

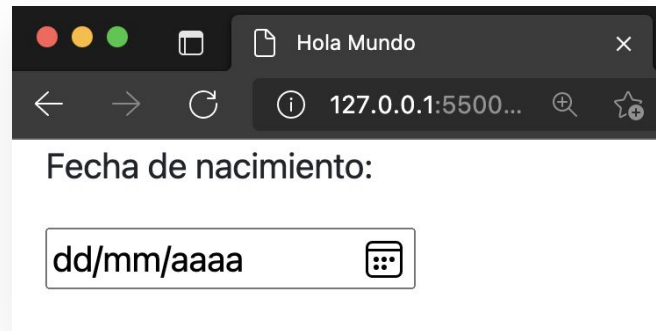


COMPONENTES WEB

```
<input type="date" min="2021-01-01" max="2021-10-31">
```

Cuando creas un elemento html, como ser: `<input type="date">`, estás utilizando un componente web.

El mismo se renderiza en el documento HTML a partir de un estilo predefinido (*bordes, fuentes, color de fuente, ícono, etcétera*) y de una lógica predeterminada (*atributos `min`, `max`, `value`, etcétera*).



COMPONENTES WEB

React y **Angular** permiten crear componentes web, como también **Vanilla JS** desde hace pocos años.

Vue se sumó a esta movida, integrando esta fabulosa capacidad con el fin de poder crearlos, escribiendo menos código que el nativo de JS, y para poder desarrollar elementos complejos y reutilizables a lo largo del ciclo de vida de nuestro proyecto.



VENTAJAS



COMPONENTES WEB: VENTAJAS



//CÓDIGO DE BOOTSTRAP NAVBAR

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Navbar</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent"
      aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav me-auto mb-2 mb-lg-0">
        <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="#">Home</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Link</a>
        </li>
        <li class="nav-item dropdown">
          <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-bs-toggle="dropdown"
            aria-expanded="false">
            Dropdown
          </a>
          <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
            <li><a class="dropdown-item" href="#">Action</a></li>
            <li><a class="dropdown-item" href="#">Another action</a></li>
            <li><hr class="dropdown-divider"></li>
            <li><a class="dropdown-item" href="#">Something else here</a></li>
          </ul>
        </li>
        <li class="nav-item">
          <a class="nav-link disabled" href="#">Disabled</a>
        </li>
      </ul>
      <form class="d-flex">
        <input class="form-control me-2" type="search" placeholder="Search" aria-label="Search">
        <button class="btn btn-outline-success" type="submit">Search</button>
      </form>
    </div>
  </div>
</nav>
```

Navbar Home Link Dropdown ▾ Disabled

Search

Search

Para crear esta  **Navigation Bar**, necesitas  todo este código HTML, sumado al código de las clases CSS y su lógica funcional en JavaScript.

Además, si tu proyecto web tiene varias páginas HTML que usan la misma Navigation Bar, terminarás repitiendo esto una cantidad significativa

de veces... 



COMPONENTES WEB: VENTAJAS

```
<navigation-bar :class="backcolor-lightgrey"  
                visibleSearchBar="true">  
</navigation-bar>
```

Allí es donde los Componentes Web sacan partido porque desarrollas un componente con todos los estilos CSS y la lógica JS y, luego, en tus documentos HTML, agregas solamente un tag más algunos atributos que le den forma.

De esta manera, ahorrarás decenas de líneas de código repetitivo, además de centrar el mantenimiento desde un único punto que luego será reflejado en todo tu proyecto.





COMPONENTES WEB: VENTAJAS

```
<nav>
  <div>
    <a class="navbar-brand" href="#">Navbar</a>
    ...
```

Otra gran ventaja de los componentes web es que, al crearlo, podemos anidar en su interior una diversidad de etiquetas HTML estándar para darle la forma que necesitamos que tenga. Combinaremos etiquetas `div`, `h1`, `p`, `img`, y otras que necesitemos, hasta conseguir el diseño que deseamos.



LIMITACIÓN

```
<nav>
  { <div class="div-primario"> }
    <p>titulo</p>
  </div>
  { <div class="div-interno"> }
    <p>detalle</p>
  </div>
</nav>
```



La limitación que tenemos al crear componentes es la de no poder definir, en un mismo nivel de anidamiento, dos etiquetas del mismo tipo.

LIMITACIÓN

¡Para
recordar!



Sí está permitido que, una misma etiqueta, quede anidada dentro de otra etiqueta de similares características.



```
<nav>
  <div class="div-principal">
    <div class="div-interno">
      <p>descripcion</p>
    </div>
  </div>
</nav>
```


CREAR UN COMPONENTE



CREAR UN COMPONENTE

```
Vue.component(mi-componente', {  
  Array de opciones...  
})
```

Para crear un componente web, debemos hacerlo invocando el método `.component()` de **Vue**. Al declararlo, debes hacerlo antes de instanciar a Vue y, además, debes agregarle al método dos parámetros:

nombre del componente y **array de opciones.**



NOMBRE DEL COMPONENTE

```
Vue.component('mi-componente',{...})
```

Su nombre es lo que luego agregamos como `<tag>` en el documento HTML.

Existen convenciones para definirlo: (`camelCase`, `kebab-case`) aunque, lo más común es hacerlo con esta última opción, ya que permite distinguir rápidamente los componentes web de los elementos HTML estándar.

El componente de 🖱️ ejemplo se verá 🖱️ así, en un documento HTML:

```
<mi-componente>...</mi-componente>
```



ARRAY DE OPCIONES

```
Vue.component('mi-componente',{array de opciones})
```

En el array de opciones definimos una serie de parámetros que le darán vida a dicho componente. Entre estos, los dos más importantes que debemos declarar son: **props** y **template**.

De igual manera que todo lo que declaramos en una instancia Vue, cada elemento a incluir aquí debe estructurarse separándolo por comas del resto.



NOMBRE DEL COMPONENTE

Como mencionamos anteriormente, la convención de nombre de un componente web siempre **apunta a utilizar kebab-case**.

Si bien HTML utiliza nombres para sus elementos HTML de una sola palabra, hacer componentes web que luego devienen en elementos HTML nos da la libertad de ser lo más claros posible al bautizarlos, dado que es un elemento personalizado.

Te dejamos más información en la sección Material Ampliado.

PROPS



PROPS

```
Vue.component(mi-componente', {  
  props: ['cover', 'title'],  
  ...
```

El nombre **props** hace referencia a **propiedades**. Se define en forma de **array[]**, que son los que indican los nombres de las diferentes propiedades que utilizaremos en el componente web.



PROPS

Y... *¿qué harán estas propiedades?*... envían valores que moldearán el componente web. Y, cuando nuestro proyecto esté en ejecución, las props se convertirán en **atributos** configurables mediante la directiva `v-bind`.

```
<mi-componente class="card" :cover="portada" :title="titulo">  
  ...  
</mi-componente>
```




PROPS

```
Vue.component('mi-componente', {  
  //props: ['cover', 'title'],  
  props: {  
    cover: String,  
    title: String  
  }...  
})
```

Y, en aquellos casos donde debemos ser claros sobre qué dato esperan recibir las **props**, tenemos la posibilidad de declararlas de forma tipada. Esto ayudará no solo a que recordemos fácilmente qué dato se espera, sino también a que se genere un error en la Consola JS, en el caso que se le envíe un dato no válido como parámetro.

TEMPLATE

TEMPLATE

Llegamos al objeto `template`.

A través de éste, definiremos la estructura Visual de nuestro componente web. Utilizaremos para ello **template strings**, el cual nos permite darle forma a lo que será el componente web en sí.

Debemos encerrar entre **backticks** el o los tags HTML que armarán nuestro web component.



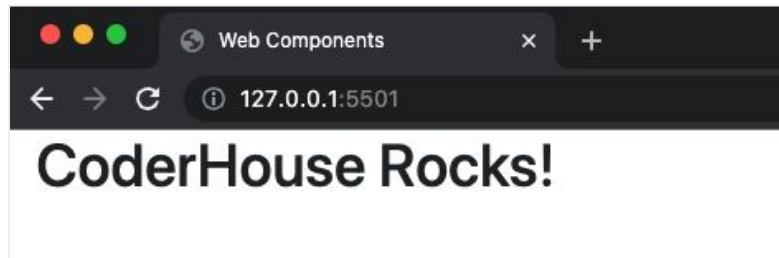
TEMPLATE

Con esta **estructura-template** ya tenemos definido un componente simple.

```
Vue.component('mi-componente', {...,  
  template: `<h1>CoderHouse Rocks!</h1>`  
})
```

```
<div id="app">  
  <mi-componente></mi-componente>  
  ...
```

Cuando se renderice en pantalla 🖱️, lo veremos tal cual lo definimos.





TEMPLATE

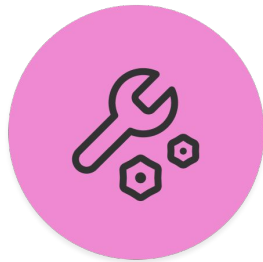
```
//REACT WEB COMPONENT
function BrickFlipbox() {
  return (
    <brick-flipbox class="demo">
      <div>front</div>
      <div>back</div>
    </brick-flipbox>
  );
}

//ANGULAR WEB COMPONENT
@Component({
  selector: 'my-popup',
  template: `
    <span>Popup: {{message}}</span>
    <button
      (click)="closed.next()">&#x2716;</button>`,
  animations: [
    trigger('state', [
```

Mencionamos al inicio de esta clase que React y Angular utilizan esta metodología, aunque de forma bastante diferente a lo que propone Vue en la creación de componentes web.

El armado de Templates en Vue se asimila al estilo de Vanilla JS al menos en la forma que propone Vue 2 para crearlos.

¡VAMOS A PRACTICAR LO VISTO!



CREA TU PRIMER COMPONENTE WEB

Diseña tu primer componente web y visualízalo en el documento HTML.

Tiempo estimado: 10 minutos.

CODER HOUSE

CREA TU PRIMER COMPONENTE WEB

Desafío
generico



Imagina un componente web simple, como ser un título de nivel `<h1>`, `<h2>`, etcétera, con un diseño personalizado. Define su estructura base y renderízalo en pantalla, agregando el mismo en **index.html**.

Tiempo estimado: 10 minutos.

CODER HOUSE

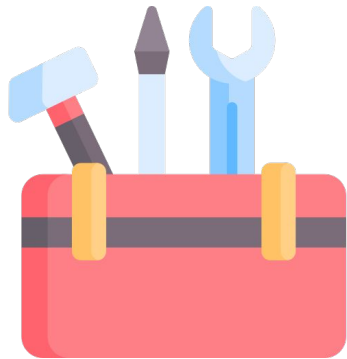


BREAK

¡5/10 MINUTOS Y VOLVEMOS!

CREAR UNA INSTANCIA DE COMPONENTE VUE

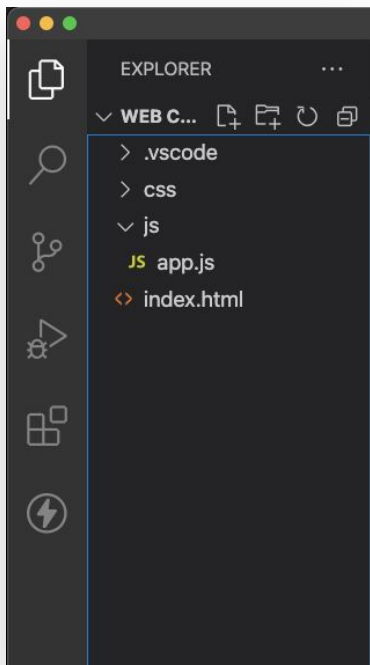
CREAR UNA INSTANCIA DE COMPONENTE VUE



Veamos a continuación cómo integrar todos estos conceptos en el armado de un componente con mayor complejidad, que sea útil en un desarrollo basado en Vue JS.



CREAR UNA INSTANCIA DE COMPONENTE VUE



Partiendo de un proyecto simple, integraremos el diseño de un componente que nos permita mostrar una tarjeta (Card) en el documento HTML de nuestro proyecto.

Dicha tarjeta recibirá como parámetros la URL de una imagen y algunos textos que compondrán su descripción.



CREAR UNA INSTANCIA DE COMPONENTE VUE

```
<body>
  <div id="app">
    <h1 class="main-title">CODERMEALS
    🍷</h1>
    <div class="container">

    </div>
  </container>
</div>
```

El documento HTML contiene en su cuerpo la estructura base de Vue. Le agregamos un título `<h1>` del primer nivel y un `<div>` el cual contendrá a nuestro componente web.

Utiliza un framework CSS como Bootstrap, para aprovechar las clases CSS que consideremos óptimas y estilizar así los elementos HTML que trabajemos.



CREAR UNA INSTANCIA DE COMPONENTE VUE

```
JS app.js x
Vue.component('codermeals-card', {
  props: {},
  template: ``,
})
```

En el archivo JS definimos en primer lugar un **Vue Component**, utilizando la instancia **Vue** seguida del método `.component()`. Le agregamos un nombre combinado, utilizando el estándar kebab-case y definimos sus objetos **props** y **template**.

PROPS TIPADAS Y TEMPLATE



PROPS TIPADAS

```
JS app.js x
Vue.component('codermeals-card', {
  props: {
    titulo: String,
    portada: String,
    costo: Number
  },
  template: ``...
```

Creamos una serie de **props tipadas**, para tener claro qué dato recibirá cada una de ellas.

Declaremos, a continuación, algo de información simple en la instancia general de **Vue**, para poder alimentar posteriormente la plantilla.

PROPS TIPADAS

Aquí tienes un listado de los diferentes tipos de datos, y su descripción, que soporta Vue en sus **props tipadas**.

- **String:** Para cadenas de texto
- **Number:** Para números
- **Boolean:** Para booleanos true/false
- **Array:** Listas
- **Object:** Objetos de javascript
- **Date:** Tipo fecha de javascript
- **Function:** Funciones
- **Symbol:** Símbolos



DATOS DE PRUEBA

```
JS app.js x
const app = new Vue({
  el: '#app',
  data: {
    titulo: "Pizza Napoletana",
    portada: "https://...738.jpg",
    costo: 675.00
  }
})
```

Estos nos servirán para poder pasarle algo de información al Template una vez que lo tengamos construido y así poder evaluar su correcto funcionamiento.

Tienes una copia funcional de ellos en las Notas de esta diapositiva 📌.



TEMPLATE

```
...
props: {...},
template: `

Generamos un template el cual se convertirá en una tarjeta (card). Dentro del mismo agregamos un tag <img> y algunos tags <p>, para poder distribuir correctamente las props que generamos anteriormente.



Sumémosle algunas clases CSS de Bootstrap o personalizadas, para darle un estilo mejorado.



CODER HOUSE


```



TEMPLATE

```
...
props: {...},
template: `

Toda clase CSS que agreguemos a los tags HTML utilizados dentro del template serán los que definan su estilo por sobre cualquier otro estilo o clase CSS que contenga el documento HTML o el tag padre, que termine luego conteniendo a nuestro componente web.



CODER HOUSE


```

INCORPORAR PROPS EN EL TEMPLATE



INCORPORAR PROPS EN EL TEMPLATE

El web component debe recibir tres datos que provienen de las props: **titulo**, **portada** y **costo**. La URL que viene parametrizada en **portada**, será asignada al atributo **src** del tag **img**. En el atributo **title** agregaremos la propiedad **titulo** como también en el primer tag **p**. Finalmente, dentro del tag **strong**, agregamos la propiedad **costo**.

```
template: `<div align="center">
  <div>
    
    <br><br>
    <p class="text-black"> {{ titulo }} </p>
    <p>${<strong> {{ costo }} </strong></p>
  </div>
</div>`
```



INCORPORAR PROPS EN EL TEMPLATE

```

```

Cuando trabajamos con los atributos 🙌 de un elemento HTML integrado en un **template Vue**, debemos referirnos a cada uno de estos, integrando la función **v-bind** para poder escribir correctamente el valor en cada atributo. En nuestro ejemplo la agregamos de forma simplificada, pero también funciona si optamos por escribirla completa 📝.

```

```



INCORPORAR PROPS EN EL TEMPLATE

```
...  
  
  <br><br>  
  <p class="text-black"> {{ titulo }} </p>  
  <p>${<strong> {{ costo }} </strong></p>  
...
```

En el caso de tener que **escribir props** en espacios donde solo va texto, es decir en espacios equivalentes a JavaScript `.textContent` o `.innerText`, simplemente utilizamos los doble moustaches `{{ }}`, tal como lo hicimos en los otros ejemplos que abordamos con Vue.



INCORPORAR PROPS EN EL TEMPLATE

```
<div class="container">  
  <codermeals-card class="card">  
  </codermeals-card>  
</div>
```

Ya estamos en condiciones de incorporar en el documento HTML el componente web desarrollado.

Definimos el mismo y le aplicamos alguna clase CSS que lo estilice, como mencionamos anteriormente.

Definamos a continuación sus atributos.


Aquí, en las Notas, las clases CSS utilizadas en este ejemplo 📌

***DEFINIR LOS VALORES DE SUS
ATRIBUTOS***



DEFINIR LOS VALORES DE SUS ATRIBUTOS

Para estructurar mejor la visualización del componente web y sus atributos se suele escribir una estructura vertical la cual permite una mejor lectura de cada `atributo="valor"`.

En este caso, también debemos configurar cada atributo a través de la directiva `v-bind:` o su versión resumida, tal como vemos aquí. 

```
...  
<codermeals-card class="card"  
  :portada="portada"  
  :titulo="titulo"  
  :costo="costo">  
</codermeals-card>  
...
```



FUNCIONAMIENTO INTERNO DE VUE

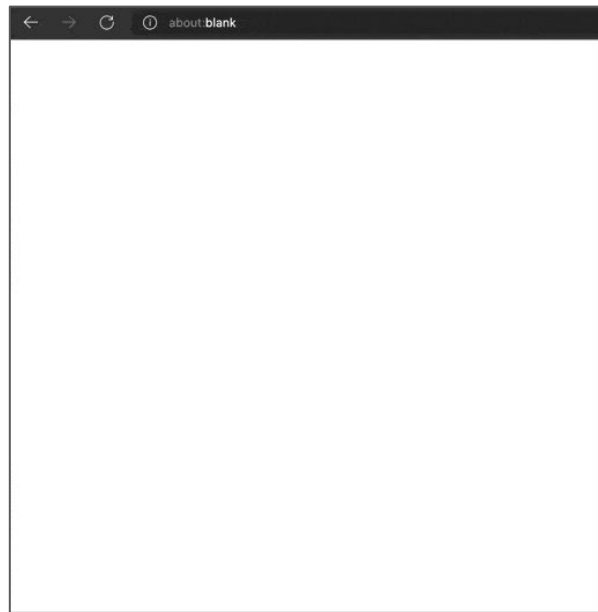
Vue se ocupa de renderizar dicho componente para que pueda ser visualizado en el documento HTML como un `<tag>` más.

A través de este documento, el componente recibe los valores de las propiedades de la instancia **data**, las cuales son tomadas por las **props** del componente, quien finalmente las envía al template para que sean renderizadas correctamente en pantalla.



COMPONENTE WEB EN ACCIÓN

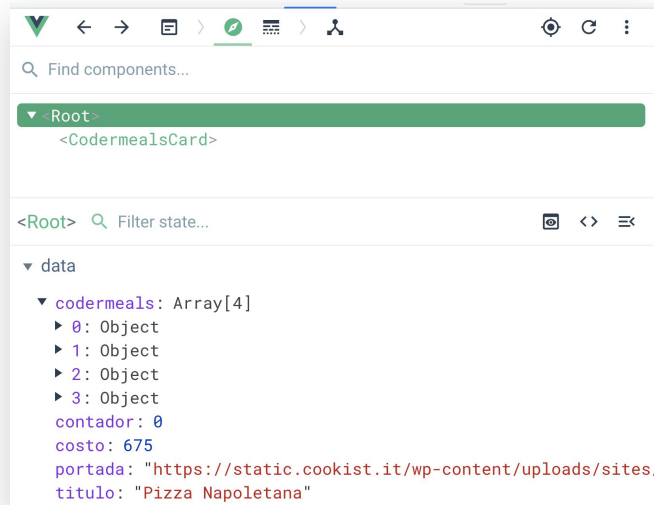
Probemos finalmente nuestro componente web en acción. Si todo salió bien, debemos poder visualizar la Card en pantalla junto al título, costo y foto del producto.





ANALIZAR UN VUE WEB COMPONENT

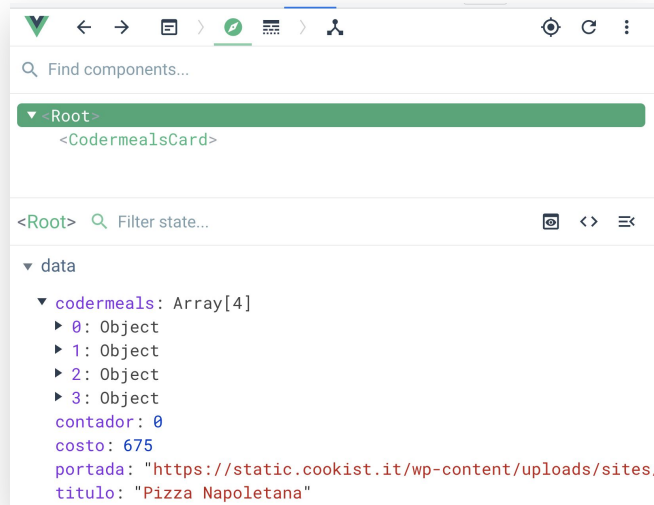
Si todavía no lo hicimos, es un buen momento para instalar en el navegador web donde realizamos las pruebas, la extensión [Vue DevTools](#).





ANALIZAR UN VUE WEB COMPONENT

La misma tiene soporte para **Chrome**, **Edge Chromium** y **Firefox**, y nos permite analizar en detalle un componente web, el DOM de Vue y cualquier propiedad que manejemos en nuestro desarrollo.



REUTILIZAR UN COMPONENTE WEB EN VUE

REUTILIZAR UN COMPONENTE WEB

Cada componente web que creemos cuenta con las mismas características y herramientas que fuimos aprendiendo a lo largo de las primeras clases. Si, por ejemplo, quisiéramos reutilizar este componente creando tarjetas (*Cards*) a partir de la iteración de un array de objetos, podríamos realizarlo sin problema.

👁️ Veamos a continuación cómo integrar un objeto JSON con varios elementos, reutilizando nuestro componente web, mediante la directiva **v-for**.

Array JSON de múltiples elementos a utilizar en el siguiente ejemplo 📌.



REUTILIZAR UN COMPONENTE WEB

```
app.js > [app] > [data] > [codermeals]

var app = new Vue({
  el: '#app',
  data: {
    codermeals: [
      {
        id: 1,
        titulo: "Spaghetti alla puttanesca",
        costo: 575.00,
        portada: "https://4.bp.blogspot.com/-D5Wvi_gX_Kg/w"
      },
      {
        id: 2,
        titulo: "Pizza Napoletana ai carciofi",
        costo: 675.00,
        portada: "https://static.cookist.it/wp-content/upl"
      },
      {
        id: 3,
        titulo: "Porchetta umbra a cottura lunga",
        costo: 845.00,
        portada: "https://www.fontecesia.it/wp-content/upl"
      },
      {
        id: 4,
        titulo: "Orecchiette alle cime di rapa",
        costo: 845.00,
        portada: "https://irepo.primecp.com/2016/03/259860/rec"
      }
    ]
  }
})
```

Si integramos en el objeto **data** de la instancia principal el array JSON con *N* cantidad de elementos, podremos representar a todos ellos en una Vista, reutilizando nuestro Vue web component.



REUTILIZAR UN COMPONENTE WEB

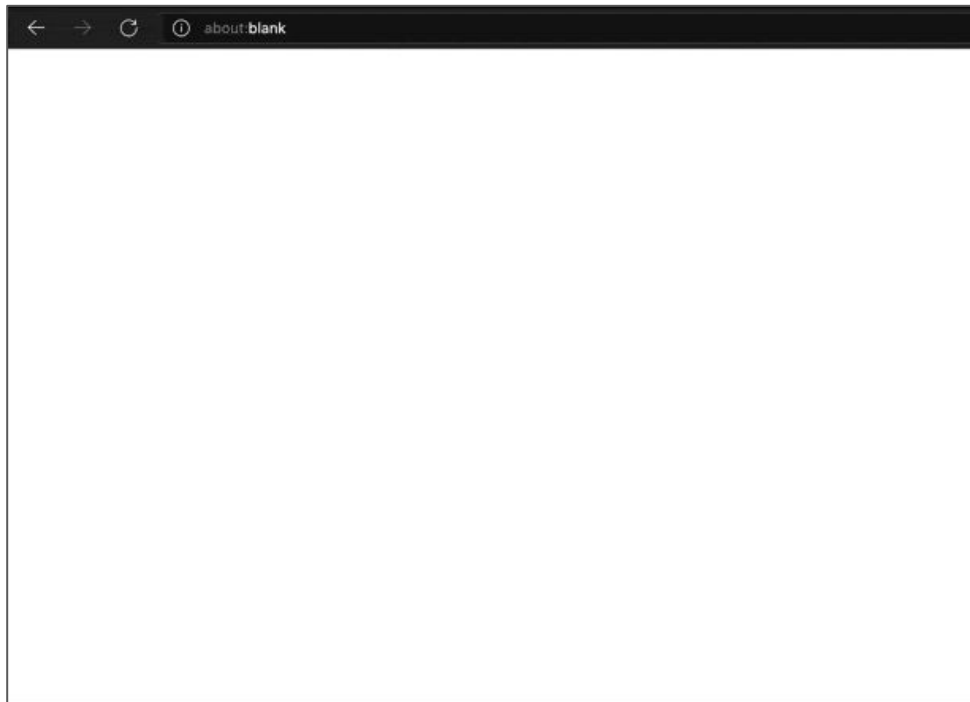
```
<codermeals-card class="card"
  v-for="(plato, index) in codermeals"
  :key="plato.id"
  :portada="plato.portada"
  :titulo="plato.titulo"
  :costo="plato.costo">
</codermeals-card>
```

Agreguemos en éste la directiva **v-for**, para poder iterar sobre el array JSON.

Modificamos cada atributo del componente para que reciba el valor de la propiedad homónima del array a medida que lo iteramos.



REUTILIZAR UN COMPONENTE WEB



Con este pequeño cambio, ya podemos disfrutar de **nuestro componente web creado con Vue**, reutilizado de forma efectiva dentro de nuestro proyecto.



MODIFICAR UN COMPONENTE WEB



MODIFICAR UN COMPONENTE WEB

```
template: `

Si deseamos potenciar la funcionalidad de nuestro componente web, podemos modificar el template original. En este ejemplo 🙌 sumamos un elemento HTML <button>. Nos restará aplicarle la lógica a través de métodos o propiedades computadas para que su funcionalidad sea efectiva.

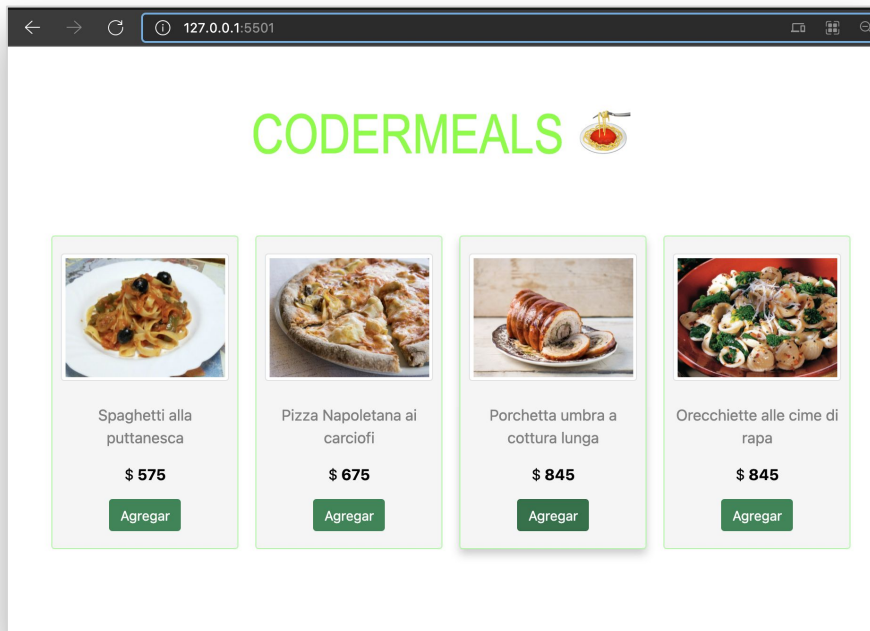


CODER HOUSE


```



MODIFICAR UN COMPONENTE WEB



Nuestro componente web modificado
cuenta ahora con una nueva
funcionalidad, casi sin esfuerzo
alguno.

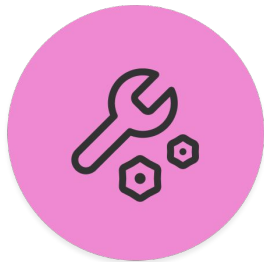
Así es como Vue JS nos facilita la
tarea de crear y mantener web
components.



FUNCIONAMIENTO INTERNO DE VUE

Vue renderiza dicho componente para que pueda ser visualizado en el documento HTML como un `<tag>` más.

El componente recibe, mediante HTML, los valores en las propiedades de la instancia **data**. Estos son tomados por las **props** del componente, quien finalmente las envía al template para que finalmente se rendericen en pantalla.



COMPONENTE INSTANCIADO CON PROPS

Instancia tu componente para que reciba props.

Tiempo estimado: 15 minutos.

CODER HOUSE

COMPONENTE INSTANCIADO CON PROPS

Desafío
generico



De acuerdo a lo último que repasamos en materia de web components, agrega props a tu componente anterior y realiza los cambios necesarios en tu proyecto para que el mismo pueda instanciarse varias veces en pantalla, mostrando diferentes valores obtenidos a través de un array de datos.

Tiempo estimado: 15 minutos.

CODER HOUSE



PROYECTO VUE WEB COMPONENTS

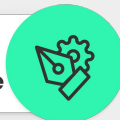
Desarrolla un proyecto VUE CDN con componentes web interactivos a partir de diferentes fuentes de entrada.

PROYECTO VUE WEB COMPONENTS

Formato: Utiliza el nombre “Proyecto + Apellido”.

Sugerencia: N/A

Desafío
entregable



>> Consigna:

1. Crear un componente web que reciba parámetros de diferentes tipos.
2. Instanciar al menos tres veces dicho componente, mostrando en cada instancia diferente información.

>>Aspectos a incluir en el entregable:

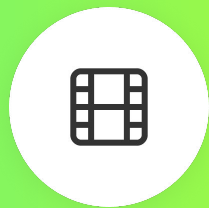
Desarrolla un proyecto en Vue CDN que integre componentes web, los cuales deben recibir props con diferentes tipos de datos, entre ellos un array de objetos, para finalmente representarlos en una vista.

Dicha vista debe estar constituida por filas (<tr>, <td>) de un elemento <table>.

Entre los parámetros que debes pasar mediante props incluye estilos o clases CSS, que permitan cambiar el color de la letra y fondo del elemento <table>. El componente web que desarrolles integrando una tabla debe tener al menos tres filas con diferentes datos por cada una de ellas.

Sube el proyecto a la plataforma de Coderhouse para su posterior corrección.

CODER HOUSE



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



- [Convención de nombres en programación](#) | **Wikipedia**
- [Vue Devtools](#) | **Vue JS**

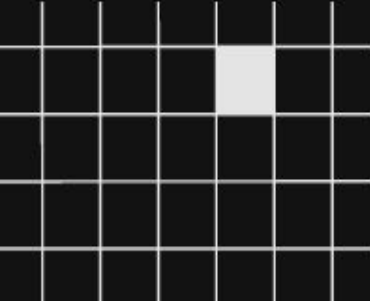
¿PREGUNTAS?





¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Vue CDN web components
 - Props
 - Templates
- 



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN

CODER HOUSE