Aprendiendo VUE con las clases de bluueweb en youtube

# Clase 1: Introducción

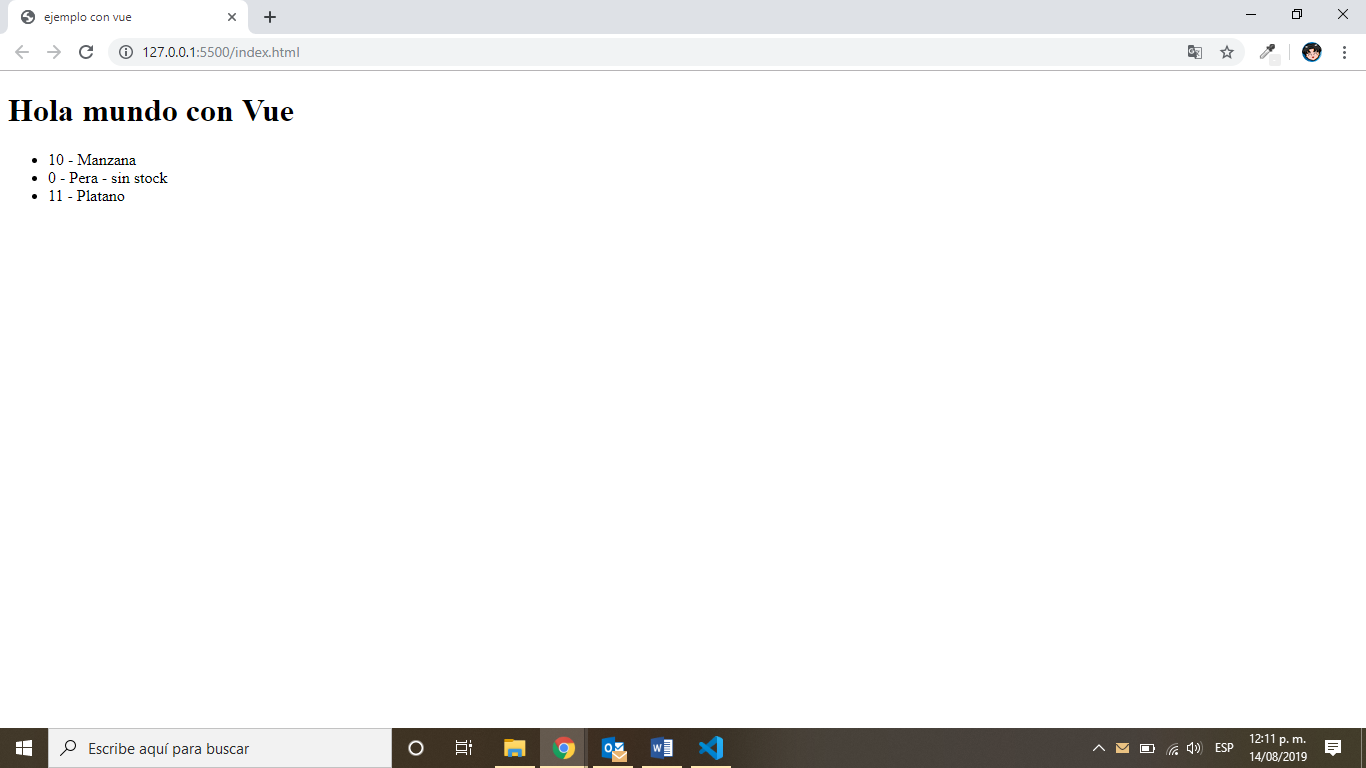
1. Para comenzar a usar vue copiamos el cdn para insértalo en nuestro html

<!-- development version, includes helpful console warnings -->

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

1. Tenemos que cerrar todo lo de vue en un container dentro del index.html
   1. <div id="app">
   2. <h1>{{titulo}}</h1>
   3. </div>
2. Para instanciar Vue dentro del archivo js/01.js
   1. const app = new Vue({
   2. el: '#app',
   3. data: {
   4. titulo: 'Hola mundo con Vue',
   5. frutas: [
   6. {nombre:'Manzana', cantidad:10},
   7. {nombre:'Pera', cantidad:0},
   8. {nombre:'Platano', cantidad:11}
   9. ]
   10. }
   11. })
3. El index.html quedaría de esta forma:
4. <!DOCTYPE html>
5. <html lang="en">
6. <head>
7. <meta charset="UTF-8">
8. <meta name="viewport" content="width=device-width, initial-scale=1.0">
9. <meta http-equiv="X-UA-Compatible" content="ie=edge">
10. <title>ejemplo con vue</title>
11. </head>
12. <body>
14. <div id="app">
15. <h1>{{titulo}}</h1>
16. <!-- COmo esun array debemos usar un siclo for y lo ponemos en una lista -->
17. <ul>
18. <li v-for="fruta of frutas">
19. {{fruta.cantidad}} - {{fruta.nombre}}
20. <span v-if="fruta.cantidad === 0"> - sin stock</span>
21. </li>
22. </ul>
23. </div>
24. <!-- development version, includes helpful console warnings -->
25. <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
26. <script src="js/01.js"></script>
27. </body>
28. </html>

Así quedaría la vista:



# Clase 2. Methods, v-model y evento click

1. Vamos ausar la etiqueta imput que es de los formularios
2. Para que el tecto sea detectdo por vue vamos a usar el v-model, con eso va ha existir un intercambio de datos desde nuestro index hacia nuestro javascript

<input type="text" v-model="nuevaFruta">

1. Ahora ben para agregarlo al array que tenemos en el index, recuerda que aparte del “el” y aparte del “data” tenemos Metódos. Dentro de esto vamos a crear un objeto y ponemos una función.

methods:{

agregarFruta () {

}

1. Vamos a probar todo esto haciendo un conole.log

console.log('diste click');

1. Ahora para conocer los eventos vamos a poner un botón en el index que diga agregar y ponemos el método de Click usando la @.

<button @click="agregarFruta">Agregar</button>

1. Ahora vamos a preparer todo para agregar la nueva fruta en nuestro array usando frutas.push(), literalmente lo que estamos haciendo es empujar la nueva fruta al array

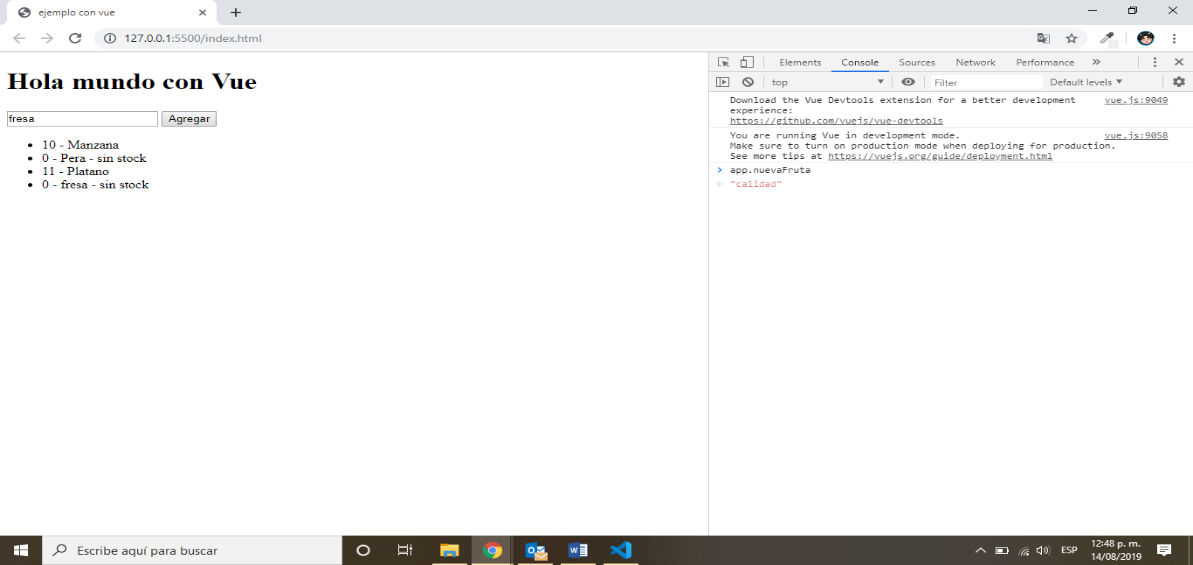
agregarFruta () {

this.frutas.push({

nombre: this.nuevaFruta, cantidad: 0

})

}



# Clase 3. Key-up, v-model y Computed

1. Para limpiar el contenido de nuestro imput vamos a sobre escribir la variable. Fuera de nuestro push() vamos a poner nuevaFruta = “”;

this.nuevaFruta = "";

1. Si queremos que al presionar el botón enter se ejecute la función agregarFruta podemos hacer lo siguiente dentro de nuestro input en el index.html

<input type="text" v-model="nuevaFruta" @keyup.enter="agregarFruta">

1. Para poder midificar el número de stock de cada fruto tenemos que transformar en imput el dato de fruta.cantidad

<input type="number" v-model="fruta.cantidad">

1. Si yo modifico la cantidad de la fruta que dice “sin stockautomaticamente desaparece esa etiqueta, sin embargo, si lo regreso a cero sigue sin aparecer, para arreglar esto. El problema recae es que el número que ponemos se refleja como string entonces para solucionar ese problema de decimios al v.model que va a ser de tipo numérico “v-model.number”

<input type="number" v-model.number="fruta.cantidad">

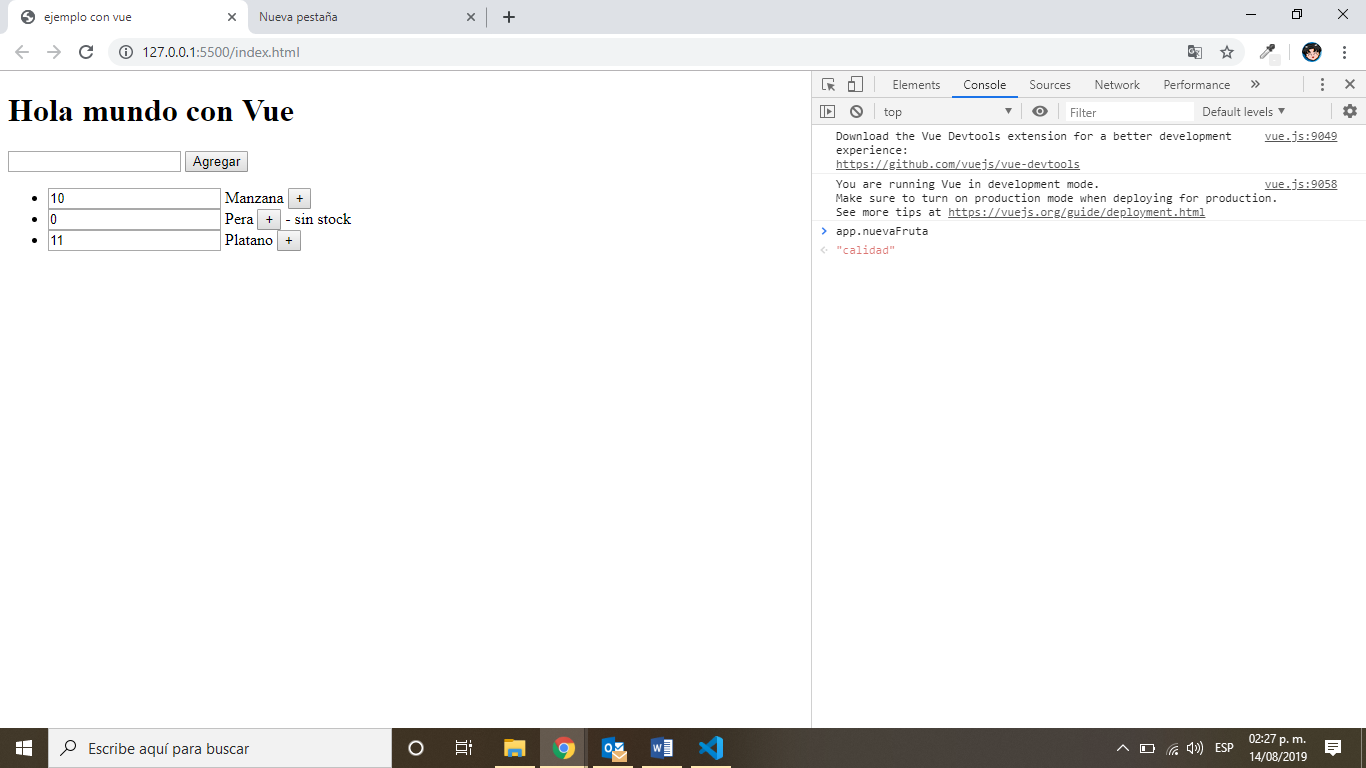
1. Ahora agregaremos un botón para poder sumar más frutas una por una de la siguiente manera

<input type="number" v-model.number="fruta.cantidad">

{{fruta.nombre}}

<button @click="fruta.cantidad++"></button>

1. Así quedaría la vista



1. Si ahora queremos sumar las cantidades, fuera del </ul> vamos aponer un h4

<h4>TOTAL: </h4>

1. Vamos a usar este nuevo elemento que se llama computed en nuestro archivo 01.js

computed: {

sumarFrutas () {

this.total = 0;

for (fruta of this.frutas){

this.total = this.total + fruta.cantidad;

}

return this.total;

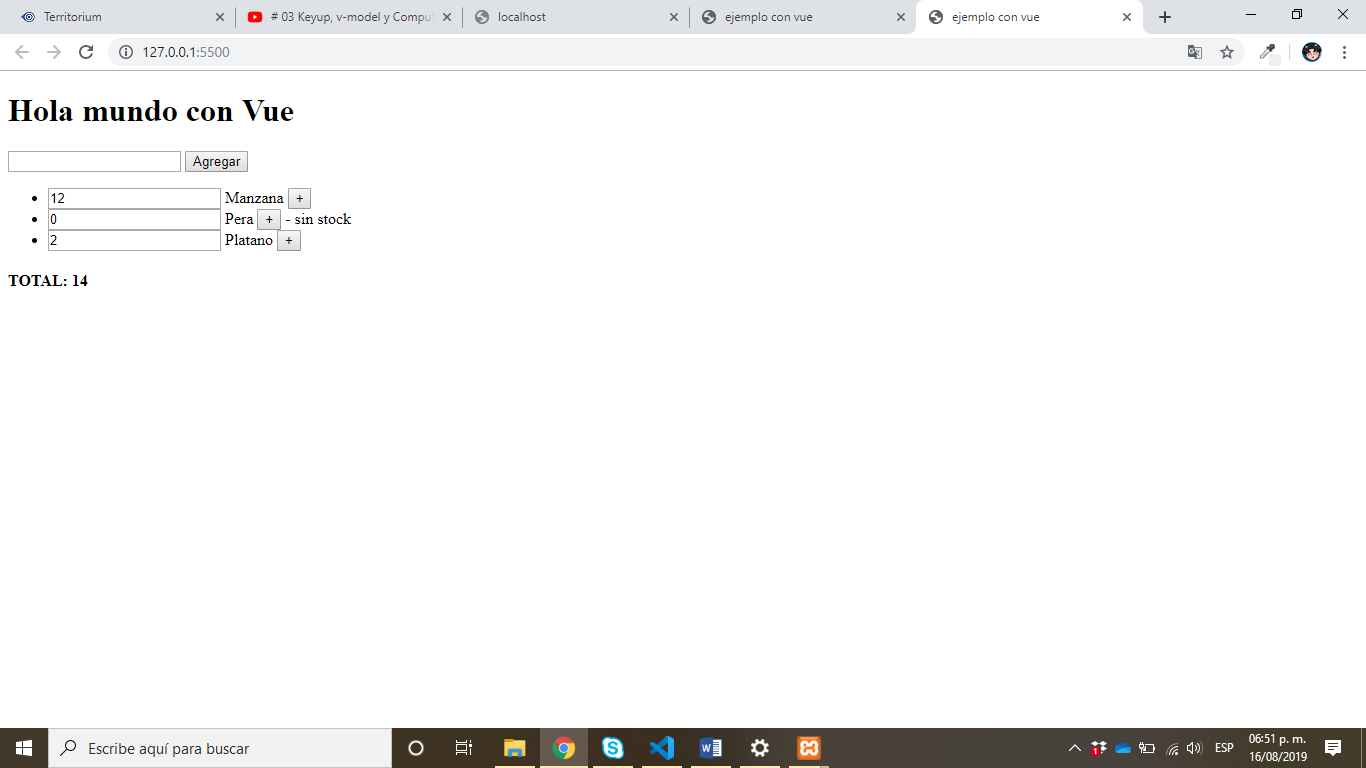
}

}

1. Cambiamos el index, solo la parte del total

<h4>TOTAL: {{sumarFrutas}} </h4>

1. Así quedaría la vista



# Clase 4. Clases (v-bind:class)

Copiamos los css de Bootstrap y vamos a trabar con vue

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

Es importante que recuerdes la clase app para poder seguir trabajando con vue, estaremos trabajando con el archivo 07.html

<div id="app" class="container mt-5">

<div class="bg-danger">

<p>Lorem ipsum dolor.</p>

</div>

</div>

Por el otro lado estaremos trabajando con javascript con su correspondiente 07.js

const app = new Vue({

el: '#app',

data: {

},

methods:{

}

})

Para poner lo mismo desde vue usamos :class=”’p-3’” para especificar las clases en vue hay que usar comillas simples dentro de las comillas dobles

class="bg-danger" :class="'p-3'"

Para poner más clases debemos hacer un array

<div class="bg-danger" :class="['p-3', 'text-white']">

Vamos a traer nuestro fondo desde vue en 07.js

const app = new Vue({

el: '#app',

data: {

fondo: 'bg-warning'

},

methods:{

}

})

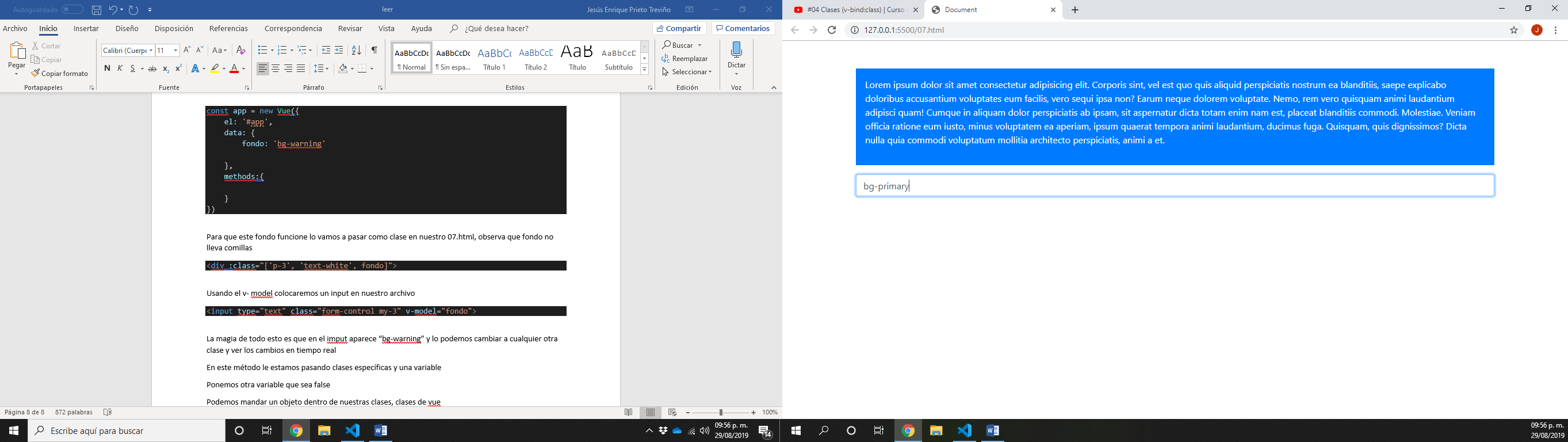
Para que este fondo funcione lo vamos a pasar como clase en nuestro 07.html, observa que fondo no lleva comillas

<div :class="['p-3', 'text-white', fondo]">

Usando el v- model colocaremos un input en nuestro archivo

<input type="text" class="form-control my-3" v-model="fondo">

En este método le estamos pasando clases específicas y una variable. La magia de todo esto es que en el imput aparece “bg-warning” y lo podemos cambiar a cualquier otra clase y ver los cambios en tiempo real



Ponemos otra variable que sea false

data: {

fondo: 'bg-warning',

color: false

},

Podemos mandar un objeto dentro de nuestras clases, clases de vue

<div class="text-white p-3" :class="{'bg-info' : color}">

Bg-info será colocoada si color es verdadero en el html

<div class="text-white p-3" :class="{'bg-info' : color, 'bg-success' : !color}">

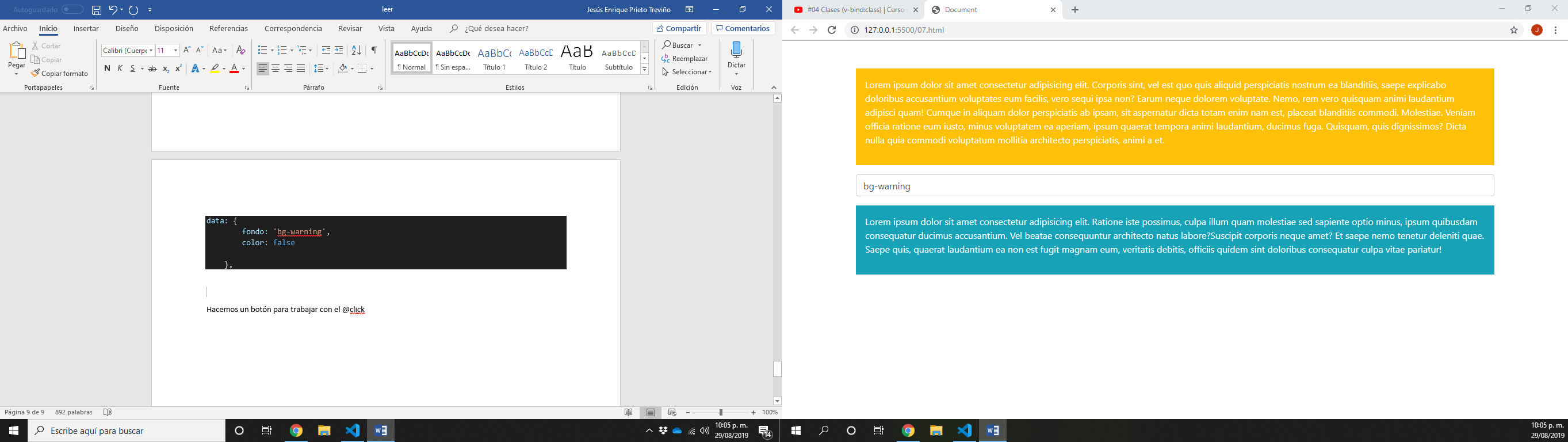
Este cambio lo hacemos en nuestro javascript

data: {

fondo: 'bg-warning',

color: false

},



Hacemos un botón para trabajar con el @click después del div para cambiar el color

<button class="btn btn-primary" @click="color = !color">cambiar</button>

El archivo final 07.html quedaría así:

<!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>Document</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

</head>

<body>

<div id="app" class="container mt-5">

<div :class="['p-3', 'text-white', fondo]">

<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Corporis sint, vel est quo quis aliquid perspiciatis nostrum ea blanditiis, saepe explicabo doloribus accusantium voluptates eum facilis, vero sequi ipsa non?

Earum neque dolorem voluptate. Nemo, rem vero quisquam animi laudantium adipisci quam! Cumque in aliquam dolor perspiciatis ab ipsam, sit aspernatur dicta totam enim nam est, placeat blanditiis commodi. Molestiae.

Veniam officia ratione eum iusto, minus voluptatem ea aperiam, ipsum quaerat tempora animi laudantium, ducimus fuga. Quisquam, quis dignissimos? Dicta nulla quia commodi voluptatum mollitia architecto perspiciatis, animi a et.</p>

</div>

<input type="text" class="form-control my-3" v-model="fondo">

<div class="text-white p-3" :class="{'bg-info' : color, 'bg-success' : !color}">

<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Ratione iste possimus, culpa illum quam molestiae sed sapiente optio minus, ipsum quibusdam consequatur ducimus accusantium. Vel beatae consequuntur architecto natus labore?Suscipit corporis neque amet? Et saepe nemo tenetur deleniti quae. Saepe quis, quaerat laudantium ea non est fugit magnam eum, veritatis debitis, officiis quidem sint doloribus consequatur culpa vitae pariatur!</p>

</div>

<button class="btn btn-primary" @click="color = !color">cambiar</button>

</div>

<!-- development version, includes helpful console warnings -->

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

<script src="js/07.js"></script>

</body>

</html>

Y el archivo final de 07.js quedaría así:

const app = new Vue({

el: '#app',

data: {

fondo: 'bg-warning',

color: false

},

methods:{

}

})

# Clase 5 Propiedades computadas (computed)

Las propiedades computadas también se pueden desarrollar a través de métodos, pero existe una gran diferencia.

Las propiedades computadas son cacheadas es decir se quedan en el caché del navegador es una gran ventaja porque reduce la velocidad de carga del sitio

En los datos vamos a poner el dato mensaje dentro de nuestro archivo js

data:{

mensaje: "hola soy bluue web"

}

Lo llamamos desde el html

<h3>{{ mensaje }}</h3>

Para poder ver el mensaje lo vamos a invertir en el html

<h3>invertido: {{ invertido }}</h3>

Cuando trabajas con propiedades computadas solamente debes poner el nombre dentro del computer desde el archivo en js

computed:{

invertido(){

return this.mensaje;

}

Ahora vamos a devolver el mensaje separado con la propiedad Split

computed:{

invertido(){

return this.mensaje.split('';

}

Ahora lo vamos a invertir

computed:{

invertido(){

return this.mensaje.split('').reverse;

}

Una vez invertido vamos juntar todo con la propiedad de join

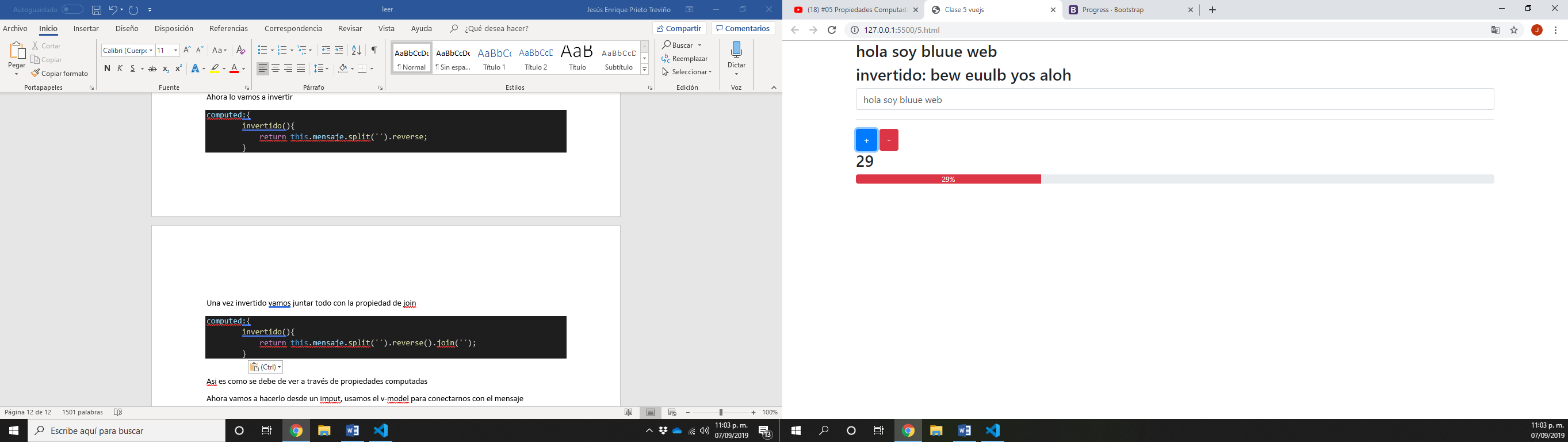
computed:{

invertido(){

return this.mensaje.split('').reverse().join('');

}

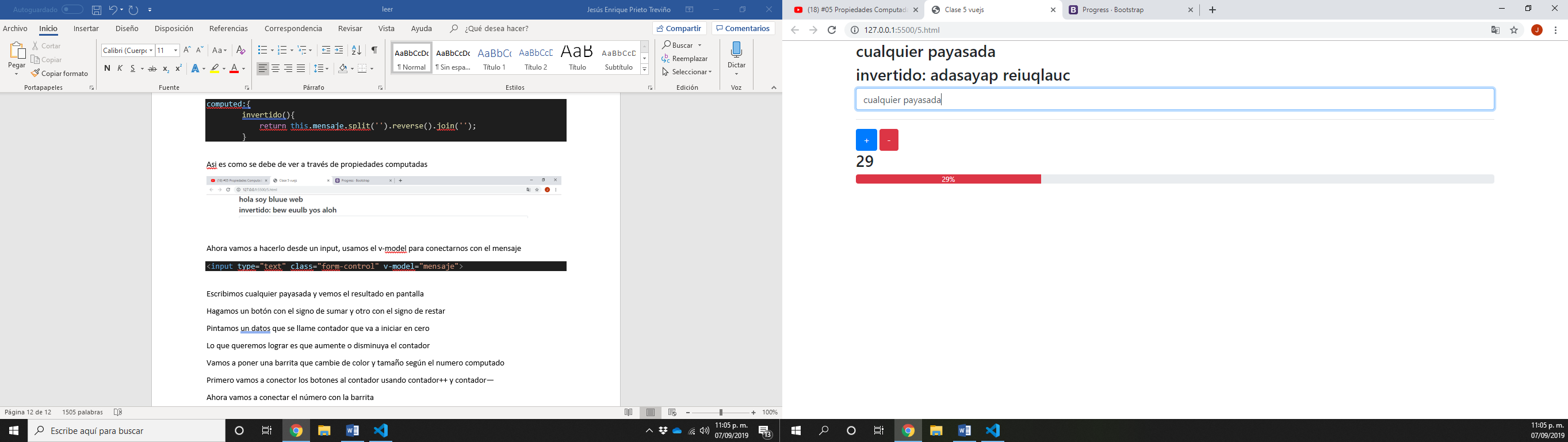
Asi es como se debe de ver a través de propiedades computadas



Ahora vamos a hacerlo desde un input, usamos el v-model para conectarnos con el mensaje

<input type="text" class="form-control" v-model="mensaje">

Escribimos cualquier payasada y vemos el resultado en pantalla



Hagamos un botón con el signo de sumar y otro con el signo de restar en el html

<button class="btn btn-primary"> + </button>

<button class="btn btn-danger"> - </button>

Pintamos un dato que se llame contador que va a iniciar en cero en el data del js

data:{

mensaje: "hola soy bluue web",

contador: 0

}

Lo que queremos lograr es que aumente o disminuya el contador

Vamos a poner una barrita que cambie de color y tamaño según el numero computado

Primero vamos a conector los botones al contador usando contador++ y contador-- en el html usando la propiedad @click

<button class="btn btn-primary" @click="contador++"> + </button>

<button class="btn btn-danger" @click="contador--"> - </button>

Ahora vamos a conectar el número con la barrita

<div class="progress">

<div class="progress-bar" role="progressbar" style="width : 25%"

aria-valuenow="25" aria-valuemin="0" aria-valuemax="100" :class="color"> {{ contador }}% </div>

</div>

La propiedad que cambia la barra es el style por lo tanto lo vamos a trabajar con vue :style=”” todo esto lo seguimos trabajando dentro del documento html

<div class="progress">

<div class="progress-bar" role="progressbar" :style="{'width': contador+'%'}"

aria-valuenow="25" aria-valuemin="0" aria-valuemax="100" :class="color"> {{ contador }}% </div>

</div>

Ahora con computer quiero que cambie de color pasando el 10 y el 20 % desde las clases y la propiedad color desde nuestras propiedades computadas vamos retornar un objeto usando las clases para el botón succes, warning y danger. Esto es en archivo js poniendo otra propiedad computada llamada color

computed:{

invertido(){

return this.mensaje.split('').reverse().join('');

},

color(){

return {

'bg-success' : this.contador <= 10,

'bg-warning' : this.contador > 10 && this.contador < 20,

'bg-danger' : this.contador >=20

}

}

}

En este punto todo ya debe de estar funcionando de maravilla pero si te perdiste de algo te dejo el código dentro del body del archivo5.html

<div id="app" class="container">

<h3>{{ mensaje }}</h3>

<h3>invertido: {{ invertido }}</h3>

<input type="text" class="form-control" v-model="mensaje">

<hr>

<button class="btn btn-primary" @click="contador++"> + </button>

<button class="btn btn-danger" @click="contador--"> - </button>

<h3>{{ contador }}</h3>

<div class="progress">

<div class="progress-bar" role="progressbar" :style="{'width': contador+'%'}"

aria-valuenow="25" aria-valuemin="0" aria-valuemax="100" :class="color"> {{ contador }}% </div>

</div>

</div>

Y el código completo del archivo 5.js

const app = new Vue({

el: '#app',

data:{

mensaje: "hola soy bluue web",

contador: 0

},

computed:{

invertido(){

return this.mensaje.split('').reverse().join('');

},

color(){

return {

'bg-success' : this.contador <= 10,

'bg-warning' : this.contador > 10 && this.contador < 20,

'bg-danger' : this.contador >=20

}

}

}

});

# Clase 6. Ciclo de vida de vue

Vamos a ver algo un poquito más teórico

Vue cuando creamos una instancia ésta se crea

Vamos a hacer un ejercicio sencillo para ver la parte del ciclo de vida

Vamos a pintar un saludito en pantalla usando Vue desde el html

<div id="app" class="container">

        <h3> {{ saludo }} </h3>

        <button @click="saludo = 'saludo cambiado'">cambiar</button>

        <button @click="destruir()">destruir</button>

    </div>

1. beforeCreate
   1. este se ejecuta en el momento en que se crea vue

beforeCreate(){

        console.log('beforeCreate');

    },

1. created
   1. al crear lo métodos, observadore y eventos por aún no accede al DOM aún no se accede al ‘el’

created(){

        // Al crear los métodos, observadores y eventos, pero aún no accede al DOM.

        // Aún no se accede a 'el'

        console.log('created');

    },

1. beforeMount
   1. se ejecuta antes de insertar el DOM

beforeMount(){

        // Se ejecuta antes de insertar el DOM

        console.log('beforeMounted');

    },

1. mounted
   1. Se ejecuta al insertar el DOM

mounted(){

        // Se ejecuta al insertar el DOM

        console.log('mounted');

    },

1. beforeUpdate
   1. Se ejecuta al detectar un cambio

beforeUpdate(){

        // Al detectar un cambio

        console.log('beforeUpdate')

    },

1. Updated
   1. Se ejecuta al realizar los cambios

updated(){

        // Al realizar los cambios

        console.log('updated');

    },

1. beforeDestroy
   1. Se ejecuta antes de destruir la instancia

beforeDestroy(){

        // Antes de destruir la instancia

        console.log('beforeDestroy');

    },

1. Destroyed
   1. Se destruye toda la instancia

destroyed(){

        // Se destruye toda la instancia :(

        console.log('destroyed');

    },

Ahora vamos a crear la la parte dónde ejecutamos destroyed que estamos llamando con un botón desde el html con el @click. Este es el método que vamos a poner desde el archivo 6.js

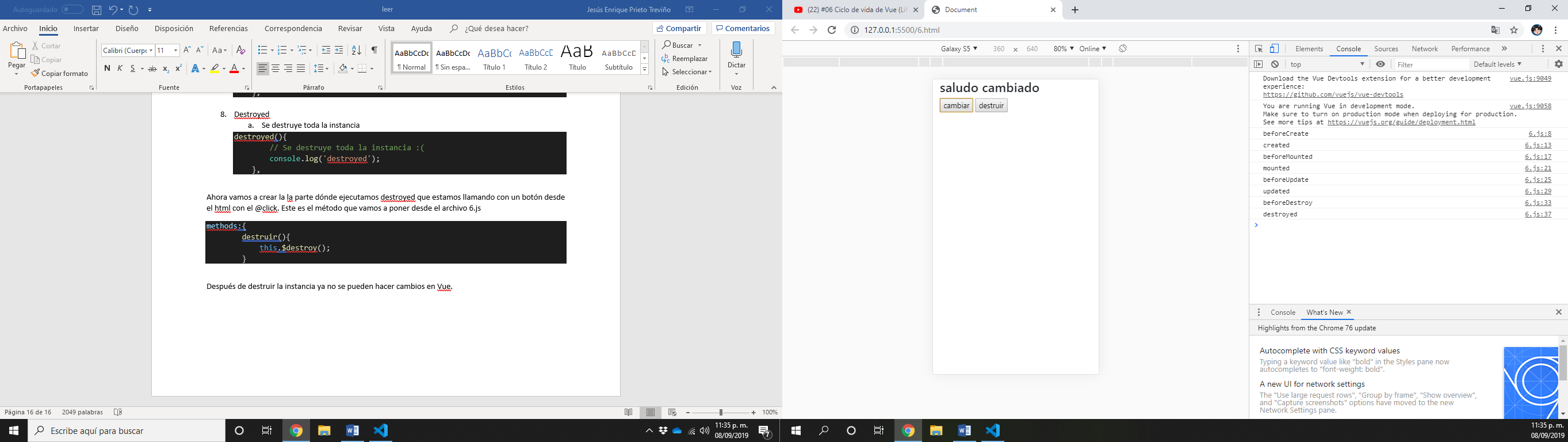
methods:{

        destruir(){

            this.$destroy();

        }

Después de destruir la instancia ya no se pueden hacer cambios en Vue.



Este es el código en el body del html

 <div id="app" class="container">

        <h3> {{ saludo }} </h3>

        <button @click="saludo = 'saludo cambiado'">cambiar</button>

        <button @click="destruir()">destruir</button>

    </div>

Y este es el código completo del archivo 6.js

const app = new Vue({

    el: '#app',

    data: {

        saludo: 'soy un ciclo de vida'

    },

    beforeCreate(){

        console.log('beforeCreate');

    },

    created(){

        // Al crear los métodos, observadores y eventos, pero aún no accede al DOM.

        // Aún no se accede a 'el'

        console.log('created');

    },

    beforeMount(){

        // Se ejecuta antes de insertar el DOM

        console.log('beforeMounted');

    },

    mounted(){

        // Se ejecuta al insertar el DOM

        console.log('mounted');

    },

    beforeUpdate(){

        // Al detectar un cambio

        console.log('beforeUpdate')

    },

    updated(){

        // Al realizar los cambios

        console.log('updated');

    },

    beforeDestroy(){

        // Antes de destruir la instancia

        console.log('beforeDestroy');

    },

    destroyed(){

        // Se destruye toda la instancia :(

        console.log('destroyed');

    },

    methods:{

        destruir(){

            this.$destroy();

        }

    }

});

# Clase 7. Componentes básicos

Los componentes son las cosas más importantes de Vue Es separar nuestro sitio web en diferentes secciones para que la lógica nos quede separada. Por ejemplo, deberíamos de hacer un componente para el navbar para evitar duplicar el código.

Vamos a crear un componente para el saludo por lo tanto en el index solamente deberíamos de poner la siguiente sintaxis.

<saludo></saludo>

Recuerdo hay que trabajar sobre la instancia de Vue para crear nuestro primer componente. Te debes grabar en la cabezota que todo componente lleva un template, por que de esta manera le podemos poner todas las etiquetas que querramos, serán muy necesarias en el caso de hacer un navbar u otro componente.

 <script>

    // componente

    Vue.component('saludo', {

        template: '<h1>Saludo de forma estática</h1>'

    });

      new Vue({

          el: '#app',

          data: {

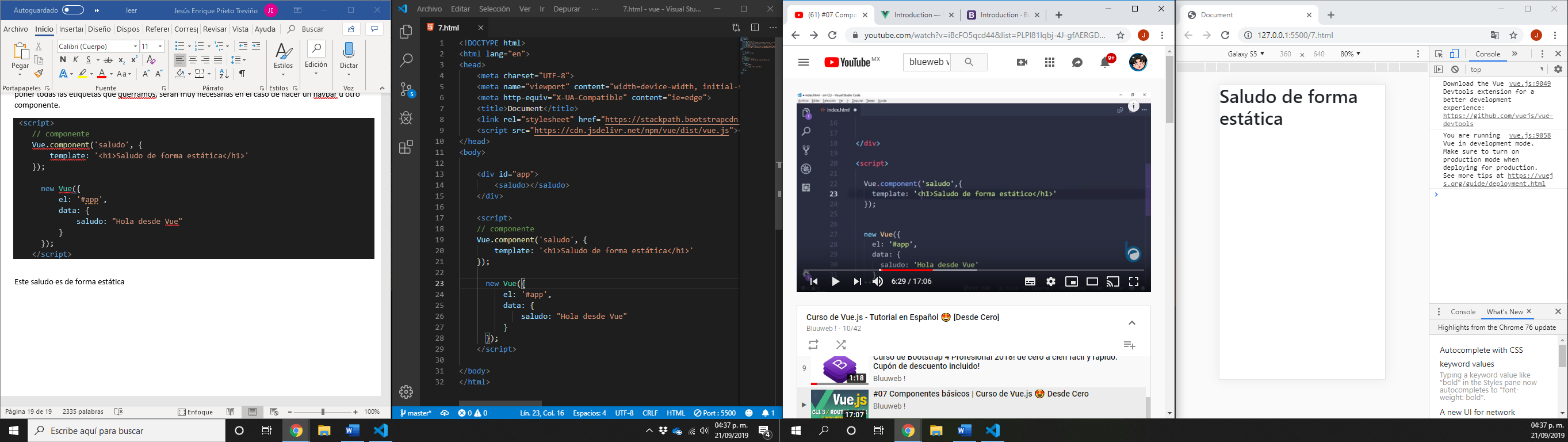
              saludo: "Hola desde Vue"

          }

      });

    </script>

Así es como se vería el saludo es de forma estática



Una de las gracias que tienen los componentes es que puedes diplcar la etiqueta y se duplicará el componente.

Para poder tener un saludo dinámico vamos a llamar a data es importante diferenciar que la data dentro de un componente es una función que tiene que retornar un objeto (la data es diferente acá Ok).

<script>

    // componente

    Vue.component('saludo', {

        template: '<h1>{{saludo}}</h1>',

        data(){

            return {

                saludo: "Hola desde Vue"

            }

        }

    });

      new Vue({

          el: '#app'

      });

    </script>

Ojo acá, las comillas simples se utilizan en una línea, es decir, si quieres escribir en varios renglones dentro del template NO puedes utilizar comilla simples. Debemos utilizar los “templates literales” esa cosa nueva de ecmascript (son unas comillas invertidas que nos permiten HTML ( ` ` ) y dentro de un div

Bueno ahora vamos a separar el index de los componentes creando una carpeta que se llame componentes y agregaremos los archivos saludo.js y contador.js

Saludo.js

Vue.component('saludo', {

    template: //html

    `

    <div>

        <h1>{{saludo}}</h1>

        <h3>fjasldkjf</h3>

    </div>

    `,

    data(){

        return {

            saludo: "Hola desde Vue"

        }

    }

});

Contador.js

Vue.component('contador', {

    template: //html

    `

        <div>

            <h3>{{numero}}</h3>

            <button @click="numero++" class="btn btn-primary">agregar</button>

        </div>

    `,

    data(){

        return{

            numero: 0

        }

    }

});

Y el index queda de la siguiente forma

Vue.component('contador', {

    template: //html

    `

        <div>

            <h3>{{numero}}</h3>

            <button @click="numero++" class="btn btn-primary">agregar</button>

        </div>

    `,

    data(){

        return{

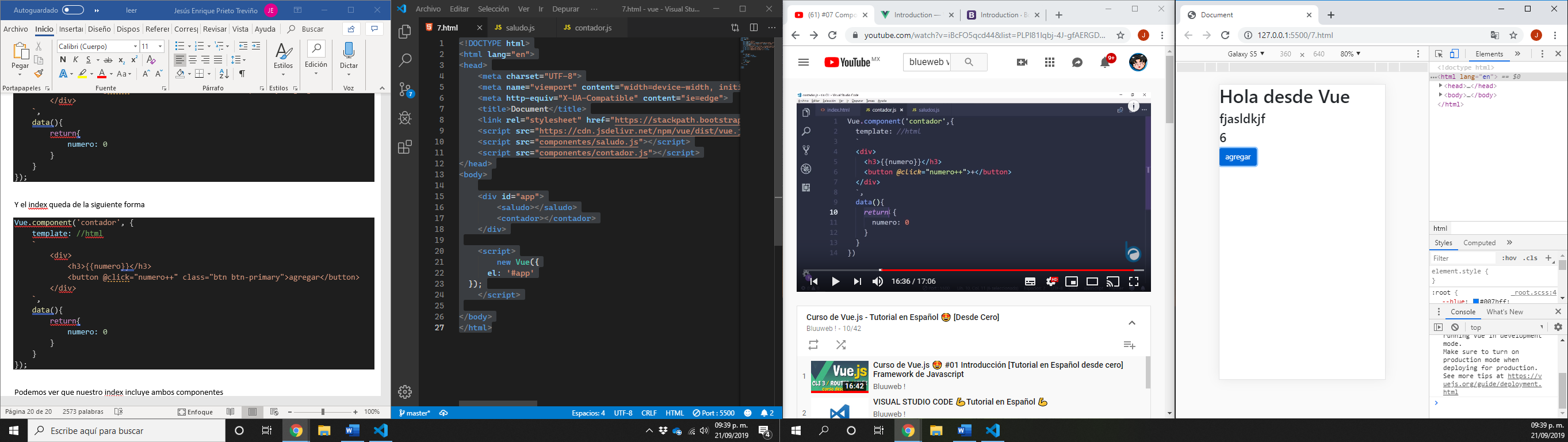
            numero: 0

        }

    }

});

Podemos ver que nuestro index incluye ambos componentes



NOTA: se pueden poner más de un dato en el data de cada componente separándolos por una coma.