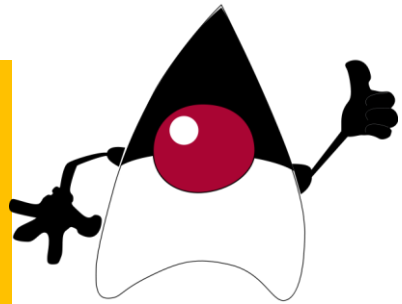
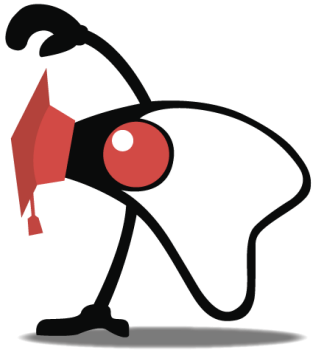
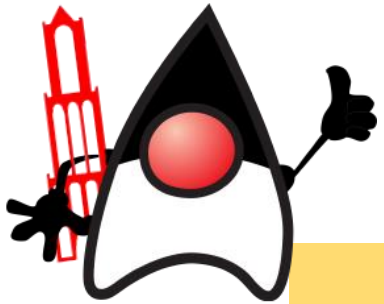
Curso FullStack Python

Codo a Codo 4.0







VUE.js

Parte 1

VUE.js

Con JavaScript nos encontramos que para hacer varias cosas necesitamos de mucho tiempo, esfuerzo y muchas líneas de código. Así como Bootstrap era un framework que me permitía resolver muy fácilmente cuestiones de estilos y estructuras, Vue me va a permitir resolver varias cuestiones del comportamiento porque es un **framework exclusivo de JS**.

Nos permitirá conectarnos con mi documento HTML en forma sencilla y podremos realizar cosas que desde JS puro y HTML implicarían un mayor esfuerzo.

Resulta un primer paso para otros frameworks de desarrollo de JS como pueden ser Angular y React, que requieren un paso más.

VUE está enfocado para armar aplicaciones de Single Page (esas que permiten navegar todo en una sola página). La idea de VUE es, por ejemplo, que no se tenga que cargar toda la información de una, que no se carguen todos los comentarios en un posteo, sino lo más relevante y a medida que vaya bajando se vayan cargando. Podemos actualizar **partes** del documento, y no toda la página que ahorra tiempo y recursos.

No nos sirve de nada ver un Framework si no tenemos las bases de JS, porque necesitamos los conocimientos previos.

VUE.js

Es un framework de JavaScript. La primera versión se lanzó en febrero de 2014.

Vue (*pronunciado /vjuː/, como view*) es un **framework** progresivo de código abierto que se utiliza para desarrollar interfaces Web interactivas. A diferencia de otros frameworks, Vue está diseñado desde cero para ser utilizado incrementalmente y simplificar el desarrollo.

La **librería central** está enfocada solo en la *capa de visualización*, y es fácil de utilizar e integrar con otras librerías o proyectos existentes en desarrollos front-end.

Por otro lado, Vue también es perfectamente capaz de impulsar sofisticadas **Single-Page Applications (SPA)** cuando se utiliza en combinación con herramientas modernas y librerías de apoyo.

Las SPA son como una “sábana” donde a medida que vamos *scrolleando* nos movemos a distintas secciones de la misma página. El contenido no se va a cargar en forma completa, ya que es poco eficiente, repercute en la experiencia de usuario que va a esperar que se cargue todo el contenido cuando en realidad quiere ver lo que ya cargó.

> ***Ejemplo**: si un influencer tiene 10000 comentarios en cada posteo no los vamos a cargar todos juntos, sino que vamos a dividir por partes el documento HTML de forma tal que la información relevante se la cargue rápido al usuario.*

DOM Virtual

Vue.js utiliza **DOM virtual**, que también es utilizado por otros frameworks como React, Ember, etc.

Los cambios **no se realizan en el DOM**, sino que se crea una **réplica del DOM** que está presente en forma de estructuras de datos JavaScript. Siempre que se deben realizar cambios, se realizan en las estructuras de datos de JavaScript y esta última se compara con la estructura de datos original.

Luego, los cambios finales se actualizan al DOM real, que el usuario verá cambiar. Esto es bueno en términos de optimización, es menos costoso y los cambios se pueden realizar a un ritmo más rápido.

> *Desde JS haremos cambios en las estructuras de datos propias de JS, esa copia la va a comparar con el DOM y **sólo va actualizar los cambios**. Aquí radica la gran ventaja de trabajar con VUE, porque si tuviese que actualizar todo repercutiría en la performance.*

Modelo de enlazado de datos

Es la forma a través de la cual JavaScript se conecta (enlaza, comunica) con Vue.js., permitiendo comunicar el documento HTML con JS.

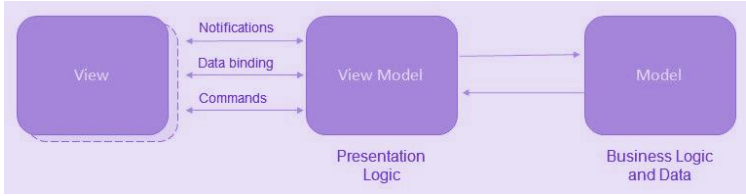
Data binding: A través de la función de data binding podremos manipular o asignar valores a atributos HTML, cambiar el estilo, asignar clases con la ayuda de la **directiva** de enlace **v-bind**.

Modelo de vista (*model–view–viewmodel o MVVM*)

El patrón modelo–vista–modelo de vista es un patrón de **arquitectura de software**. Se caracteriza por tratar de desacoplar lo máximo posible la interfaz de usuario de la lógica de la aplicación.

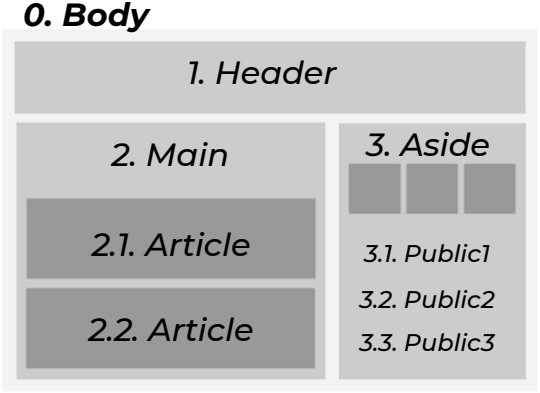
Sus elementos son:

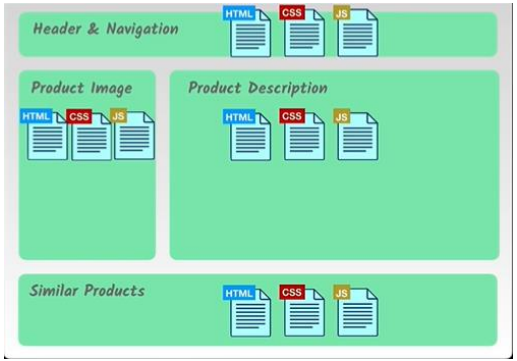
- **La vista**: Representa la información a través de los elementos visuales que la componen. Son activas, contienen comportamientos, eventos y enlaces a datos que, en cierta manera, necesitan tener conocimiento del modelo subyacente.
- **Modelo de vista**: Actor intermediario entre el ***modelo*** y la ***vista***, contiene toda la lógica de presentación y se comporta como una abstracción de la interfaz. La comunicación entre la vista y el viewmodel se realiza por medio de los enlaces de datos.
- **El modelo:** Representa la capa de datos y contiene la información, pero nunca las acciones o servicios que la manipulan. No tiene dependencia con la vista.

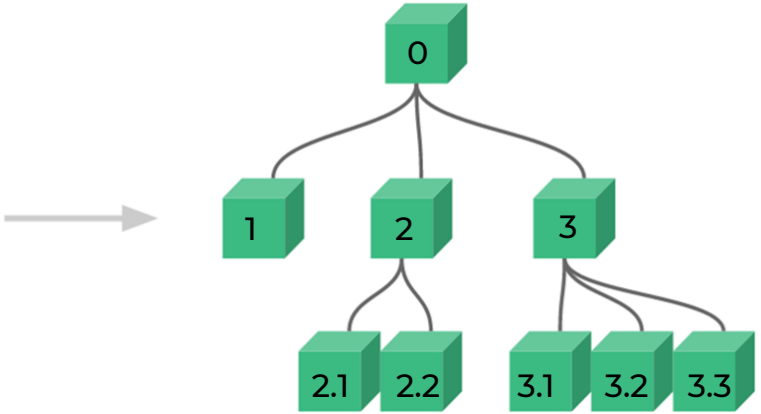


Organización de componentes en VUE.js

Es común que una aplicación se organice en un árbol de componentes anidados:







Vue te permite tomar una pagina web y dividirla en componentes cada uno con su HTML, CSS y JS necesario para generar esa parte de la página.

Permite hacer una ***"intervención por partes",*** por ejemplo para intervenir sobre el header y footer, que es siempre el mismo

Comenzando con VUE.js

La forma más fácil de comenzar a usar Vue.js es crear un archivo index.html e incluir Vue con:

```html
<script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
```

HTML

La página de instalación https://es.vuejs.org/v2/guide/installation.html proporciona más opciones de instalación de Vue.

Nota: No recomendamos que los principiantes comiencen con vue-cli.

IMPORTANTE: Debemos colocar la referencia al CDN al final del <body> y antes de la referencia a nuestro archivo .js

```js
const app= new Vue({

})
```

JS

Esta constante me conecta VUE con mi HTML y tiene un objeto de tipo VUE. Dentro de las llaves voy a tener propiedades y valores.

Más información: https://www.w3schools.com/whatis/whatis_vue.asp
https://es.vuejs.org/v2/guide/

Hola mundo con VUE.js

En nuestro primer caso, tendremos en el documento HTML un elemento div con un ID que va a conectar con mi archivo JS

HTML

```
<body>
    <div id="app">
        <p>
            {{mensaje}}
        </p>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    <script src="intro-vue.js"></script>
</body>
```

JS

```
const app= new Vue({
    el: '#app',
    data: {
        mensaje: "Hola Mundo con Vue!",
        nombre: "Juan Pablo"
    }
})
```

Hola Mundo con Vue!

new Vue *es el objeto de tipo VUE y lo que está entre {{}} es el contenido que yo quiero* ***cambiar*** *de la página.*

La conexión desde JS con mi documento HTML a través de VUE se llama ***renderización declarativa***. Tiene que ver con enlazar el contenido del HTML que estoy presentando a través de VUE, ya no modificándolo desde JS a través del manejo del DOM, sino a través de VUE.

Ver ejemplo intro-vue (.html y .js)

Renderización declarativa (interpolación)

Nos permite insertar texto en el documento HTML, algún valor, propiedad o atributo. Por ejemplo, podremos agregar algún mensaje dentro de las etiquetas HTML.

VUE utiliza las **llaves dobles** para encerrar el dato que quiere mostrar {{ }}, similar a Template String de JS, que lo hace con **${ }**

{{ message }}

Al partir del uso de la doble llave lo que vamos a estar haciendo es **vincular los datos con el DOM**, reaccionando a esos nuevos valores. Al cambiar esa réplica del DOM (*DOM virtual*) lo voy a ver reflejado en el DOM ya que framework al detectar un cambio lo actualiza.

Los **datos** y el **DOM** ahora están vinculados, y ahora **todo es reactivo** (sólo se modifica ante los cambios). Si cambio el valor de app.message a un valor diferente, debería ver que el ejemplo se ha renderizado con el nuevo valor que acaba de ingresar.

Directivas

Una **directiva** es el término usado para referirse a algunos atributos especiales que le indican a Vue.js que debe realizar ciertos cambios en un elemento del DOM, cada vez que la expresión asociada con dicha directiva cambie. Vue utiliza **directivas** para aplicar un comportamiento especial al DOM. Las directivas nos permiten enlazar VUE con nuestro HTML pero con los **atributos** de las etiquetas, no solo con el contenido. Tienen el prefijo **v-** para indicar que son atributos especiales proporcionados por Vue.

- **v-text:** https://es.vuejs.org/v2/api/#v-text
- **v-bind:** https://es.vuejs.org/v2/api/#v-bind
- **v-if, v-else, v-elseif:** https://es.vuejs.org/v2/api/#v-if
- **v-for:** https://es.vuejs.org/v2/api/#v-for
- **v-show:** https://es.vuejs.org/v2/api/#v-show
- **v-model:** https://es.vuejs.org/v2/api/#v-model

Más directivas: https://es.vuejs.org/v2/api/#Directivas

v-bind: permite enlazar (*bindear*) una variable de Vue con un atributo específico de una etiqueta HTML.

```html
<a href="#" v-bind:title="mensaje">mail@mail.com</a>
```
HTML

*Con **v-bind:title** estamos modificando el atributo title dentro de la etiqueta a, mostrando el contenido de la propiedad **mensaje***

Más información: clic aquí y aquí

Directivas: v-for (renderización de una lista)

Podemos usar la directiva **v-for** para representar una lista de elementos basada en un Array. La directiva **v-for** requiere una sintaxis especial en forma de ***item*** *in* ***items***, donde los **items** son el array de datos de origen y el **item** es un alias para el elemento del Array que se está iterando:

HTML

```
<ul id="example-1">
    <li v-for="fruta in frutas">
        {{ fruta.nombre }}
    </li>
</ul>
```

JS

```
var example1 = new Vue({
    el: '#example-1',
    data: {
      frutas: [
        {nombre:"naranja"},
        {nombre:"banana"},
        {nombre:"pera"}
      ]
    }
})
```

- naranja
- banana
- pera

En este caso cargamos ítems desde una lista, pero con datos almacenados en un array desde JavaScript, aprovechando la directiva v-for.
El contenido podrá ser dinámico, cargándose en función de algo previamente almacenado.

fruta *es cada elemento de la lista, mientras que* ***frutas*** *es el array en sí.*

Directivas: v-if, v-else, v-elseif

Podemos establecer que el contenido se muestre dependiendo de alguna condición.

Ejemplo: En un listado de productos podemos hacer que aquellos que tengan un stock igual a 0 nos avise de alguna manera:

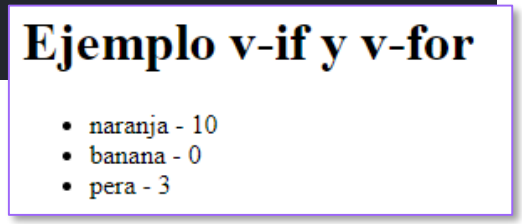
HTML

```
<div id="ejemplo">
    <h1> {{ titulo }}</h1>
    <ul>
        <li v-for="fruta in frutas">
            {{ fruta.nombre }} - {{ fruta.cantidad }}
        </li>
    </ul>
</div>
```

JS

```
var ejemplo_vIf_vFor = new Vue({
    el: '#ejemplo',
    data: {
      titulo: "Ejemplo v-if y v-for",
      frutas: [
        {nombre:"naranja", cantidad: 10},
        {nombre:"banana", cantidad: 0},
        {nombre:"pera", cantidad: 3}
      ]
    }
})
```

*En este ejemplo iteramos sobre el array de frutas, mostrando de ese objetos dos propiedades: **nombre** y **cantidad**.*



Utilizaremos un condicional **v-if** para determinar qué elementos no tienen stock (= 0).

Directivas: v-if, v-else, v-elseif

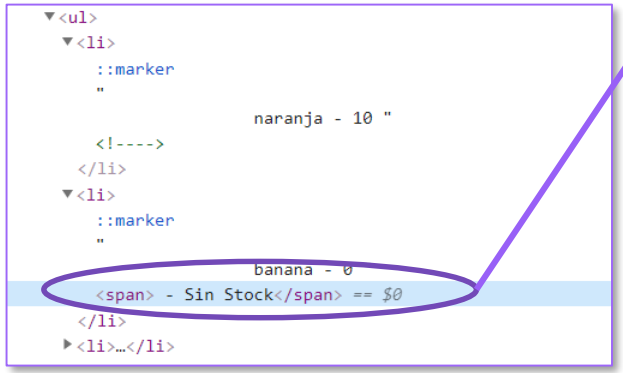
Agregaremos una etiqueta **** que aparecerá solamente en caso de cumplirse una condición:

HTML

```
<li v-for="fruta in frutas">
   {{ fruta.nombre }} - {{ fruta.cantidad }} <span v-if="fruta.cantidad===0"> -
 Sin Stock</span>
</li>
```

- naranja - 10
- banana - 0 - Sin Stock
- pera - 3

Cuando dentro del array la propiedad cantidad del elemento fruta sea igual a 0 se mostrará el texto "Sin Stock"



*Si inspeccionamos el documento veremos que el **** sólo aparece en el segundo ítem,*
Esto lo resuelve VUE a través de JavaScript.

Ver ejemplo: v-if for (.html y .js)

Directivas: v-if, v-else, v-elseif

Ampliaremos el ejemplo anterior incorporando más elementos al array y estableciendo otras condiciones:

- Stock = 0: Sin stock
- Stock < 5: Stock bajo
- Stock >=5 Stock alto

Para esto emplearemos v-else-if y v-else:

HTML

```
<li v-for="fruta in frutas">
    {{ fruta.nombre }} - {{ fruta.cantidad }}
    <span v-if="fruta.cantidad===0"> - Sin Stock</span>
    <span v-else-if="fruta.cantidad<5"> - Stock Bajo</span>
    <span v-else="fruta.cantidad>=5"> - Stock Alto</span>
</li>
```

*El **v-for** iterará sobre cada elemento y determinará con los condicionales cuál es la situación de cada elemento (sin Stock Stock Bajo o Stock Alto)*

Ver ejemplo: v-if for2 (.html y .js)

Las directivas v-model y v-on

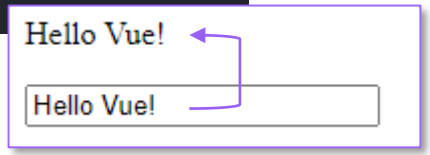
La directiva **v-model** establece un enlace bidireccional, es decir, vincula el valor de los elementos HTML a los ***datos*** de la aplicación. La directiva **v-on** permite escuchar eventos DOM y ejecutar algunas instrucciones de JavaScript cuando se activan.

HTML

```
<div id="app">
    <p>{{ message }}</p>
    <p><input v-model="message"></p>
</div>
```

JS

```
var myObject = new Vue({
    el: '#app',
    data: { message: 'Hello Vue!' }
})
```



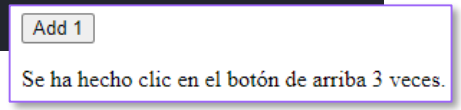
Aquí en el input, lo que ingrese el usuario, modifica el message y el message modifica el 1er párrafo. A medida que se escriba en el input ese cambio se verá reflejado en el párrafo.

HTML

```
<div id="example-1">
    <button v-on:click="counter += 1">Add 1</button>
    <p>Se ha hecho clic en el botón de arriba {{ count
er }} veces.</p>
</div>
```

JS

```
var example1 = new Vue({
  el: '#example-1',
  data: {
    counter: 0
  }
})
```



*Se asocia la directiva **v-on** al evento clic y se incrementa en 1 el valor de la propiedad counter*

Ver ejemplos v-model y v-on (.html y .js)

Ejemplos, cursos y guías de VUE.js. APIS

- **Guía de VUE,js:** https://es.vuejs.org/v2/guide/index.html#
- **Ejemplos VUE:** https://vuejsexamples.com/
- **Escuela VUE:** https://escuelavue.es/series/
- **¿Qué son las APIs y para qué sirven?:** https://youtu.be/u2Ms34GE14U
- **Curso de Vue JS - Tutorial en Español [Desde Cero]**: https://www.youtube.com/playlist?list=PLPl81lqbj-4J-gfAERGDCdOQtVgRhSvIT
- **VUE Mastery (curso):** https://www.vuemastery.com/courses/intro-to-vue-js/vue-instance/
- **Lenguaje JS - ¿Qué es VUE?**: https://lenguajejs.com/vuejs/introduccion/que-es-vue/