# El Framework Progresivo

Conocemos a Vue como el Framework progresivo justamente por su vesatilidad. Es una pequeña librería que se encarga de resolver el problema de la vista dentro de lo que es el patrón MVC (Modelo, Vista, Controlador), pero también puede ser muy potente cuando la combinamos con parte de su ecosistema, es decir, que Vue puede ir escalando a medida que nuestro proyecto vaya incrementando los requerimientos o vaya incrementando su complejidad. Podemos utilizar Vue para proyectos muy sencillos como una Landing Page o un prototipo o para proyectos más complejos como una SPA (Single Page Application). Es decir que a medida que tengamos cosas más difíciles para resolver, vamos a ir incorporando herramientas para resolver esos problemas específicos.

Vue es una librería enfocada a la vista que utiliza el Virtual DOM y es totalmente reactiva. Tiene dos caracteristicas principales, el sistema declarativo y el sistema de componentes.

El sistema declarativo o declarative rendering es una de las funcionalidades que nos permite de forma imperativa poder manejar el DOM y interactuar con el mismo.

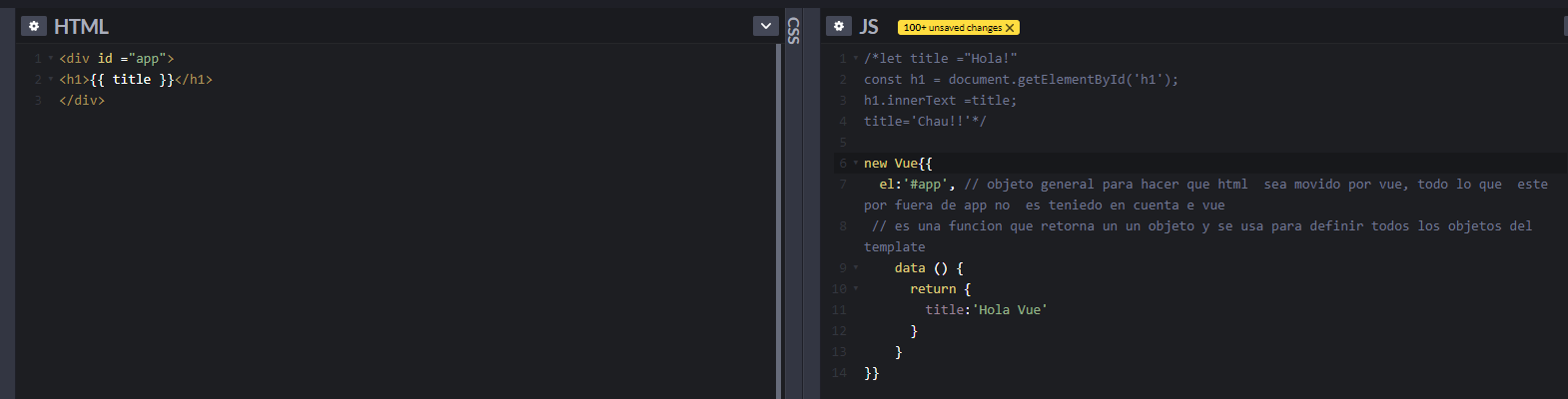
El sistema de componentes es lo que hace que podamos modularizar nuestra aplicación en diferentes componentes reutilizables o no y que nos permitan tener un código semántico, mucho más fácil de mantener y de leer.

<https://platzi.com/tutoriales/1548-react/1934-conceptos-basicos-de-react-js-que-deberias-saber/>

<https://platzi.com/tutoriales/1548-react/3779-crea-tu-propio-react/>

<https://platzi.com/tutoriales/1339-fundamentos-javascript/322-entendiendo-el-virtual-dom/>

# Hola Vue



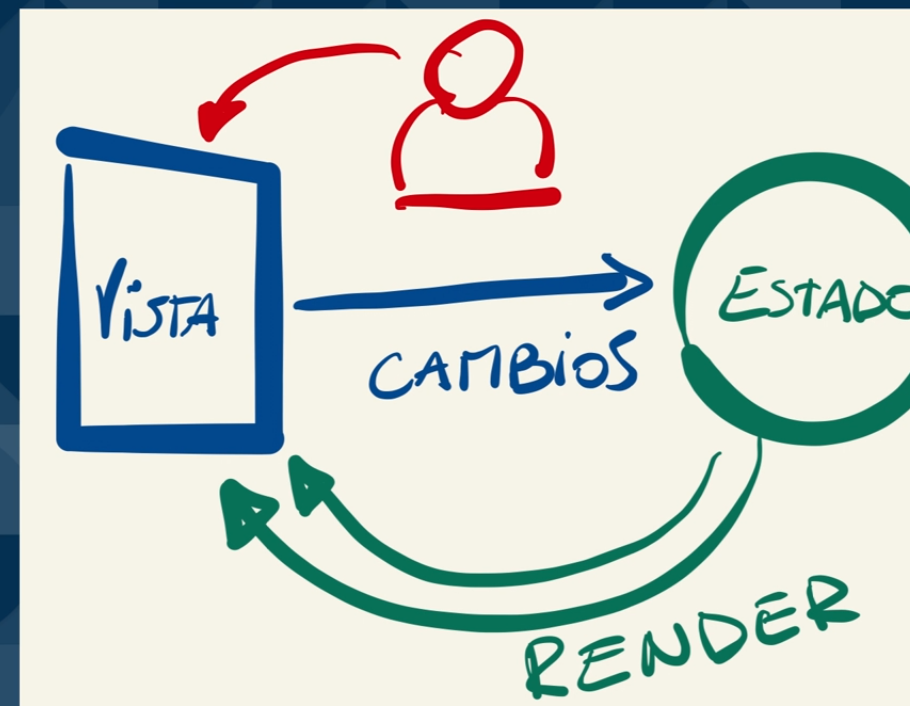
# Introducción al Rendering Declarativo

El Declarative Rendering es la opción que nos propone VueJS para interactuar con el DOM, principal con el concepto de Two Way Data Binding, esto quiere decir que vamos a tener:

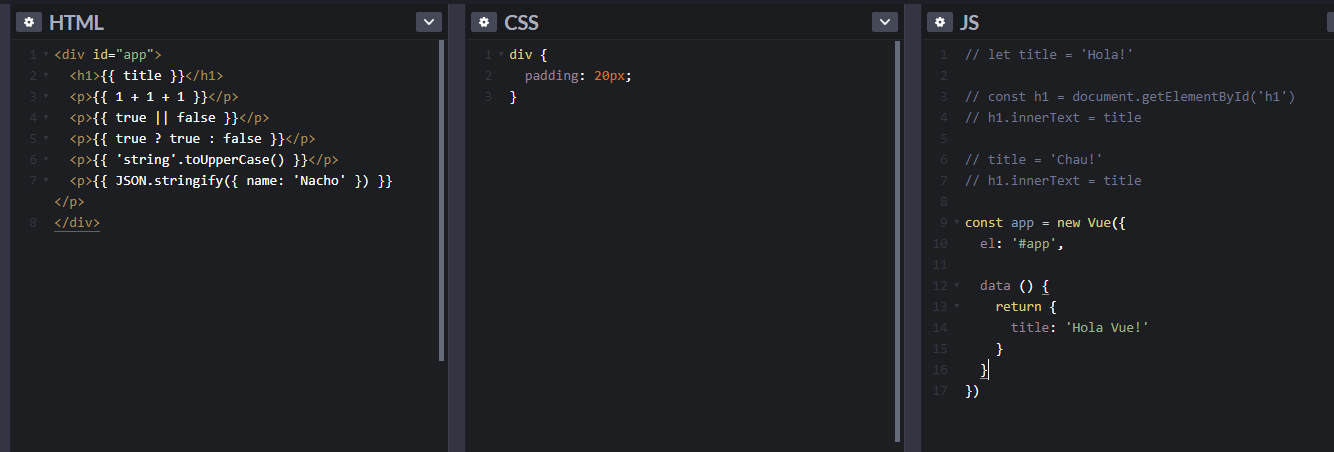
* Vista: aquí tenemos el HTML.
* Estado: aquí tenemos nuestros datos de JS.
* Usuario: es el que introduce cambios en la vista.

La vista se encarga de decirle al estado que hay cambios, a lo cual el estado va a reaccionar y mandar una nueva vista.

El estado se encarga de decirle a la vista como y cuando tiene que compilar y cuál es el resultado que se tiene que lograr, esto lo hace con una función que se llama render.



# Expresiones y Propiedades



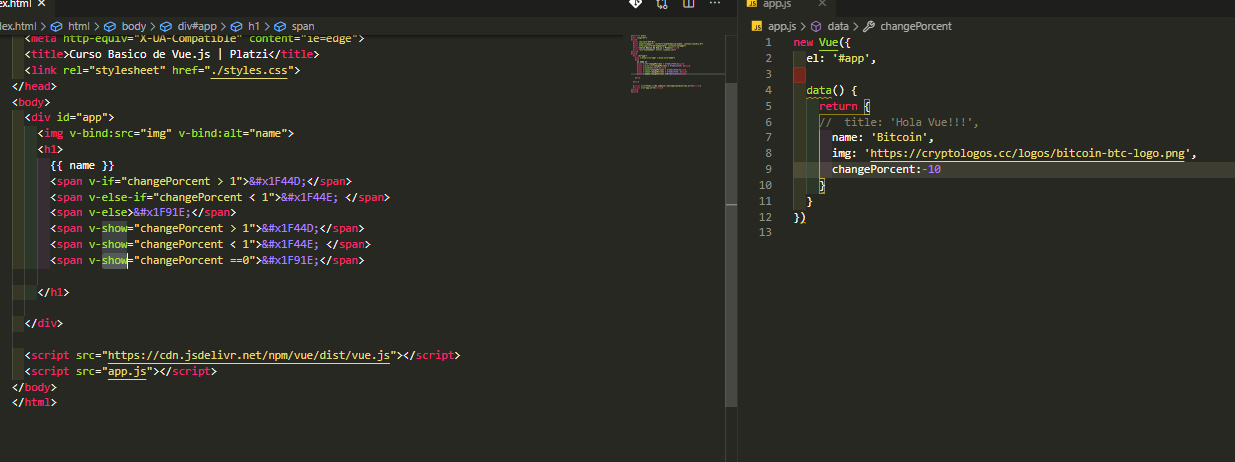
# Atributos Dinámicos

# 

# Control de Flujo con Directivas

Mostrar elementos Html cuando se cumplen condiciones.

La diferencia entre la condicional v-if y la condicional v-show es que una elimina el elemento del DOM y la otro lo oculta mediante el CSS display:none



# Renderizado de Listas

Para realizar la visualización de de listas usamos la directiva v-for la cual no permite llemar los valores del array, la propiedad key no ayuda a detectar el elemento y no perder el traqueo en caso de que la lista sufra modificaciones, otra cosa que se puede hacer es identificar el índice para ello usamos la llaves pues el primer valor identificara el índice y el segundo valor el valor,

Ejemplo sin índices

     <ul>

       <li v-for="p in prices" v-bind::key="p">{{p}} </li>

     </ul>

Ejemplo con índice

     <ul>

       <li v-for="(p, i) in prices" v-bind::key="p">{{i}} - {{p}} </li>

     </ul>

Con el v-for también se pueden mostar objetos, pero se debe tener en cuenta que el objeto tiene propiedades por lo cual al hacer la interpolación de valores se debe acceder a la propiedad y en el key se debe definir cual es el elemento referencia para no perder la referencia del objeto.

     <ul>

      <li v-for="(p, i) in pricesWithDays" v-bind::key="p.day">{{i}} - {{p.day}} {{p.value}} </li>

    </ul>

# Manejo de Eventos

Vue js

 showPrices:true

Vue HTML

     <ul v-show="showPrices">

      <li v-for="(p, i) in pricesWithDays" v-bind::key="p.day">{{i}} - {{p.day}} {{p.value}} </li>

    </ul>

## Methods

Instancia de vue donde puedo definir funciones y se usan para attachar eventos que puede ser  disparados por  vista, para acceder a las directivas de Data se debus usar la palabra reservada **this** smkskdlsl

VUE JS

     showPrices:false

    }

  },

  methods: {

    togggleShowPices() {

      this.showPrices = !this.showPrices

    }

  },

Vue Html

 <div id="app">

    <img  v-on:mouseover="togggleShowPices"

          v-on:mouseo ut="togggleShowPices"

          v-bind:src="img" v-bind:alt="name">

    <h1>

   <span v-on:click="togggleShowPices()">

{{showPrices ? '&#x1F648;':'&#x1F649;'}}

</span>

    </h1>

<https://marozed.com/vue-cheatsheet>

# Clases en tiempo real

<h1 v-bind:class="changePorcent > 0 ? 'green' : 'red'">

Con el v-bind usamos la directiva class para determinar el atributo css a cambiar

     <ul v-show="showPrices">

      <li class="uppercase"

          v-bind:class="{ orange: p.value === prices, red: p.value < prices, green: p.value > prices}"

          v-for="(p, i) in pricesWithDays"

          v-bind::key="p.day">{{i}} - {{p.day}} {{p.value}} </li>

    </ul>

# Estilos en tiempo real

# Para bindear alguna propiedad, como en este caso usamos v-bind:class podemos utilizar el atajo :, por lo que si queremos bindear la clase podemos utilizar el atajo como: :class, ambas maneras son válidas dentro de Vue.js

color: 'f4f4f4',

  methods: {

    togggleShowPices() {

      this.showPrices = !this.showPrices

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

    },

<div id="app" v-bind:style="{background: '#' + color}">

# Computed Properties y Watchers

# Son aquellas propiedades que se generan a partir del valor de otras propiedades , es decir que hay propiedades definidas en data que el coputarce generan otra propiedad

 name: 'Bitcoin',

      symbol: 'BTC',

  computed: {

    title() {

       return  ` ${this.name} - ${this.symbol}`

    }

  },

 <h1 v-bind:class="changePorcent > 0 ? 'green' : 'red'">

      {{title}}

# Watchers o disparador de codigo funciones que ejecutan un código y deben tener el nombre de una propiedad en el data, el Watchers recibe dos valores (valor Nuevo , valor viejo)

  watch: {

    showPrices(*newVal*, *oldVal*) {

      console.log(*newVal*, *oldVal*)

    }

  },

# Two-Way Data Binding

# Directiva v-model, permite linkiar las cosas que modifica un usuario desde in input con las propiedades definidas en data, esto se conoce como Two-Way Data Binding, es permitir que cada vez que se cambie la vista se refresque el código y que cada vez que cambie el código se refresque la vista

   <input type="number" v-model='value'>

     <span>{{convertedValue}}</span>

  data() {

    return {

value:0

}

}

  computed: {

    convertedValue() {

      if (!this.value) {

        return 0

      }

       return this.value / this.prices

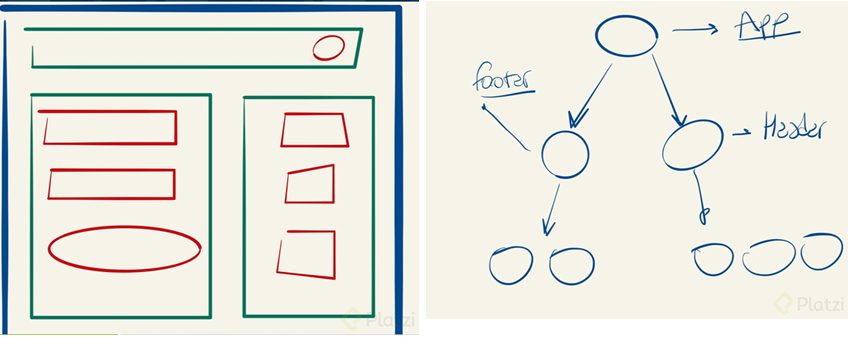
    }

  },

# SISTEMA DE COMPONENTES

Los componentes son la segunda funcionalidad importante que tiene Vue. Se basan en las especificaciones de web components APIs. Permiten modularizar mi aplicación en diferentes pedazos de htm, javascript y Css para tener un código más legible y semántico. Es decir, cada componente puede ser utilizado a lo largo del proyecto, tiene todo lo que necesita para existir, tiene su lógica, tiene diseño y tiene estructura.

Cuando trabajamos con Html y el DOM siempre tenemos una estructura de árbol, es decir, una estructura jerárquica DOM. Tenemos un componente principal (en este caso, el recuadro azul en la imagen abajo) y luego tenemos componentes hijos que representan contenedores con diferente tipo de contenido. Además, podemos tener otros componentes hijos, es decir, cada uno también tiene sus propios elementos Html.



Entonces podemos decir que la estructura al ser de árbol, siempre tiene un componente principal y componentes hijos que salen de ese componente principal. Este componente principal estaba representado por lo que conocemos como el componente root, o nuestra aplicación (App, en este caso) y luego tenemos los componentes hijos, como puede ser un header , un footer y después diferente tipo de contenido que podemos tener a lo largo nuestra estructura. La idea de los componentes es llevar toda nuestra lógica a pedazos de Html reutilizables que luego vamos a escribir de manera muy sencilla y semántica dentro de nuestro elemento o dentro de nuestro componente principal.

Por ejemplo, podríamos pensar un componente que tenga un header de Html, que también tenga algo así como un componente que se llame login . De igual modo otro componente que podría ser Custom y luego tener un footer. De esta forma combinamos los elementos de Html5 con nuestros propios componentes para conseguir un código semántico y además tenemos la ventaja de que dentro de otra página o dentro de otro componente, también podemos utilizar los componentes que creamos, por ejemplo, el componente Custom o el componente footer que es nativo de Html

# Crear Componentes Custom

//definición del componente

Vue.component('counter',{

  data(){

    return {

      counter:0

    }

  },

    methods:{

      increment() {

        this.counter +=1;

      }

    },

  template: `

    <div>

      <button v-on:click="increment">Click Me!!</button>

      <span>{{counter}}</span>

    </div>

`

})

//definición de la instancia vue

new Vue ({

  el:'#app',

  data(){

   return {

     title:'hola'

   }

}

})

# HTML con el nuevo componente counter

<div id="app">

  <h1>{{ title }}</h1>

  <counter></counter>

</div>

# Comunicación entre Componentes: propiedades

# Definición de nuevo componente

Vue.component('CoinDetail', {

  props: ['changePorcent','title'],

  data () {

    return {

      showPrices:false

    }

  },

  methods: {

    togggleShowPices() {

      this.showPrices = !this.showPrices

    }

  },

template: `

  <h1 v-bind:class="changePorcent > 0 ? 'green' : 'red'">

    {{title}}

    <span v-if="changePorcent > 1">&#x1F44D;</span>

    <span v-else-if="changePorcent < 1">&#x1F44E; </span>

    <span v-else>&#x1F91E;</span>

    <span v-on:click="togggleShowPices()">{{showPrices ? '&#x1F648;':'&#x1F649;'}}</span>

</h1>`

})

# Uso del nuevo conponente en Html

      <coin-detail

          v-bind:title="title"

          v-bind:changePorcent="changePorcent">

      </coin-detail>

# Comunicación entre Componentes: eventos

# La comunicación entre componentes padres a hijos es mediante propiedades (v-bind:), mientras que la comunicación de hijos a padres es mediante eventos (v-on)

# Ejemplo

  methods: {

    togggleShowPices() {

      this.showPrices = !this.showPrices

      //cada vez que se ejecute la funcion togggleShowPices se emitira el evento change-color

      this.$emit('change-color')

    },

  },

# Para que el html pueda leer el evento usaremos la propiedad v-on y el nombre del evento para invocar la función principal

# 

# 

# <https://es.vuejs.org/v2/guide/components.html#Enviando-mensajes-a-componentes-padre-con-eventos>

# Slots

En los componentes cuando ocupamos recibir contenido en lugar de recibir propiedades, podemos utilizar Slots que es una API de distribución de contenido que permite que un componente padre le inyecte HTML a un componente hijo.

Para utilizar varios Slots, se debe colocar un nombre a cada slot para identificarlos:

<slot name="text"></slot>

Y para llamar al Slot en el HTML se debe encerrar el Slot dentro del tag template que es un tag de VueJS, que permite renderizar contenido sin la necesidad de utilizar un tag, el tag template en la renderización final es eliminado:

<template v-slot:text>

<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Excepturi, eaque quisquam. Exercitationem, quisquam alias magnam vel unde quidem eius ipsa!</p>

</template>

6

# Ciclo de Vida y Hooks

# Vue tiene diferentes estados y esto nos permite usar ciertas acciones antes y después de cada estado.

# A estas acciones se las conoce como hooks y tienen unos propósitos claros:

• Nos permite conocer como se crea, actualiza y se destruye un componente.

• Podemos saber en que punto se encuentra el componente y poder actuar en consecuencia.

# A continuación vamos a ver los hooks que nos ofrece Vue:

1. beforeCreate.

2. created.

3. beforeMount.

4. mounted.

5. beforeUpdate.

6. updated.

7. beforeDestroy.

8. destroyed.

# <https://vuejs.org/v2/guide/instance.html#Lifecycle-Diagram>

  created() {

    console.log('created !!!!')

    // la mejor opcion para  api rest o servidor  (info dinamica )

  },

  mounted() {

    console.log('mounted !!!!!')

    // ya esta disponible el dom y puedo accedier elemento html oy/o otros que no esten el evento created

  },

  methods: {

    updateColor(*color*) {

      this.color = *color* || this.color.split('').reverse().join('')

    },

  },

})

# Estos eventos también pueden existir en los componentes Hijos y este es un ejemplo de ejecución

# 

# El ciclo de vida de la instancia de Vue

# Qué es, cómo usarlo y aplicaciones profesionales con el CLI

El vue-cli (es decir, la “interfaz de línea de comandos Vue”) es una herramienta creada por el equipo Vue.js para ayudar a facilitar el rápido desarrollo de las aplicaciones Vue.

Si bien puede usar Vue.js en el navegador directamente, la mayoría de las aplicaciones frontales usan Node.js para las herramientas de compilación.

# Impot5ancia, reducir todo el esfuerzo de configuración y tener el proyecto listo para trabajar lo más rápido posible

# <https://cli.vuejs.org/>

instalación cli

npm install -g @vue/cli

# OR

yarn global add @vue/cli

Crear proyecto

vue create my-project

# OR

vue ui

Para iniciar el proyecto debemos ejecutar el comando **npm run serve si todo sale bien generara la siguiente pestaña**

# 