Capítulo 2

Introducción a Vue.js

2.1. Introducción

Convertirse en un desarrollador frontend es un camino de constante aprendizaje, en el cual después de dominar HTML, CSS y Javascript a un nivel medio-avanzado comienza a abrirse un amplio abanico de posibilidades con todas las herramientas que un desarrollador tiene a su alcance, principalmente el conjunto de **Frameworks y Librerías** que implemente en cada proyecto.

Vue JS es un **framework progresivo** para desarrollar interfaces de usuario creado en el 2014 por Evan You que trabajó como desarrollador frontend en Google. Vue fue desarrollado buscando obtener una herramienta que pudiera ser de fácil aprendizaje y se adaptara a las diferentes necesidades de proyectos simples y complejos.

¿ Qué es Vue.js y por qué es tan especial? https://www.youtube.com/watch?v=AqesL138vMA (14:57)

Vue JS es uno de los Frameworks de mayor popularidad junto con ReactJS https://www.youtube.com/watch?v=bvxm389cYVI (4:58) y Angular https://www.youtube.com/watch?v=nJsCbxsdZ28 (5:55) y presenta las siguientes características:

- Es **accesible**: Es Software Open Source y es posible acceder a él directamente desde Vuejs.org en donde se puede encontrar la documentación oficial.
- Es **progresivo**: Vue JS puede ser implementado para proyectos muy básicos o para algo más complejo. Una de las grandes ventajas al ser un framework progresivo es su facilidad para adaptarse al crecimiento del proyecto.
- Es **escalable**: Su librería principal es pequeña lo que le permite adaptarse a proyectos grandes a través de Plugins.
- Es **reactivo**: Esto quiere decir que es posible desarrollar una aplicación que tenga una interacción constante con su entorno, de esta forma los cambios de estado interno se realizan por medio de eventos y generan diferentes reacciones cuando son accionados.
- Utiliza **componentes**: Permite crear componentes y utilizarlos en diferentes secciones de la aplicación.

2.2. Web Components

Los web components son un estándar del W3C *(Consorcio internacional que genera recomendaciones y estándares que aseguran el crecimiento de la World Wide Web a largo plazo)* que nos permiten desglosar el desarrollo de aplicaciones web en pequeños contenedores que encapsulan marcado HTML, código JavasScript y estilos CSS, recibiendo el nombre de componentes.

Los componentes web son bloques de código que encapsulan la estructura interna de elementos HTML, incluyendo CSS y JavaScript, permitiendo así que el código se pueda volver a usar como se quiera en otras webs y aplicaciones.

Los web components surgieron como propuesta por parte de Google a la W3C. El W3C está encargado de mantener los estándares, pero lo cierto es que sus procedimientos para permitir la evolución de la web son un poco lentos. Los desarrolladores habitualmente detectan necesidades mucho antes que la W3C realice un estándar para poder cubrirlas. De hecho, pueden pasar años desde que algo comienza a ser usado en el mundo de la web hasta que se presenta el estándar. Como el mundo de la web va mucho más rápido que la definición de los estándares han decidido no crear nuevas etiquetas HTML sino que han generado unos estandares para que los desarrollares creen sus propias etiquetas.

Con Web Components, se puede hacer casi cualquier cosa que pueda ser hecha con HTML, CSS y JavaScript, y puede ser un componente portable que puede ser reutilizado fácilmente.

Los web components son componentes que tienen propiedades y eventos y podemos montar una paleta de componentes reutilizable.

Los Web Components constan de cuatro tecnologías o estandares que pueden ser usados conjuntamente o por separado.

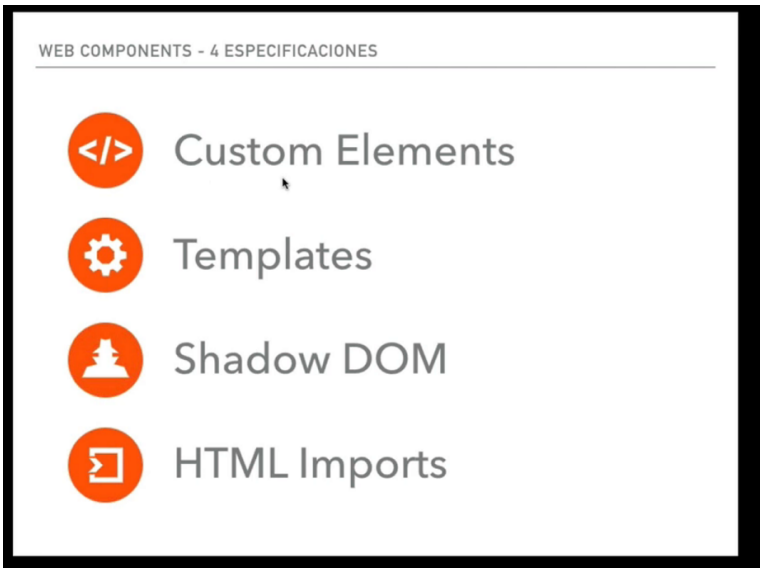


Figura 2.1: Especificaciones de los web components

2.2.1. Custom elements (Elementos personalizados)

https://developer.mozilla.org/es/docs/Web/Web_Components/Using_custom_elements

Los Custom Elements son una característica que permite crear elementos HTML personalizados. Pueden tener un comportamiento personalizado y estilos CSS propios. Son una parte de los Web Components, pero también pueden ser utilizados independientemente.

Para crear nuestra etiqueta debemos crear una clase que herede de HTMLElement y luego con el método define, unir la etiqueta y la clase.

```
1  // Creación
2  class BotonComprar extends HTMLElement {
3    // Aquí iría el código del elemento
4    // Eventos, funciones, etc...
5  }
6
7  window.customElements.define('boton-comprar', BotonComprar);
8
9  // Uso
10 <boton-comprar></boton-comprar>
```

La clase HTMLElement posee los siguientes métodos de ciclo de vida:

- **connectedCallback**: Se llama cada vez que el elemento se inserta en

el DOM.

- **disconnectedCallback**: Este método se llama cuando el componente es eliminado del DOM.
- **attributeChangedCallback**: Este otro método se llama cuando se añade un nuevo atributo, se actualiza o se elimina.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script>
        let variable;

        class BotonComprar extends HTMLElement {
            constructor () {
                super();
            }

            connectedCallback () {
                this.innerHTML = `<div>        <button
                    onclick="fComprar()">Comprar Ahora</button>    </div>
                    `;
                variable = prompt("Indica el nombre del producto");
                alert("Si quieres comprar " + variable + " haz clic en el
                    botón");
            }

            disconnectedCallback () {
                alert("Compra anulada");
            }
        }

        function fComprar(){
            alert("Acabas de comprar una unidad de " + variable);
        }

        window.customElements.define('boton-comprar', BotonComprar);

        function fanular()
        {
            document.getElementById("b1").remove();
        }
    </script>
</head>
<body>
<boton-comprar id="b1"></boton-comprar>
<input type="button" value="Anular compra" onclick="fanular()"/>
</body>
</html>
```

El **nombre de la etiqueta** para que sea válido debe seguir unas reglas muy concretas. Debe empezar por una letra entre la a y la z en minúsculas (no

puede ser ñ o una vocal acentuada), debe contener un guión - y no puede utilizar letras mayúsculas. No hay ni habrá etiquetas HTML que contengan un guión, así que cuando veamos una con guión es un custom components.

A partir de estas reglas, se pueden añadir todo tipo de caracteres Unicode, siempre y cuando no coincidan con palabras reservadas.

Hoy en día casi todos los navegadores soportan los web components. Si no es asi se pueden usar **polyfills**. Estos polyfills a través de librerías implementan los web componentes en navegadores que no lo soportan. Se pueden descargar desde `https://www.webcomponents.org/polyfills/`

Desde direcciones como `https://www.webcomponents.org/` podemos acceder a componentes hechos por la comunidad.

2.2.2. Templates (plantillas)

https://developer.mozilla.org/es/docs/Web/HTML/Element/template

https://developer.mozilla.org/es/docs/Web/Web_Components/Using_templates_and_slots

Cuando hay que reutilizar las mismas estructuras de lenguaje de marcado repetidas veces en una página web, tiene sentido utilizar algún tipo de plantilla en lugar de repetir la misma estructura una y otra vez. Esto ya era posible hacerlo antes, pero ahora es mucho mas fácil con el elemento HTML template.

```
1
2 <template id="my-paragraph">
3   <p>Mi párrafo</p>
4 </template>
```

Esto no aparecerá en tu página hasta que hagas una referencia a él con JavaScript y luego lo agregues al DOM, usando algo parecido a lo siguiente:

```
1 let template = document.getElementById('my-paragraph');
2 let templateContent = template.content;
3 document.body.appendChild(templateContent);
```

```
1 customElements.define('my-paragraph',
2   class extends HTMLElement {
3     constructor() {
4       super();
5       let template = document.getElementById('my-paragraph');
6       let templateContent = template.content;
```

2.2.3. Shadow DOM (DOM oculto)

https://developer.mozilla.org/es/docs/Web/Web_Components/Using_shadow_DOM

Shadow DOM permite adjuntar arboles DOM ocultos a elementos en el arbol DOM regular. Este árbol shadow DOM comienza con un elemento shadow root, debajo del cual se puede adjuntar cualquier elemento que se desee, de la misma manera que el DOM normal.

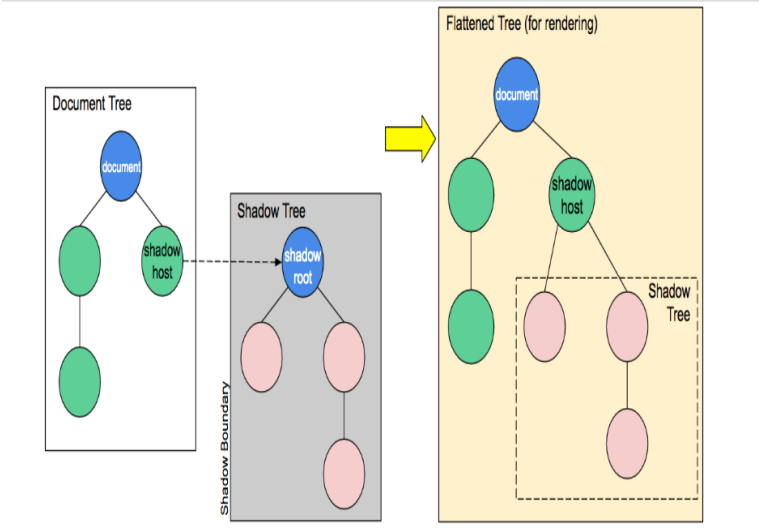


Figura 2.2: Árboles DOM

Shadow host: El nodo regular del DOM al que es atado el shadow DOM.

Shadow tree: El árbol DOM dentro del shadow DOM.

Shadow root: El nodo raíz del árbol Shadow.

Se puede adjuntar un 'shadow root' a cualquier elemento utilizando el método Element.attachShadow (). Éste toma como parámetro un objeto que contiene una propiedad mode con dos posibles valores: 'open' o 'closed'.

```
1 let shadow = elementRef.attachShadow({mode: 'open'});
2 let shadow = elementRef.attachShadow({mode: 'closed'});
```

open significa que puede acceder al shadow DOM usando JavaScript en el contexto principal de la página.

Ejemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
      <script>
      customElements.define('show-hello', class extends HTMLElement {
      connectedCallback() {
          const shadow = this.attachShadow({mode: 'open'});
          shadow.innerHTML = `<p>
            Hello, ${this.getAttribute('name')}
            </p>`;
        }
       });
      </script>

</head>
<body>
<show-hello name="John"></show-hello>
</body>
</html>
```

Ejemplo más completo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <template id="mi-parrafo">
        <style>
            p {
                color: white;
                background-color: #666;
                padding: 5px;
            }
        </style>
        <p>Mi párrafo</p>
    </template>
    <script>
        customElements.define('mi-parrafo',
            class extends HTMLElement {
                constructor() {
                    super();
                    let template = document.getElementById('mi-parrafo');
                    let templateContent = template.content;

                    const shadowRoot = this.attachShadow({mode: 'open'})
                        .appendChild(templateContent);
                }
            })
    </script>
</head>
<body>

<mi-parrafo></mi-parrafo>
</body>
</html>
```

2.2.4. HTML imports

Originalmente, la especificación de los Web Components estableció un mecanismo de importación por medio de la etiqueta link. Para ello añadió el atributo rel=”import” y utilizaba el atributo href para apuntar a un fichero HTML donde teníamos las etiquetas template con la plantilla y el script con el código del componente.

Lo cierto es que resulta sencillo de utilizar. Normalmente estas etiquetas se incluyen en el elemento head y quedan más o menos de esta forma:

```
<head>
 <meta charset="UTF-8";>
  <title>HTML Import</title>
  <link ref="import" href="./components/my-component.html">
</head>
```

En estos momentos se considera una funcionalidad deprecated, es decir, no recomendada y descontinuada. Todo surgió cuando Mozilla puso en entredicho la idoneidad de esta solución y anunció que no la implementaría en Firefox. Se produjo una reacción en cadena que ha terminado con el consenso de todos los fabricantes, incluso Google, que había implementado de forma nativa esta funcionalidad, la retiró en Chrome 73 (abril de 2019).

La solución que nos ofrecen desde el consorcio que promueve los Web Components es utilizar la instrucción **import** de Javascript (ES6). De esta forma se traslada la importación, desde el ámbito declarativo del HTML, al ámbito del lenguaje de programación.

La instrucción import permite importar un módulo escrito en Javascript. Hasta ES6 la única forma que teníamos de cargar un programa JS en el navegador era por medio de la etiqueta script. Con la implementación de ES6, que ya tienen todos los navegadores modernos, podemos utilizar import para cargar otro programa.

```
<script type="module" src="./modules/my-module.js">
```

También se puede utilizar de esta forma:

```
<script type="module">
  import MyComponent from './modules/my-module.js';
  customElements.define('my-component', MyComponent);
</script>
```

Ejemplo más completo paso a paso:

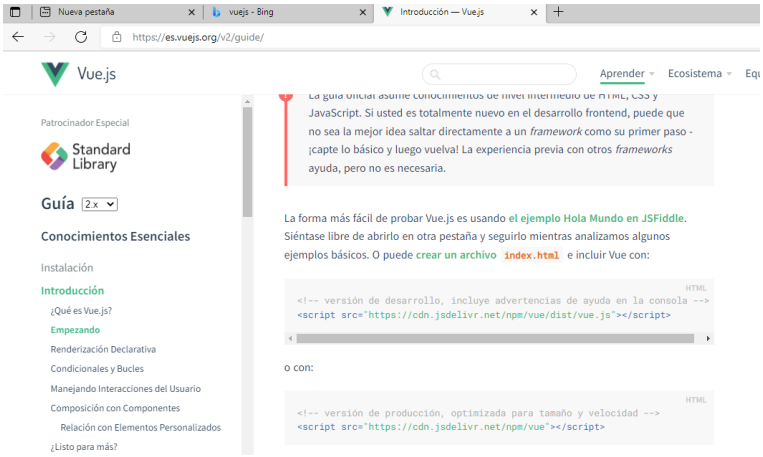
https://carlosazaustre.es/como-crear-webcomponent-de-forma-nativa
Vídeo (17:15)

2.3. Vuejs

Para empezar a trabajar con vue vamos a desarrollar la versión más simple del típico *hola mundo* de dos formas diferentes

- Crearemos un proyecto vacío y posteriormente en un fichero html introduciremos el código necesario:

  1. Un enlace a la librería vuejs a través de una etiqueta script (igual que para trabajar con jquery), obtenida desde la documentación oficial de vuejs.



  2. La creación de una instancia de vue.
     Información sobre la instancia vue https://es.vuejs.org/v2/guide/instance.html
     El constructor puede contener datos, una plantilla, el elemento donde montar la aplicación, métodos, callbacks para el ciclo de vida, etc...

```
1
2   const app = new Vue({
3           el: '#saludo',
4           data: {
5               mensaje: 'Hola Mundo v1 !'
6           }
7       })
```

     Código completo:

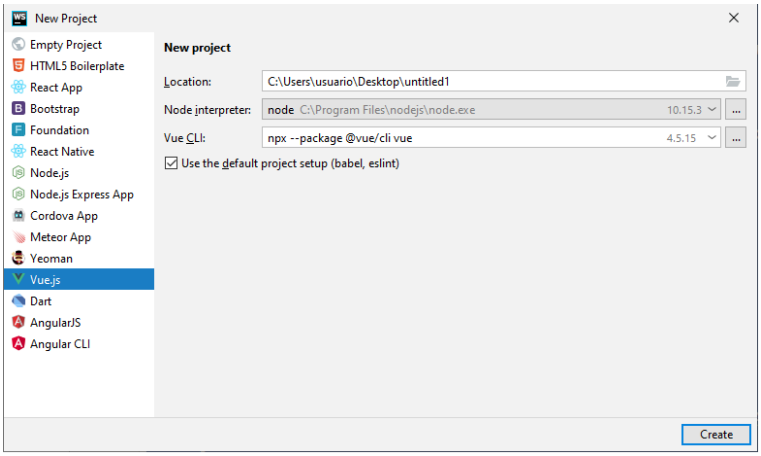
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>

<div id="saludo">
    {{ mensaje }}
</div>

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

<script>
    // Crear una instancia de vue
    const app = new Vue({
        el: '#saludo',
        data: {
            mensaje: 'Hola Mundo !'
        }
    })
</script>

</body>
</html>
```

La doble llave es la manera de acceder a los datos de la instancia de vue.

Ejemplos varios (de más sencillo a más complejo) https://tecnops.es/2-vue-js-varios-ejemplos-sencillos-vue/

Ejemplo más completo (factura) https://anexsoft.com/ejemplo-tutorial-practico-de-vu contiene un vídeo de 32:07

- Creando un proyecto vuejs



Explicado más adelante

2.4. Directivas

https://blog.eperedo.com/2019/12/25/directives-vue-js/

Una directiva es un atributo especial para HTML, por parte de Vue, que nos permite realizar una serie de operaciones. Las directivas comienzan por v-.

v-for Es una directiva que nos permite iterar sobre un array o conjunto de datos definido en una prop, data o computed, y repetir todo el código HTML dentro de esta directiva.

```
data: {
    tareas: [
        'Tarea uno',
        'Tarea dos',
        'Tarea tres'
    ]
}


<ul>
    <li v-for="tarea in tareas">{{ tarea }}</li>
</ul>
```

v-if Es la forma de agregar condicionales a nuestros templates, de tal forma que si la condición de v-if es cierta, renderizará el código dentro de este bloque.

```
<div v-if="vue.years < 2">
    ...
</div>
```

v-else

v-show El comportamiento de v-show es parecido al de v-if. En este caso, el bloque solo se muestra si la condición es cierta, pero al contrario que ocurre con v-if, en este caso si se renderiza, lo que ocurre, es que aparece oculto en el HTML con un display: none;.

v-on Esta directiva nos permite escuchar escuchar eventos javascript y actuar dependiendo del evento que se produzca.

v-bind Con esta directiva podremos vincular un atributo html a un valor que tenemos en nuestro modelo. Similar a la doble llave.

Custom directives También podemos crear nuestras propias directivas.

```

Vue.directive('demo', function (...) {

})

```

```
6
7 <div v-demo="{ ... }"></div>
```

2.5. Componentes en vue

Las arquitecturas web basadas en componentes es lo que más se emplea hoy en día en el desarrollo web. Vue.js también nos permite crear componentes y montar nuestra aplicación con ellos.

Una de las maneras de crear componentes hace uso de la función **Vue.component(tagName, options)** que recibe como parámetros un string con el nombre de nuestro componente, que será el nombre que tendrá el elemento en el DOM, y un objeto con las opciones de configuración que contendrá el nombre de las props, el template, datos, métodos, etc...

Vue no te obliga a utilizar las reglas de W3C para nombres de etiquetas personalizadas (todo en minúscula, con un guión medio) aunque seguir esta convención es considerado una buena práctica.

El contenido de un componente, es decir, las opciones pueden ser los siguientes elementos:

https://es.vuejs.org/v2/guide/components.html

- **template (plantilla)**

  La parte HTML del componente.

- **data**

  La propiedad data es una función que devuelve un objeto. Son los valores iniciales del componente.

- **Propiedades computadas**

  Funciones que realizan operaciones con los datos.

- **props**

  La propiedad props permite pasar datos de un componente padre a un componente hijo. Los hijos se comunican con los padres **emitiendo eventos**.

  Vienen a ser los atributos de las etiquetas.

- **métodos**

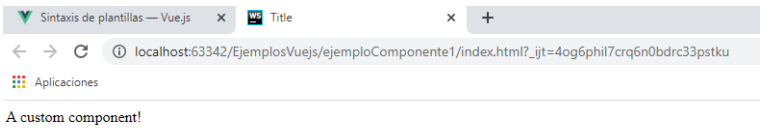
  Para crear métodos hay que poner dentro del componente un objeto llamado methods. Dentro, creamos las funciones, es decir, tantos métodos como queramos separados por comas.

- ....

2.5.1. Ejemplos

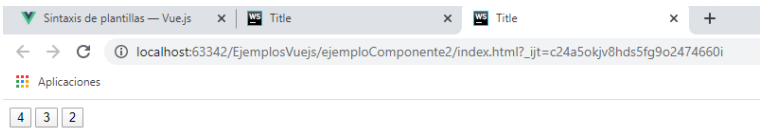
Ejemplo uno:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>

<div id="example">
    <my-component></my-component>
</div>

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script>
    // registro
    Vue.component('my-component', {
        template: '<div>A custom component!</div>'
    })
    // crear la instancia principal
    new Vue({
        el: '#example'
    })
</script>

</body>
</html>
```



Ejemplo dos:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>

<div id="example-2">
    <simple-contador></simple-contador>
    <simple-contador></simple-contador>
    <simple-contador></simple-contador>
</div>

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

    Vue.component('simple-contador', {
        template: '<button v-on:click="contador += 1">{{ contador
            }}</button>',

        data: function () {
         var objeto= {contador: 0}
         return objeto;

        }
    })

    new Vue({
        el: '#example-2'
    })

</script>

</body>
</html>
```



data es un función que devuelve un objeto distinto cada vez que se instancia. En el constructor de vue no es una función porque sólo se ejecuta una vez.

Podemos utilizar **la directiva v-on** para escuchar eventos del DOM y ejecutar JavaScript cuando son disparados.

Versión dos (Función sumar)

```
Vue.component('simple-contador', {
        template: '<button v-on:click="fsumar">{{ contador }}</button>',

        data: function () {
         var objeto= {contador: 0}
         return objeto;
        },

        methods: {
            fsumar: function (event) {
                if (event) {
                    alert(event.target.tagName)
                }
               this.contador += 1;
            }
        }
    })
```

Ejemplo tres:

El css está en la etiqueta style del index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <style>
        @import url('https://fonts.googleapis.com/css?family=Sahitya');

        body{
            font-family: 'Sahitya' , serif;
        }

        .calc-button{
            border-radius: 50%;
            padding : 20px;
            color : white;
            width : 20px;
            height : 20px;
            text-align: center;
            position : relative;
            display : inline-block;
            margin-right: 10px;
            cursor: pointer;

            -webkit-user-select: none; /* Chrome/Safari */
            -moz-user-select: none; /* Firefox */
            -ms-user-select: none; /* IE10+ */

            /* Rules below not implemented in browsers yet */
            -o-user-select: none;
            user-select: none;
        }

        .base-button{
            background-color:  rgba(155, 89, 182,1.0);
        }

        .pow-button{
            background-color: rgba(39, 174, 96,1.0);
        }

        .result-button{
            background-color: rgba(44, 62, 80,1.0);
        }

        .calc-button span{
            position : absolute;
            left : 50%;
            top : 50%;
            transform : translate( -50% , -50%);
            font-size : 1.5em;
        }

        .container{

            width : 50%;
            padding-top : 100px;
```

```
58              margin : auto;
59          }
60      </style>
61  </head>
62  <body>
63
64  <div class="container">
65  </div>
66
67  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
68  <script>
69      /*
70      Una calculadora basada en componentes, donde un botón representa la
            base de una potencia, otro el exponente, y un tercero en
            resultado de la operación.... */
71
72      Vue.component('calculator' , {
73          template : `<div class="container">
74                          <div class="calculator">
75                              Base<base-button :value="this.base"
                                    @incrementBase="incrementBaseValue" />
76                              Exponente <exp-button :value="this.exponente"
                                    @incrementExp="incrementExpValue"/>
77                              Resultado <result-button
                                    :value="this.resultValue"/>
78                          </div>
79                  </div>`,
80
81
82          data : function(){
83              return {
84                  base : 0,
85                  exponente : 0
86
87              }
88          },
89          computed : {
90              resultValue : function(){
91                  return Math.pow(this.base , this.exponente);
92              }
93          },
94          methods : {
95              incrementBaseValue : function(){
96                  this.base++;
97
98              },
99              incrementExpValue : function(){
100                 this.exponente++;
101             }
102         }
103     });
104
105
106     Vue.component('baseButton', {
107         template : '<div class="calc-button base-button"
                v-on:click="fSumar"><span>{{value}}</span></div>',
108         props : ['value'],
109
110         methods : {
```

```
111             fSumar : function(){
112                 this.$emit('incrementBase');
113             }
114         }
115
116     });
117
118     Vue.component('expButton', {
119         template : '<div class="calc-button pow-button"
                v-on:click="fSumar"><span>{{value}}</span></div>',
120         props : ['value'],
121         methods : {
122             fSumar : function(){
123                 this.$emit('incrementExp');
124             }
125         }
126     });
127
128     Vue.component('resultButton', {
129         template : '<div class="calc-button
                result-button"><span>{{value}}</span></div>',
130         props : ['value'],
131
132     });
133
134     new Vue({
135         el : '.container',
136         template : '<calculator/>'
137     })
138
139 </script>
140
141 </body>
142 </html>
```

Ejemplo cuatro (Varios ficheros)

Contenido del fichero app.js

```
Vue.component('datos-pelicula', {
    props: ['imagen', 'titulo'],
    template: `
    <div>
      <img width="100" v-bind:src="imagen" v-bind:alt="titulo"/>
      <h2>{{ titulo }}</h2>
    </div>
  `,
})

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

Contenido del fichero index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Componentes con Vue.js</title>
</head>
<body>
<div id="app">
    <datos-pelicula
            titulo="Regreso al Futuro"
            imagen="http://es.web.img3.acsta.net/pictures/14/04/03/13/45/034916.jpg">
    </datos-pelicula>
</div>
<script
    src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.4.2/vue.js"></script>
<script src="app.js"></script>
</body>
</html>
```


Sintaxis de plantillas — Vue.js
Title
localhost:63342/EjemplosVuejs/ejemploP
Aplicaciones
Regreso al Futuro

2.6. Componentes .vue (Single File Components)

En muchos proyectos los componentes son definidos utilizando Vue.component, seguido de new Vue como en los ejemplos vistos hasta ahora.

Esto puede funcionar bien en proyectos pequeños a medianos pero en proyectos más complejos existen desventajas que se solucionan con ficheros .vue que nos permiten tener en un único fichero la vista (template), el diseño (css) y la lógica (javascript). Estos ficheros tienen una extensión .vue y tienen la siguiente forma:

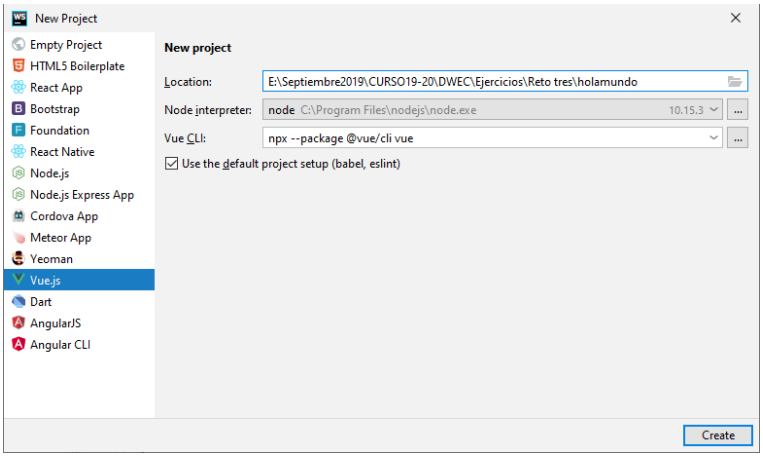
```
<template>
    <!-- Marcado HTML -->
</template>

<script>
    /* Código JavaScript */
</script>

<style>
    /* Estilos CSS */
</style>
```

```
<template>
    <div>
        <img width="100" v-bind:src="imagen" v-bind:alt="titulo"/>
        <h2>{{ titulo }}</h2>
    </div>
</template>

<script>
    export default {
        name: 'DatosPelicula',
        props: {
            imagen: String,
            titulo: String
        }
    }
</script>

<style scoped>
    h2 {
        font-size: 18pt;
    }
</style>
```

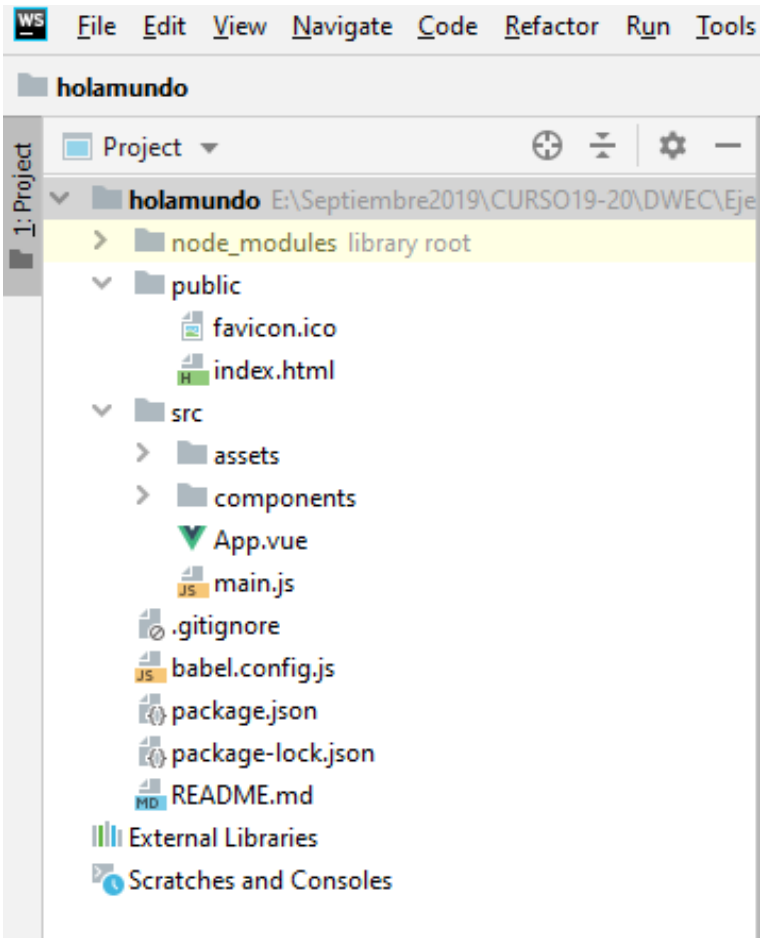
2.7. Estructura de un proyecto vue creado con WebStorm

Tu primera aplicación con vue.js https://www.youtube.com/watch?v=iENgabVQSYY (1:08:18)

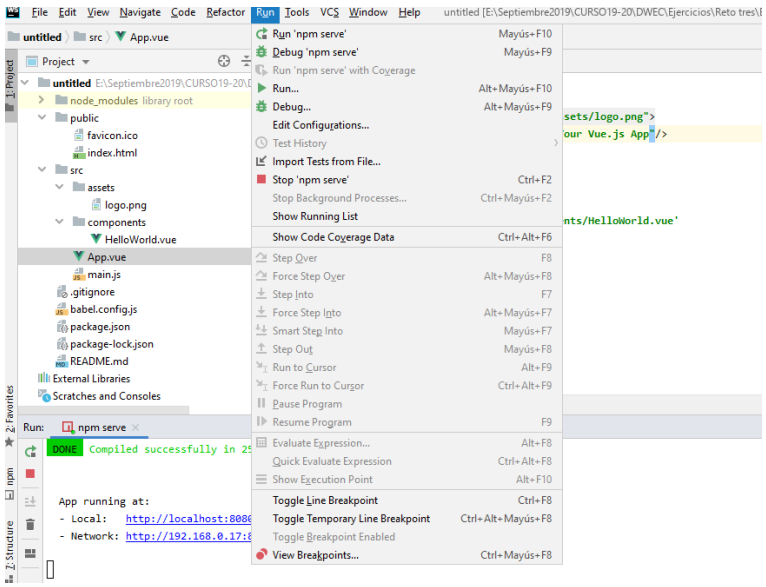
Desde el IDE WebStorm vamos a crear un nuevo proyecto con Vue.js llamado holamundo.



Después de un buen rato y la instalación de manera automática de los paquetes necesarios, se crea el proyecto.


File Edit View Navigate Code Refactor Run Tools
holamundo
1: Project
Project
holamundo E:\Septiembre2019\CURSO19-20\DWEC\Eje
node_modules library root
public
favicon.ico
index.html
src
assets
components
App.vue
main.js
.gitignore
babel.config.js
package.json
package-lock.json
README.md
External Libraries
Scratches and Consoles

Tras arrancar el servidor podremos ejecutarlo.



Los principales elementos de la estructura del proyecto son:

1. **Carpeta node-modules**: Como en todo proyecto javascript que utilice los paquetes NPM, aquí también existe esta carpeta en la que se encuentran todas las dependencias ya instaladas del proyecto.

2. **Carpeta public**: En esta carpeta hay dos archivos:

   favicon.ico: Es el icono que aparece al lado del nombre de la pestaña en el navegador, por defecto es el logo de Vue pero lo puedes cambiar por tu icono en formato .ico

   index.html: Es el archivo html sobre el que se montará toda la página. Toda la página se incrustará dentro de la etiqueta:

```
1 <div id="app"></div>
```

3. **Carpeta src**: En la carpeta src se sitúan los archivos fuentes de nuestra aplicación.

   assets: En esta carpeta se encuentran todos los archivos estáticos de la aplicación. Cuando creas la estructura se añade el archivo logo.png de ejemplo. Cuando quieras meter una imagen, fuentes, o cosas así lo puedes hacer en esta carpeta. Vue y webpack se encargarán de servir los archivos de esta carpeta para que los puedas llamar.

components: Aquí se crearan todos los componentes web. Al crear el proyecto te viene el componente HelloWorld.vue de ejemplo. Lo más recomendable es que dentro de esta carpeta crees carpetas para separar los componentes entre sí. Lo que se suele hacer es separar los componentes por páginas o por área a la que pertenece.

main.js: Este archivo va a ser el nuevo archivo principal donde se va a instanciar vue. En la primera línea del archivo cargamos e importamos a Vue (ya no hace falta la etiqueta script). En la segunda línea se importa otro componente (App) que luego se carga en nuestra aplicación mediante una Render.

App.vue: El componente App es el archivo App.vue. Las extensiones .vue son un tipo de archivo con el que vamos a trabajar de ahora en adelante y es en estos archivos donde se definirá todo el código de la aplicación, no sólo JavaScript sino también las plantillas de Vue e incluso CSS. Sin embargo en lugar de crear un componente global dentro del archivo, exportamos un objeto donde vamos a tener los datos, métodos, computed properties, etc...

Al pasar el código de nuestra aplicación a App.vue ya no necesitamos hacer referencia al elemento app ya que el componente App.vue se cargará a través de la función render en el archivo main.js.

4. **Otros archivos**: Aparte de las carpetas que acabamos de ver, Vue crea estos archivos.

   .gitignore: Aquí le dices a git qué ficheros quieres que ignore. Por defecto Vue habrá colocado los recomendados.

   babel.config.js: Configuraciones de babel, la herramienta que transpila el javascript generado a código entendible por navegadores antiguos.

   package.json y package-lock.json: Donde se establecen las versiones de la librerías requeridas. Además se pueden establecer otros parámetros como el nombre de la aplicación, versión, autores, etc.

   README.md: Archivo que se muestra al entrar en un proyecto en el github o gitlab. Aquí puedes escribir información sobre el proyecto que creas conveniente.