VUE

Características

* Framework JS, frontend de fácil aprendizaje
* Reactivo
* Basado patrón en model-view-modelview

Interpolación {{ }}

<!DOCTYPE html>

<html>

<head>

<title>Mi primera aplicaci&oacute;n Vue</title>

<script src="https://unpkg.com/vue@next"></script>

</head>

<body>

<div id="app">

<h2>{{ message }}</h2>

</div>

<script>

const HelloVueApp = {

data() {

return {

message: 'Hola Vue!!!'

}

}

}

vue\_app = Vue.createApp(HelloVueApp).mount('#app')

</script>

</body>

</html>

**DIRECTIVAS**

**v-bind**: para bindear un atributo de un tag html con datos de vue. Así le podés agregar clases, atributos

<div id="app">

<span v-bind:title="message">

Hover your mouse over me !!

</span>

</div>

<script>

const App = {

data() {

return {

message: 'Fecha ' + new Date().toLocaleString()

}

}

}

vue\_app = Vue.createApp(App).mount('#app')

</script>

------------

<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>

…

data: {

activeColor: 'red',

fontSize: 30

}

**v-text**

Para mostrar contenido adentro de un elemento. Actualiza el textContent del elemento

<h1 v-text="name"></h1>

Es como hacer <h1>{{ name }}</h1>

Previene XSS

**v-html**

Para mostrar contenido adentro de un elemento. Actualiza el innerHTML. Es débil a ataques XSS renderiza el html en la página, y si eso tiene un script adentro cagaste

**v-on**

Para bindear un elemento del DOM con un manejador(método) cuando se produzca un evento

<div id="app">

<p>{{ message }}</p>

<button v-on:click="reverseMessage">Reverse Message</button>

</div>

<script>

const App = {

data() {

return {

message: 'Bienvenidos a Proyecto de Desarrollo!!'

}

},

methods: {

reverseMessage: function () {

this.message = this.message.split('').reverse().join('')

}

}

}

vue\_app = Vue.createApp(App).mount('#app')

</script>

**v-model**

Es para bindear el input de un html con una variable de vue

<div id="two-way-binding">

<p>{{ message }}</p>

<input v-model="message">

</div>

<script>

const App = {

data() {

return {

message: 'Bienvenidos a Proyecto!!'

}

}

}

vue\_app = Vue.createApp(App).mount('#two-way-binding')

</script

**Bucles: v-for.**

<div id="app">

<ul>

<li v-for="product in products" :key:”producto”>

{{ product }}

</li>

</ul>

</div>

<script>

const App = {

data() {

return {

products: [

'Harina',

'Arroz',

'Yerba'

]

}

}

}

vue\_app = Vue.createApp(App).mount('#app')

</script>

**Condicionales: v-if, v-else, v-else-if.**

<div id="app">

<p>

<span v-if="seen">Now you see me</span>

<span v-else>Oh no 😢</span>

</p>

</div>

<script>

const App = {

data() {

return {

seen: true

}

}

}

vue\_app = Vue.createApp(App).mount('#app')

</script>

**v-show**

Esta relacionado con la propiedad display de CSS, que es para mostrar o no un elemento

El v-if renderiza lazy, solo te renderiza la partecita de la condición que se cumple, y el v-show no, renderiza todo de entrada y después muestra o no en pantalla según condición

<div id="app">

<p>

<span v-show="seen">Now you see me</span>

</p>

</div>

<script>

const App = {

data() {

return {

seen: true 🡪 cuando cambie a false, le pondría display: none

}

}

}

vue\_app = Vue.createApp(App).mount('#app')

</script>

**PROPIEDADES COMPUTADAS**

Es para no poner código de cálculo o cosas así en el template

<div id="app">

<ul>

<li v-for="product in products">

{{ product }}

</li>

</ul>

Cantidad de elementos: {{ countProducts }}

</div>

<script>

const App = {

data() {

return {

products: [

'Harina',

'Arroz',

'Yerba'

]

}

},

computed: {

// a computed getter

countProducts: function () {

return this.products.length

}

}

}

vue\_app = Vue.createApp(App).mount('#app')

</script>

También si quisieras hacerle algo a un string.. parsear, convertirlo, algo así

La diferencia con los métodos normales es que están cacheadas y solo se re-evalúa si cambia la dependencia (o sea el products por ej)

El método se ejecutasiempre, cada vez que se llama

**WATCHER**

Para reaccionar cuando un dato cambia (sea un objeto interno, o un input ponele). Es como si fuera un manejador de un evento, es para operaciones asíncronas o costosas. Por ej si tenés que consultar una api o algo así y querés ir mostrando en tiempo real algún mensaje o cambio según lo que ponga el user en un input

<div id="app">

<input placeholder="Escribe tu nombre" v-model="nombre" type="text" name="nombre" id="nombre"/>

</div>

const app = new Vue({

el: "#app",

data: () => ({

nombre: "",

}),

watch: {

nombre(nuevoValor, valorAnterior){

console.log("El nombre pasó de ser '%s' a '%s'", valorAnterior, nuevoValor);

}

}

});

data: () => ({

nombre: "",

}),

watch: {

nombre(nuevoValor, valorAnterior){

console.log("El nombre pasó de ser %s a %s", valorAnterior, nuevoValor);

}

}

**CREATE VS MOUNTED**

the created method will be called before component template is made. so you can not access template but you can change values that should be used in the template. for example you can convert json props to object or change component static data and ... The only difference 'created' with 'mounted' is that you can perform performance before the template is made

**COMPONENTES**

Los componentes son instancias reutilizables de Vue con un nombre

Tenés el componente Raiz, y a partir de ese se ponen todos los demás, como un árbol de componentes anidados

Data tiene que ser una función para que cada instancia de un componente tenga una copia distinta del objeto que tiene adentro el data

**REGISTRO GLOBAL**

Vue.component(‘component-a', {

// ... opciones ...

})

new Vue({ el: '#app' })

----------------------------------

<div id="app">

<component-a></component-a>

</div>

Se puede registrar escribiendo-asi o EscribiendoAsi

**REGISTRO LOCAL**

Los declarás como objetos JS simples, importás el archivito si está en otro lado, y lo declarás en el components: {} del componente padre

Sin webpack:

var ComponentA = { /\* ... \*/ }

new Vue({

el: '#app',

components: {

'component-a': ComponentA,

}

})

Con webpack:

import ComponentA from './ComponentA.vue'

export default {

components: {

ComponentA

},

// ...

}

Estructura componente base base

<template> </template>

<script>

export default {};

</script>

COMUNICACIÓN

Padre->Hijo PROPS

Hijo->Padre EVENTOS

Webpack -> Para resolver

Templates como strings: no poseemos syntax highlight en el desarrollo.

No tenemos soporte para CSS.

No hay etapa de contrucción: nos restringe a utilizar puramente HTML y JavaScript.

**VUE-ROUTER**

npm install vue-router o vue add router con vue-cli

path: El path relativo a la base de la aplicación.

name: El nombre de la ruta para referenciarla en los componentes.

component: El componente que va a estar en esa ruta.

redirect: Una redirección.

alias: Alias.

children: Un arreglo con mas rutas que se concatenan a la ruta padre.

params: Parámetros del componente.

<template>

<div id="app">

<div id="nav">

<router-link to="/">Home</router-link> |

<router-link to="/ruta1">Ruta 1</router-link> |

<router-link to="/ruta2">Ruta 2</router-link> |

<router-link to="/about">About</router-link>

</div>

<router-view/>

</div>

</template>

VUEX

librería para manejar un estado global para aplicaciones Vue.js

**VUE-CLI**

Es una interfaz de línea de comandos de Vue para ayudar con la creación del scaffolding (o sea la base) y la configuración inicial del proyecto.

Instalación

npm install -g @vue/cli

Crear el proyecto base

vue create my-project

Correrlo

npm run serve

Tiene una ui, que la corres poniendo vue ui, para gestionar los proyectos vue que tengas

vue add <algo> para que agregue plugins