Vue js Primeros pasos

Resumen en base a https://vuejs.org/

Vue js Características

Vue (pronunciado /vjuː/, como vista) es un marco de JavaScript para construir interfaces de usuario. Se basa en HTML, CSS y JavaScript estándar y proporciona un modelo de programación declarativo y basado en componentes que lo ayuda a desarrollar interfaces de usuario de manera eficiente, ya sean simples o complejas.

Dos características principales de Vue

- Declarative Rendering (representación declarativa): Vue amplía el HTML estándar con una sintaxis de plantilla que nos permite describir de forma declarativa la salida HTML en función del estado de JavaScript.

- Reactivity (Reactividad): Vue rastrea automáticamente los cambios de estado de JavaScript y actualiza de manera eficiente el DOM cuando ocurren cambios.

Single-File Components

En la mayoría de los proyectos de Vue habilitados para herramientas de compilación, creamos componentes de Vue utilizando un formato de archivo similar a HTML llamado componente de archivo único (también conocido como archivos *.vue, abreviado como SFC). Un SFC de Vue, como sugiere el nombre, encapsula la lógica del componente (JavaScript), la plantilla (HTML) y los estilos (CSS) en un solo archivo.

Esto lo vamos a aplicar cuando veamos componentes.

Options API

Con la API de opciones, definimos la lógica de un componente usando un objeto de opciones como datos, métodos y montado. Las propiedades definidas por las opciones se exponen en estas funciones internas, que apuntan a la instancia del componente.

Vue js - Uso con CDN

```
<!DOCTYPE html>
<body>
    <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
    <div id="app">{{ mensaje }}</div>
    <script>
        const { createApp } = Vue
        createApp({
            data() {
                return {
                    mensaje: 'Hola Vue!'
                }
            }
        }).mount('#app')
    </script>
</body>
</html>
```

Style Bindings

Ejemplo de como enlazar un estilo al objeto de vuejs:

```
<div id="app">
        <p style="{ color: color, fontSize: fontSize + 'px' }" >Texto</p>
    </div>
    <script>
        const { createApp } = Vue
        const obj = createApp({
            data() {
                return {
                    color: 'blue',
                    fontSize: 30
                }
            },
        }).mount('#app')
    </script>
```

Conditional Rendering. Directiva v-if. Ejemplo

```
<div id="app">
        <p v-if="verParrafo" >Algun parrafo</p>
    </div>
    <script>
        const { createApp } = Vue
        const obj = createApp({
            data() {
                return {
                    verParrafo: true
                }
            }
        }).mount('#app')
    </script>
```

Conditional Rendering. Directiva v-else

```
<div id="app">
        <p v-if="verParrafo" >Algun parrafo</p>
        <p v-else="verParrafo">Este parrafo se ve, si v-else da falso</p>
    </div>
    <script>
        const { createApp } = Vue
        const obj = createApp({
            data() {
                return {
                    verParrafo: true
                }
            }
        }).mount('#app')
    </script>
```

Conditional Rendering. Directiva v-show (oculta elemento)

```
<div id="app">
        <p v-show="verParrafo" >Algun parrafo</p>
    </div>
    <script>
        const { createApp } = Vue
        const obj = createApp({
            data() {
                return {
                    verParrafo: true
                }
            }
        }).mount('#app')
    </script>
```

List Rendering. Directiva v-for. Ejemplo

```
<div id="app">
        <li v-for="producto in productos">
            {{ producto.id }} {{ producto.desc }}
        </li>
    </div>
    <script>
        const { createApp } = Vue
        const obj = createApp({
            data() {
                return {
                    productos: [{id:10,desc:'Cocina'},{id:15,desc:'Silla'}]
                }
            }
        }).mount('#app')
    </script>
```

Detección de cambios en array

Vue puede detectar cuándo se llama a los métodos de mutación de una matriz reactiva y desencadenar las actualizaciones necesarias. Estos métodos de mutación son:

push()

pop()

shift()

unshift()

splice()

sort()

reverse()

Detección de cambios en array

Los métodos de mutación, como sugiere el nombre, mutan la matriz original a la que se les llama. En comparación, también hay métodos que no mutan, p. filter(), concat() y slice(), que no mutan la matriz original pero siempre devuelven una nueva matriz.

this.items = this.items.filter((item) => item.message.match(/Foo/))

Event Handling (Manejando eventos)

Una forma es asociar el evento click de un botón a un método del objeto de vue.

```
<div id="app">
        {{ mensaje }}
        <button v-on:click="saludar">Saludar</button>
    </div>
    <script>
        const { createApp } = Vue
        const obj = createApp({
            data() {
                return {
                    mensaje: ''
                }
            },
            methods: {
                saludar() { this.mensaje = 'Hola pnt2 !'}
            }
        }).mount('#app')
    </script>
```

Form Input Bindings (Enlaces de entrada de formulario)

Para enlazar, usamos la sentencia v-model

```
<div id="app">
        {{ mensaje }}
        <button v-on:click="saludar">Saludar</button>
        <input v-model="mensaje">
    </div>
    <script>
        const { createApp } = Vue
        const obj = createApp({
            data() {
                return {
                    mensaje: ''
                }
            },
            methods: {
                saludar() { this.mensaje = 'Hola pnt2 !'}
            }
        }).mount('#app')
    </script>
```

Ejercicio 1

Dada esta lista de numeros: [4,6,3,2,-3,2]

1- Cargar esta lista en un array del objeto vue.

2- Mostrar la lista con un li

3- Pedir un numero y agregarlo a la lista.

Ejercicio 2

Dado una lista de clientes, con id, nombre y edad:

1- Cargar la lista de clientes en el objeto vue y mostrar la lista

2- Pedir los datos de un cliente y con un boton, agregarlo a la lista.

3- Bonus I : agregar un botón por cada elemento de la lista para permitir eliminar

4- Bonus II: agregar un botón por cada elemento para permitir modificar.