Primeros pasos con VueJS

Índice

0.- Introducción..... .1
1.- La gran mejora en la renderización.....1
1.1.- HTML + CSS + JS (Puro)..... .1
1.2.- Vue.js.....2
1.3.- Comparativa práctica.....2
2.- API Styles.....2
2.1.- Options API..... .2
2.2.- Composition API..... .3
2.3.- ¿Cuál usar?..... .4
3.- Usando Vue..... .4
3.1.- Desde CDN.....4
3.1.1.- Usando la Compilación Global.....5
3.1.2.- Usando el Módulo de Construcción ES.....5
3.2.- Aplicación de Vue.....6
4.- Soporte para Visual Studio Code..... .6
5.- Componentes de archivo único (SFC)..... .6
5.1.- Proyecto Composition API sin formato SFC.....7
5.3.- Proyecto Composition API con formato SFC..... .8
ANEXO I: Ejemplos de renderizado.....10

0.- Introducción

Vue JS es un framework progresivo para crear webs Single-Page-Application (S.P.A.) Las webs S.P.A son webs de una sola página que no se recargan, los componentes van variando, van cambiando pero la web nunca se recarga.

Puedo declarar rutas en mi web. Que aunque sea una sola página yo podré hacer navegaciones a otras páginas dentro de nuestro sitio.

Lo de framework progresivo se refiere a que puede servir para proyectos pequeños, medianos o muy grandes. Podremos ir escalando sin problema.

1.- La gran mejora en la renderización

Vamos a comparar el efecto de renderizado de una página web hecha con HTML, CSS y JavaScript puro frente a una construida con Vue.js puede mostrar diferencias interesantes en términos de rendimiento, actualización del DOM y organización de la estructura del proyecto. Aquí tienes algunos puntos a considerar y un esquema para crear una comparativa:

1.1.- HTML + CSS + JS (Puro)

Ventajas: Control total sobre el DOM, sin necesidad de una biblioteca externa.

Desventajas: Cuando se trata de actualizaciones complejas del DOM, el código puede volverse más complicado y menos eficiente.

Ejemplo: Un archivo simple donde todos los scripts están directamente relacionados con eventos del DOM y actualizan los elementos en tiempo real.

1.2.- Vue.js

Ventajas: Reactividad fácil de gestionar, separación de responsabilidades con componentes y mejor rendimiento en la actualización del DOM gracias al Virtual DOM.

Desventajas: Mayor complejidad al principio, ya que es necesario entender la estructura y sintaxis de Vue.js.

Ejemplo: La misma página renderizada como componentes, con un flujo de datos reactivo.

1.3.- Comparativa práctica

Escenario de Prueba:

Creamos una página sencilla que muestre una lista de elementos. Los elementos pueden ser actualizados dinámicamente, como agregar o eliminar artículos de la lista.

Agregar algunas animaciones simples o cambios de estilo al actualizar la lista.

Los códigos de los ejemplos puedes verlos en el “ANEXO I: Ejemplos de renderizado”.

2.- API Styles

Los componentes de Vue se pueden crear con dos tipos de API diferentes: **Options API** y **Composition API**.

2.1.- Options API

Con la Options API, **definimos la lógica de un componente usando un objeto de opciones tales como data, methods y mounted**. Las propiedades definidas por las opciones se exponen en el this dentro de las funciones, el cual apunta a la instancia del componente:

```
<script>
        export default {
                // Las propiedades retornadas desde data() pasan a un estado
                // reactivo y serán mostradas en el `this`.
                data() {
                  return {
                    count: 0
                  }
                },

                // Los métodos son funciones que mutan el estado y disparan
                // actualizaciones. Estos pueden ser ligados como manejadores de
```

```
        // eventos en las plantillas.
        methods: {
          increment() {
            this.count++
          }
        },

        // Los hooks del ciclos de vida son llamados en diferentes
        // etapas del ciclo de vida del componente.
        // Esta función será llamada cuando el componente sea montado.
        mounted() {
          console.log(`El conteo inicial es ${this.count}.`)
        }
    }
</script>

<template>
  <button @click="increment">El conteo es de: {{ count }}</button>
</template>
```

2.2.- Composition API

Con la Composition API, **definimos la lógica de un componente utilizando funciones importadas de la API**. En un SFC, la Composition API **se usa normalmente con <script setup>**.

El atributo setup es una indicación que hace que Vue realice transformaciones en tiempo de compilación, lo que nos permiten usar la Composition API con menos repeticiones. Por ejemplo, las importaciones y las variables/funciones de nivel superior declaradas en un <script setup> se pueden usar directamente en la plantilla.

Aquí está el mismo componente, con exactamente la misma plantilla, pero usando la Composition API y <script setup> en su lugar:

```
<script setup>
    import { ref, onMounted } from 'vue'

    // estado reactivo
    const count = ref(0)

    // funciones que mutan el estado y disparan actualizaciones
    function increment() {
      count.value++
    }

    // hooks del ciclo de vida
    onMounted(() => {
      console.log(`El conteo inicial es ${count.value}.`)
    })
</script>
```

```
<template>
  <button @click="increment">El conteo es de: {{ count }}</button>
</template>
```

2.3.- ¿Cuál usar?

Ambos estilos de API son totalmente capaces de cubrir casos de uso comunes. De hecho, ¡la Options API es implementada sobre la Composition API! Los conceptos y conocimientos fundamentales sobre Vue se comparten entre los dos estilos.

La **Options API** se centra en el concepto de una "instancia de componente" (this como viste en el ejemplo), que generalmente se alinea mejor con un modelo mental basado en clases para usuarios que provienen de entornos de lenguajes de POO. Mejor opción para iniciarse.

La **Composition API** se centra en la **declaración de variables de estado reactivas** directamente en el ámbito de una función y en la composición del estado de múltiples funciones para manejar juntas la complejidad. Tiene una forma más libre y **requiere una comprensión de cómo funciona la reactividad en Vue** para ser utilizada de manera efectiva. A cambio, su flexibilidad permite patrones más poderosos para organizar y reutilizar la lógica.

Recomendaciones:

Si estás aprendiendo Vue, elige el estilo que te resulte más fácil. Los conceptos básicos son compartidos, y puedes cambiar de estilo cuando lo desees.

Para proyectos en producción:

- Usa la Options API si no necesitas herramientas de compilación o trabajas en proyectos de baja complejidad.
- Prefiere la Composition API y Componentes de un Solo Archivo si desarrollas aplicaciones completas.

3.- Usando Vue

3.1.- Desde CDN

Puedes usar Vue directamente desde un CDN a través de una etiqueta script:

```
<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
```

Aquí estamos usando unpkg, pero también puedes usar cualquier CDN que sirva paquetes npm, por ejemplo jsdelivr o cdnjs. Por supuesto, **también puedes descargar este archivo y servirlo tú mismo.**

Cuando se utiliza Vue desde un CDN, no hay ningún "paso de compilación" involucrado. Esto hace que la configuración sea mucho más simple, y es adecuado para mejorar el HTML estático o la integración con un marco de backend. Sin embargo, n**o podrás usar la sintaxis de Componentes de un Solo Archivo (SFC).**

3.1.1.- Usando la Compilación Global

El enlace anterior carga la compilación global de Vue, donde todas las APIs de alto nivel están expuestas como propiedades en el objeto Vue global. Aquí hay un ejemplo completo usando la compilación global:

```
<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
<div id="app">{{ message }}</div>
<script>
  const { createApp, ref } = Vue
  createApp({
    setup() {
      const message = ref('¡Hola Vue!')
      return {
        message
      }
    }
  }).mount('#app')
</script>
```

3.1.2.- Usando el Módulo de Construcción ES

La mayoría de los navegadores modernos soportan ahora módulos ES de forma nativa, por lo que podemos usar Vue desde un CDN a través de módulos ES nativos como este:

```
<script type="importmap">
  {
    "imports": {
      "vue": "https://unpkg.com/vue@3/dist/vue.esm-browser.js"
    }
  }
</script>
<div id="app">{{ message }}</div>
<script type="module">
  import { createApp, ref } from 'https://unpkg.com/vue@3/dist/vue.esm-browser.js'
  createApp({
    setup() {
      const message = ref('¡Hola Vue!')
      return {
        message
      }
    }
  }).mount('#app')
</script>
```

Con <script type="importmap"> enseñamos al navegador desde dónde localizar la importación de las librerías.

3.2.- Aplicación de Vue

Asegúrate de tener instalada una versión actualizada de Node.js y que tu directorio de trabajo actual sea donde quieras crear un proyecto.

npm create vue@latest

Este comando instalará y ejecutará create-vue, la herramienta oficial de estructuración de proyectos de Vue. Se te presentarán solicitudes para varias características opcionales como TypeScript y soporte de pruebas:

```
✓ Project name: … <your-project-name>
✓ Add TypeScript? … No / Yes
✓ Add JSX Support? … No / Yes
✓ Add Vue Router for Single Page Application development? … No / Yes
✓ Add Pinia for state management? … No / Yes
✓ Add Vitest for Unit testing? … No / Yes
✓ Add an End-to-End Testing Solution? … No / Cypress / Playwright
✓ Add ESLint for code quality? … No / Yes
✓ Add Prettier for code formatting? … No / Yes

Scaffolding project in ./<your-project-name>...
Done.
```

Una vez creado el proyecto, sigue las instrucciones para instalar las dependencias e iniciar el servidor de desarrollo:

```
$ cd <your-project-name>
$ npm install
$ npm run dev
```

Ahora deberías tener tu primer proyecto Vue funcionando. Ten en cuenta que los componentes de ejemplo del proyecto generado están escritos utilizando la Composition API y <script setup>, en lugar de la Options API.

4.- Soporte para Visual Studio Code

La configuración recomendada del IDE es **VSCode + la extensión Vue Language Features (Volar)**. La extensión proporciona resaltado de sintaxis, soporte para TypeScript, intellisense para expresiones de plantillas y props de componentes.

5.- Componentes de archivo único (SFC)

Un Simple File Component (SFC) o en español “Componentes de archivo único” es un archivo .vue que consta de 3 partes:

- <template>donde está el contenido HTML.
- <script>para nuestro código Vue.
- <style>donde escribimos el estilo CSS.

Lo cuál facilita el manejo de proyectos más grandes y obtener un mejor entorno de desarrollo. Tiene sentido porque:

- Resulta más fácil gestionar proyectos más grandes con el uso de plantillas y componentes.
- Podemos ver y probar nuestro proyecto a través del protocolo https, tal como los usuarios verán la página.
- La página se actualiza inmediatamente cuando se guardan los cambios, sin necesidad de recargar.
- Así es como se construyen las páginas web reales en Vue.
- Así es como trabajan los desarrolladores.

A partir de ahora escribiremos los archivos de esta forma. Pero antes tenemos que configurar nuestro equipo de una manera diferente.

Los SFC son más fáciles de usar, pero no se pueden ejecutar directamente en el navegador, por lo que debemos configurar nuestra computadora para compilar nuestros archivos *.vue en archivos *.html, *.css y *.js para que el navegador pueda ejecutar nuestra aplicación Vue.

Para construir nuestra página web basada en SFCs utilizamos un programa llamado **Vite** como herramienta de construcción, y escribimos nuestro código en el editor de VS Code con la **extensión Volar** para las características del lenguaje Vue 3.

5.1.- Proyecto Composition API sin formato SFC

Este ejemplo se puede usar directamente en un archivo HTML y ejecutar con Vue 3.

Usamos <script> dentro de un archivo HTML hola.html y trabajamos con createApp() para montar la aplicación.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vue Composition API Example</title>
  <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
  <div id="app">{{ message }}</div>
  <button @click="increment">Click Count: {{ count }}</button>
```

```
<script>
  const { createApp, ref } = Vue;

  createApp({
    setup() {
      const message = ref('Hello, Vue with Composition API!');
      const count = ref(0);

      function increment() {
        count.value++;
      }

      return {
        message,
        count,
        increment
      };
    }
  }).mount('#app');
</script>
</body>
</html>
```

5.3.- Proyecto Composition API con formato SFC

Este ejemplo se usa en un archivo Vue con la estructura de un Single File Component.

Usamos la estructura de <template>, <script setup>, y <style> para encapsular todo el código del componente en un solo archivo. La Composition API se utiliza dentro de <script setup>, que es una forma más directa y sencilla de definir la lógica de tu componente en Vue 3.

Para que esto funcione, debes servir tu proyecto usando un servidor de desarrollo como Vite o Vue CLI. No puedes abrir index.html directamente en el navegador y esperar que funcione, ya que necesitas un entorno que compile los archivos .vue y gestione las importaciones de módulos.

Usando Vite sería:

```
npm create vite@latest my-vue-app --template vue
cd my-vue-app
npm install
npm run dev
```

La estructura de carpetas sería:

```
/project-root
  /src
```

Hola.vue
main.js
index.html

El fichero **index.html.**

Este es el archivo HTML que usas para cargar tu aplicación Vue:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vue App</title>
</head>
<body>
  <div id="app"></div>
  <script type="module" src="/src/main.js"></script>
</body>
</html>
```

El fichero **main.js**

Este archivo es el punto de entrada que importa Hola.vue y monta la aplicación:

```
import { createApp } from 'vue';
import Hola from './Hola.vue';

createApp(Hola).mount('#app');
```

El fichero **Hola.vue.**

```
<template>
  <div>
    <p>{{ message }}</p>
    <button @click="increment">Click Count: {{ count }}</button>
  </div>
</template>

<script setup>
import { ref } from 'vue';

const message = ref('Hello, Vue with Composition API and SFC!');
const count = ref(0);
```

```
function increment() {
  count.value++;
}
</script>

<style scoped>
button {
  margin-top: 10px;
  padding: 5px 10px;
  font-size: 16px;
}
</style>
```

ANEXO I: Ejemplos de renderizado

1. Ejemplo de HTML + CSS + JS (Ejemplo básico)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>HTML + CSS + JS</title>
    <style>
        /* Simple styles */
        .item { margin: 10px 0; }
    </style>
</head>
<body>
    <div id="app">
        <h1>Item List</h1>
        <ul id="item-list"></ul>
        <button onclick="addItem()">Add Item</button>
    </div>
    <script>
        const itemList = document.getElementById('item-list');
        let itemCount = 0;

        function addItem() {
            const item = document.createElement('li');
            item.className = 'item';
            item.textContent = `Item ${++itemCount}`;
            itemList.appendChild(item);
```

```
        }
    </script>
</body>
</html>
```

2. Ejemplo de Vue.js (Ejemplo básico)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue.js Example</title>
    <script src="https://cdn.jsdelivr.net/npm/vue@2"></script>
    <style>
        /* Simple styles */
        .item { margin: 10px 0; }
    </style>
</head>
<body>
    <div id="app">
        <h1>Item List</h1>
        <ul>
            <li v-for="(item, index) in items" :key="index" class="item">{{ item }}</li>
        </ul>
        <button @click="addItem">Add Item</button>
    </div>

    <script>
        new Vue({
            el: '#app',
            data: {
                items: [],
                itemCount: 0
            },
            methods: {
                addItem() {
                    this.itemCount++;
                    this.items.push(`Item ${this.itemCount}`);
                }
            }
        });
```

```
    </script>
</body>
</html>
```