

Descripción momento práctico

Momento Práctico. En esta sesión los estudiantes practicarán con componentes reutilizables que rendericen data de fuentes externas o API con las indicaciones dadas por el formador.

Para empezar, lo único que tendremos que hacer es incluir la dependencia de VueJS a nuestro proyecto de catalogo de productos. Dependiendo de nuestras necesidades, podremos hacer esto de varias maneras. Lo primero que hacemos es ejecutar los siguientes comandos en el terminal:

```
$ mkdir mi_primer_proyecto_vue
```

```
$ cd mi_primer_proyecto_vue
```

```
$ npm init
```

Esto nos genera una nueva carpeta para el proyecto y nos inicia un paquete de NodeJS. Una vez que tengamos esto, añadiremos VueJS como dependencia de la siguiente manera:

```
$ npm install vue --save
```

De esta forma, ya tendremos todo lo necesario para trabajar en nuestro primer proyecto. Lo que esta dependencia nos descarga son diferentes VueJS dependiendo del entorno que necesitemos. Lo que hacemos ahora es añadir un fichero **index.html**, **app.js** y **style.css** para nuestros estilos css.

Para recordar lo de sesión anterior, En VueJS, los componentes son una forma de crear instancias de VueJS personalizadas que se pueden reutilizar fácilmente en su código. Para explicar adecuadamente qué son los componentes de VueJS, crearemos un componente muy simple **artículos**

En su editor de texto en su computadora, en su archivo **index.html** actualice pegando el contenido de este fragmento:

```
<!doctype html>
<html>
<head>
  <title>Mi primer proyecto</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div id="aplicacion">
  </div>
  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```



```
<script src="node_modules/vue/dist/vue.js"></script>
<script src="app.js"></script>
</body>
</html>
```

Como esta será una aplicación VueJS simple, lo primero que tenemos que hacer es incluir la biblioteca VueJS en nuestro documento. Como se puede observar en la línea 16.

Después de que se incluye la biblioteca VueJS, se crea un div con id establecido en **aplicacion**. En nuestro archivo app.js, se crea una aplicación VueJS básica y se vincula a este elemento.

```
new Vue({
  el: '#aplicacion',
});
```

Hasta ahora, esta nueva aplicación no hace nada en absoluto, pero la usaremos para el esqueleto de nuestra aplicación.

Creando su primer componente

El componente que desarrollaremos es un simple catálogo de productos reutilizable y se ilustrará cómo funcionan los componentes de VueJS. Explicaremos cada parte del componente a lo largo del camino. Empecemos:

En su app.js, actualice sección para incluir la Vue.component() función de este fragmento:

```
Vue.component('articulos', {
  props: ['datos'],
  template: `
    <div>
      <h1>Ejemplo vue js componentes</h1>
      <div v-for="art in datos" class="card">
        
        <p class="price">{{art.price}}</p>
        <p>{{art.name}}</p>
        <p><button>COMPRAR</button></p>
      </div>
    </div>
  `
});

new Vue({
  el: '#aplicacion',
```



```
});
```

El primer parámetro de la `Vue.component()` función es el nombre del componente, en este caso **'articulos'**. Se referirá a este componente por su nombre para renderizarlo (descrito en la siguiente sección: Usar el componente).

El segundo argumento de la `Vue.component()` función son las opciones del componente. Solo usamos dos opciones: la función de datos y `template`.

Los datos del componente deben ser una función. Esto se debe a que cada instancia del componente debe tener una copia separada e independiente del objeto de datos. De lo contrario, cada vez que reutilizamos un componente, heredará los datos de las otras instancias del componente.

La plantilla contiene el HTML que este componente representará:

VueJS usa etiquetas de bigote para representar cadenas. En nuestro ejemplo, `{{ art }}` representará el recuento de los datos del componente.

Lo que es interesante es que como VueJS es reactivo, cuando cambia su variable, la vista la actualiza automáticamente y no tiene que agregar ningún código para que esto funcione.

Otra cosa que probablemente notó en la plantilla. La directiva **v-for** se utiliza para representar nuestra lista de resultados. También utilizamos llaves dobles para mostrar el contenido de cada una del json, traído desde una api de terceros en este caso desde la siguiente url: <https://my-json-server.typicode.com/jubs16/Products/Products> (que más adelante se explicará).

En este punto, hemos creado nuestro primer componente, pero aún no estará visible si lo carga `index.html` en su navegador.

Ahora actualizaremos nuestro archivo `app.js` con el siguiente fragmento de código:

```
Vue.component('articulos', {
  props: ['datos'],
  template: `
    <div >
      <h1>Ejemplo vue js componentes</h1>
      <div v-for="art in datos" class="card">
        
        <p class="price">{{art.price}}</p>
        <p>{{art.name}}</p>
      </div>
    </div>`
});
```



```
        <p><button>COMPRAR</button></p>
      </div>
    </div>
  },
});

new Vue({
  el: '#aplicacion',
  data() {
    return {
      datos: null
    };
  },
  mounted() {
    axios
      .get('https://my-json-
server.typicode.com/jubs16/Products/Products')
      .then(response => (this.datos = response.data));
  }
});
```

La **data** de la aplicación principal se establecen. La aplicación ahora realiza un seguimiento de un objeto llamado **datos**.

En nuestro props componente agregamos **datos**.

Para obtener la lista de artículos desde el api de terceros utilizamos el hook **mounted**, que se explicó en la sesión anterior, aquí realizamos las solicitudes de Ajax y manejo de respuestas con axios.

Axios es un cliente HTTP basado en la promesa para realizar solicitudes de Ajax, y funcionará muy bien para nuestros propósitos. Proporciona una API simple y rica. Es bastante similar a la fetch API pero sin la necesidad de agregar un polyfill para navegadores antiguos, y algunas otras sutilezas.

El resultado de esta solicitud se la asignamos al objeto de nuestro data, datos, como se ve en la imagen anterior, (**this.datos = response.data**);

El siguiente paso es actualizar nuestro index.html como el siguiente fragmento:

```
<!doctype html>
<html>

<head>
  <title>Prueba Vue</title>
  <meta charset="utf-8">
```



```
<meta name="viewport" content="width=device-width, initial-
scale=1, shrink-to-fit=no">
<link rel="stylesheet" href="styles.css">
</head>

<body>
  <div id="aplicacion">
    <articulos v-bind:datos="datos"></articulos>
  </div>
  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
  <script src="node_modules/vue/dist/vue.js"></script>
  <script src="app.js"></script>
</body>

</html>
```

Se puede observar que a nuestro div con el id="aplicación" le agregamos el componente creado anteriormente .

el valor del **prop** se asigna con el **v-bind:datos**. Al usar la v-bind sintaxis , le está diciendo a VueJS que vincule el datos del componente a cualquier propiedad de datos que proporcione, en este caso el datos de nuestra instancia vue de nuestro archivo app.js.


actualizamos nuestro navegador para observar los cambios y tendremos el siguiente resultado:



El futuro digital
es de todos

MinTIC


Mi Primer componente reutilizable

 Denim Jeans

110

Sport Shoes


COMPRAR

 Denim Jeans

91

Red SNEAKER


COMPRAR

 Denim Jeans

94

Sport Shoes Women

COMPRAR

 Denim Jeans

143

Winter boots children

COMPRAR

Para darle estilo a nuestro catálogo de artículos actualizamos nuestro archivo style.css con el siguiente fragmento de código :

```
.card {  
  box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2);  
  max-width: 300px;  
  margin: auto;  
  text-align: center;  
  font-family: arial;  
}
```



```
.price {  
  color: grey;  
  font-size: 22px;  
}  
  
.card button {  
  border: none;  
  outline: 0;  
  padding: 12px;  
  color: white;  
  background-color: #000;  
  text-align: center;  
  cursor: pointer;  
  width: 100%;  
  font-size: 18px;  
}  
  
.card button:hover {  
  opacity: 0.7;  
}
```

Obtenemos como resultado:

Mi Primer componente reutilizable

