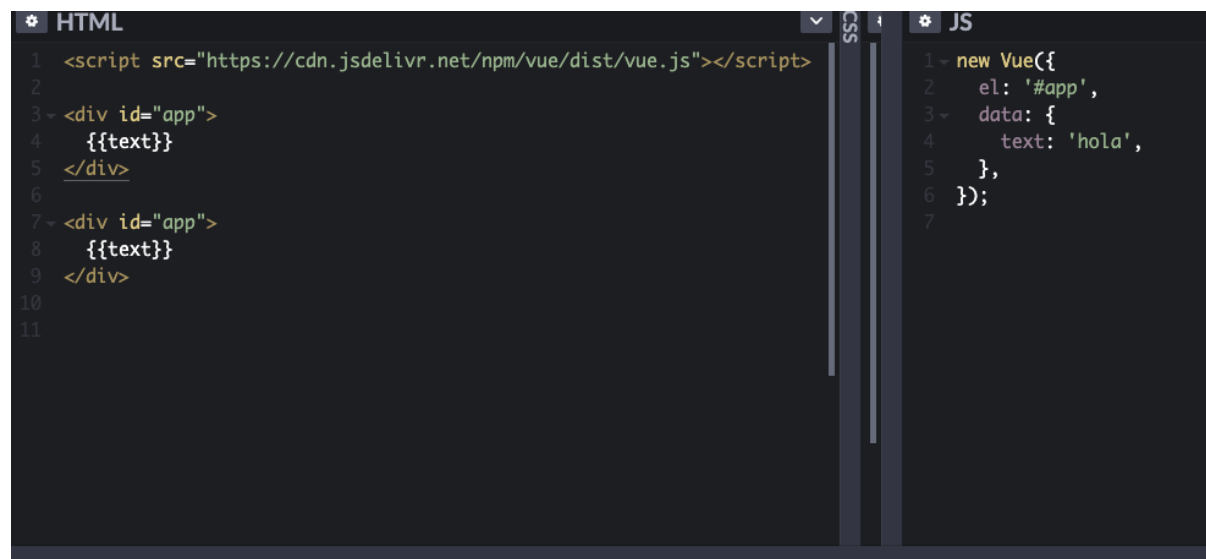




Componentes

Nos permite hacer que todo nuestro código sea bastante reusable y organizado a la vez que más fácil de encontrar. En el siguiente caso vemos que vue sólo renderiza la primera coincidencia y el resto las obvia.



```
HTML
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4   {{text}}
5 </div>
6
7 <div id="app">
8   {{text}}
9 </div>
10
11

JS
1 new Vue({
2   el: '#app',
3   data: {
4     text: 'hola',
5   },
6 });
7
```

ola
{{text}}

En este caso no podríamos reutilizar nada, por esto aparecen los componentes.



```
HTML
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4   <mi-componente></mi-componente>
5   <mi-componente></mi-componente>
6   <mi-componente></mi-componente>
7 </div>
8
9 <div id="app">
10  {{text}}
11 </div>
12
13

JS
1 Vue.component('mi-componente',{
2   data() {
3     return {
4       text: 'Hola soy un componente',
5     };
6   },
7   template: '<h1>{{text}}</h1>'
8 });
9
10 new Vue({
11   el: '#app',
12   data: {
13     text: 'hola',
14   },
15 });
16
```

Crearemos el componente en el js dándole un nombre y definiendo sus propiedades, aunque será obligatorio que creamos un template.

Ahora si queremos complicarlo un poco más podemos crear un botón y cada vez que lo pulse se incremente en uno el contador.

```

HTML
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4   <mi-componente></mi-componente>
5 </div>
6
JS
1 var data = {counter: 0};
2
3 Vue.component('mi-componente',{
4   data() {
5     return data;
6   },
7   template: '<div><h1>He pulsado el botón {{counter}} veces </h1><button @click="counter++">Ahadir click</button></div>',
8 });
9
10 const vm1 = new Vue({
11   el: '#app',
12   data: {
13     text: 'hola',
14   },
15 });
16

```

He pulsado el botón 14 veces

Ahadir click

Hemos definido un objeto con un apropiada `counter` en su interior. Todos trabajan sobre la misma propiedad y deberían ser independientes aunque el código fuera el mismo.

Esto lo podríamos arreglar haciendo que cada componente generara su propio objeto.

```

1 var data = {counter: 0};
2
3 Vue.component('mi-componente',{
4   data() {
5     return {counter: 0,
6           };
7   },
8   template: '<div><h1>He pulsado el botón {{counter}} veces
9

```

Hasta ahora hemos estado viendo componentes globales, que afectan a todo, pero también existen los componentes locales. Para hacerlo local lo definiremos de la siguiente así:

```

HTML
1 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
2
3 <div id="app">
4   <mi-componente></mi-componente>
5   <mi-componente></mi-componente>
6   <mi-componente></mi-componente>
7 </div>
8
JS
1 var miComponente = {
2   data() {
3     return {text: 'Hola soy un componente',
4           };
5   },
6   template: '<h1>{{text}} </h1>',
7 };
8
9 new Vue({
10   el: '#app',
11   data: {
12     text: 'hola',
13   },
14   components: {
15     miComponente
16   }
17 });
18

```

Hola soy un componente

Hola soy un componente

Componentes dinámicos: nos permiten cambiarlos sobre la marcha, partiendo de un tag reservado `component`, a través de información que le hacemos podremos cambiar un componente por otro.

Para verlo creamos tres comonentes locales:

```

var inicio = {
  data() {

```

```

    return {text: 'Hola, estáis en el inicio',
          };
  },
  template: '<h1>{{text}}</h1>',
};

var galeria = {
  data() {
    return {text: 'Bienvenido a la galeria',
          };
  },
  template: '<h1>{{text}} </h1><p>+1</p>',
};

var contacto = {
  data() {
    return {text: 'Contacta con nosotros',
          };
  },
  template: '<h1>{{text}}</h1>',
};

```

En el new Vue:

```

new Vue({
  el: '#app',
  data: {
    currentView: 'inicio',
  },
  components: {
    inicio,
    galeria,
    contacto
  }
});

```

Y añadiremos tres botones para cambiar de un template al otro:

```

<div id="app">
  <component :is="currentView"></component>
  <button @click="currentView = 'inicio'">Cambio a inicio</button>
  <button @click="currentView = 'galeria'">Cambio a inicio</button>
  <button @click="currentView = 'contacto'">Cambio a inicio</button>
</div>

```

Una vez ya sabemos pasar de un componente a otro, vamos a ver como pasamos de un componente padre a un hijo, para ello utilizaremos los **props**, lanzamos nuestro proyecto npm run dev, y modificamos el código para que aparezca así:

Lo primero, app.vue

```

<template>
  <div id="app">
    <h1 class="title">{{title}}</h1>

```

```

    <app-comment-block :title="title"></app-comment-block>
    <app-info></app-info>

  </div>
</template>

<script>
import AppCommentBlock from './ChildOne.vue';
import AppInfo from './ChildTwo.vue';

export default {
  data () {
    return {
      title: 'Título del padre'
    };
  },
  components: {
    AppCommentBlock,
    AppInfo
  }
}
</script>

```

Vamos a añadir un componente que estará en otro archivo ChildOne.vue, que será el componente sombreado en naranja haciéndole un bind de la propiedad título, para enlazarla En el ChilOne.vue:

```

<template>
  <div>
    
    <h3 class="title">titulo: {{title}}</h3>
    <p>
      <strong>John Smith</strong><small>@johnsmith</small><small>31m</small>
    </p>
    Lorem ipsum dolor sit amet, Lorem ipsum dolor sit amet, Lorem ipsum dolor sit
    amet, Lorem ipsum dolor sit amet, Lorem ipsum dolor sit amet,
  </p>
  </div>
</template>

<script>
export default {
  props: ['title'],
}
</script>

```

Las partes resaltadas en naranja deben tener el mismo nombre, que son las que luego enlazaremos con nuestro app.vue.

Y el componente Two:

```

<template>
  <div>
    <h3 class="title">titulo: {{title}}</h3>

```

```

    <p>
      <strong>John Smith</strong><small>@johnsmith</small><small>31m</small>
    <br>
      Lorem ipsum dolor sit amet, Lorem ipsum dolor sit amet, Lorem ipsum dolor sit
    amet, Lorem ipsum dolor sit amet, Lorem ipsum dolor sit amet,
  </p>
</div>
</template>

<script>
export default {
  data () {
    return {
      title: 'Título del componente 2'
    };
  }
}
</script>
<style>
div {
  border: 1px black solid;
}
</style>

```

El proyecto se puede encontrar en [test-vue-cli.zip](#)

Siguiendo con el ejemplo anterior ahora veremos si un prop es requerido o no. Para ello haremos un objeto dentro del **prop** que a su vez lo convertimos en objeto.

De esta forma convertimos al prop en un objeto.

```

props:{
}

```

Y dentro de él la propiedad **title** también tendrá otro objeto que lo que contendrá es un String.

```

export default {
  props: {
    title: {
      type: String,
      required: true,
    },
  },
}

```

Ahora si queremos comprobar la propiedad requerido, dejaremos el app.vue:

```
<app-comment-block></app-comment-block>
```

En la consola nos saldrá un mensaje de error aunque en la página parezca que todo vaya bien:

```
[link] Checking for updates on the server... log.js:14244:23
✖ [Vue warn]: Missing required prop: "title" vue.esm.js?efeb:628
found in
--> <AppCommentBlock> at src/ChildOne.vue
      <App> at src/App.vue
        <Root>
```

Donde debía aparecer el título en el componente uno ya no aparece. Aunque en vez del required también se puede incluir un valor por defecto si el título no apareciese:

```
default: 'Soy el default',
```

Una vez hemos visto como enviar información del padre al hijo, ahora nos interesará como comunicar el componente hijo con el padre. Esto se hace con los **custom events**.

Ahora el ChildOne lo vamos a dejar así, emitiremos un evento que llamará a closeModal:

```
export default {
  props: ['title'],
  methods: {
    closeModal() {
      this.$emit('closeModal',this.id);
    }
  }
}
```

Para incluir botones con el estilo Bulma, iremos a la página oficial:

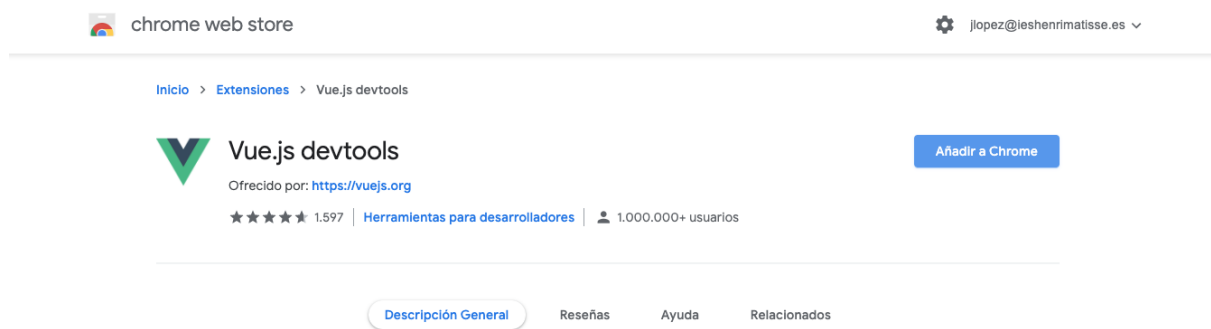
<https://bulma.io/>

Y el html lo modificaremos el head as.i:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>test-vue-cli</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@0.8.0/css/bulma.min.css">
    <script defer src="https://use.fontawesome.com/releases/v5.3.1/js/all.js"></script>
  </head>
```

Para poder ver qué pasa cuando ejecutamos Vue, modificaremos nuestro Consola de Chrome con, vue dev tools, y pulsando command + alt+ I se nos abrirá el command dev tools y podremos inspeccionar la herramienta Vue:

<https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjimejliglipccpnannhbledajbpd?hl=es>



Y en el html añadir la versión de producción tal y como marca la guía:

<https://vuejs.org/v2/guide/installation.html#Direct-It-script-gt-Include>

```
<script src="https://cdn.jsdelivr.net/npm/vue@2.6.11"></script>
```

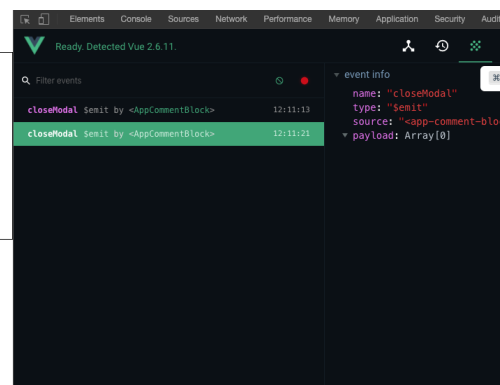
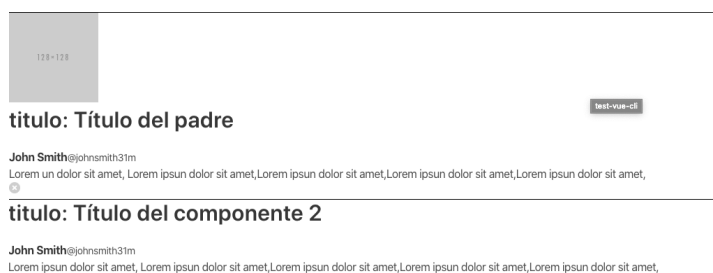
Ahora ya podemos continuar:

Añadimos un botón:

```
<button class="delete is-small" @click="closeModal"></button>
```

Y cada vez que hagamos clic veremos como se ejecuta ese evento:

Título del padre



Ahora nuestra intención es ocultar cuando hagamos click y mostrar el componente:

Nuestro template en el app.vue:


```

<template>
  <div id="app">
    <h1 class="title">{{title}}</h1>
    <app-comment-block
      v-if="show"
      :title="title"
      @closeModal= "show = false">
    </app-comment-block>
    <button v-if="!show"
      class="button is-danger"
      @click="showBlock">Mostrar de Nuevo
    </button>
    <app-info></app-info>
  </div>
</template>

```

Y un poco más abajo:

```

export default {
  data () {
    return {
      show: true,
      title: 'Título del padre',
    };
  },
  methods: {
    showBlock() {
      this.show = true;
    }
  },
  components: {
    AppCommentBlock,
    AppInfo
  }
}
</script>

```

Con esto ya comunicamos eventos del hijo al padre.

Paso de información entre componentes

Para este objetivo vamos a usar **eventBus**. En nuestro main.js vamos a crear una variable que será nuestro lugar de almacenamiento (como si fuera una variable global)

```
index.html  App.vue  main.js
1  import Vue from 'vue'
2  import App from './App.vue'
3
4  export const eventBus = new Vue();
5
6  new Vue({
7    el: '#app',
8    render: h => h(App)
9  })
10
```

Vamos a modificar el ChildTwo en el cual vamos a escribir el nombre de una película y a través de un evento que llamaremos changeMovie, se lo pasaremos al primer componente.

ChildTwo.vue

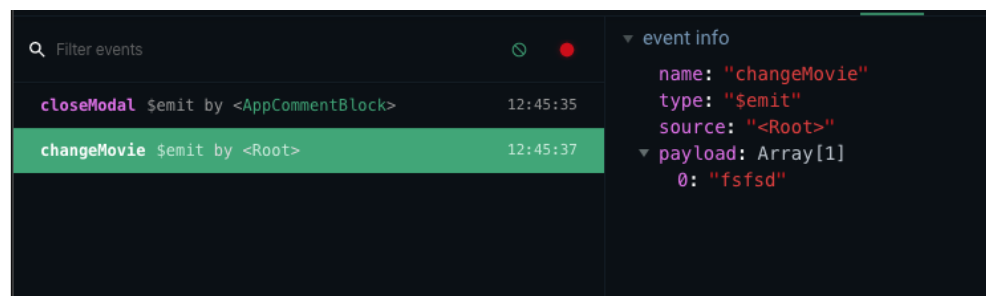
```
<template>
  <div class="">
    <div class="field">
      <input v-model="movieName" class="input is-primary">
    </div>
    <button class="button is-primary" @click="changeMovie"> Cambiar película</button>
  </div>
</template>

<script>

import {eventBus} from './main'

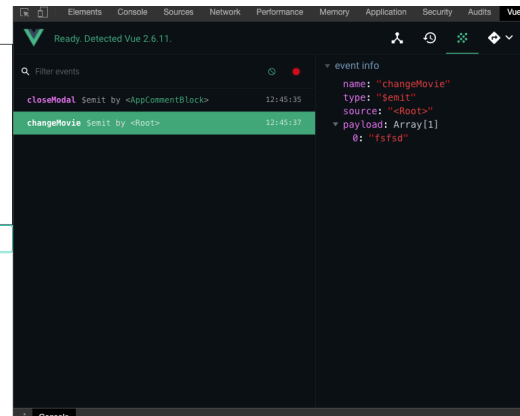
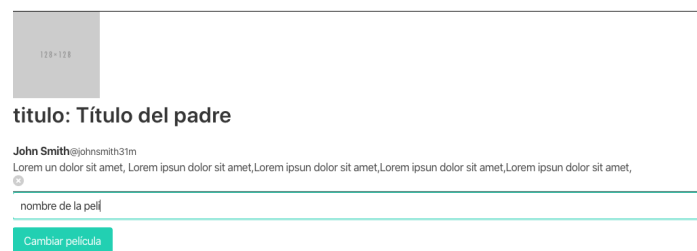
export default {
  data () {
    return {
      movieName: 'fsfsd',
    }
  },
  methods: {
    changeMovie(){
      eventBus.$emit('changeMovie',this.movieName);
    }
  }
}
```

Ahora cuando hagamos click podremos ver como la información se guarda en el objeto:



Y visualmente:

Título del padre



Ahora tenemos que preparar nuestro ChildOne.

```
<template>
  <div class="midiv">
    <h3 class="title">titulo: {{title}}</h3>
    <div class="tile notification is-danger">
      <p class="subtitle">¿Qué película estás viendo?</p>
      <div class="content">
        <p>{{watching}}</p>
      </div>
    </div>
    <button class="delete is-small" @click="closeModal"></button>
  </div>
</template>

<script>
import {eventBus} from './main';

export default {
  props: {
    watching :{
      type:String,
      required:true,
    }
  },
  methods: {
    closeModal() {
      this.$emit('closeModal');
    }
  },
  created() {
```

```
eventBus.$on('changeMovie', (movie) => {  
  this.watching = movie;  
});  
}  
</script>
```

Recogemos en el eventbus la información del otro componente en una variable movie, y se lo pasamos luego al objeto watching.

Slots

Vamos a modificar el app.vue y crear un SongList.vue

app.vue

```
<template>
  <div id="app">
    <song-list></song-list>
  </div>
</template>

<script>
import SongList from './SongList.vue';

export default {
  name:'app',
  components:{
    SongList,
  }
}
</script>

<style>
</style>
```

Songlist.vue

```
<template>
  <div class="box">
    <h1 class="title is-1">Lista de canciones</h1>
    <table>
      <thead>
        <tr>
          <th>Song Name</th>
        </tr>
      </thead>
      <tbody>
        <tr v-for="songNumber in 5">
          <td>Song {{songNumber}}</td>
        </tr>
      </tbody>
    </table>
  </div>
</template>

<script>
export default {
  props: ['playList'],
}
</script>

<style>
```

```
.box {  
  width: 80%;  
  margin: 1em auto;  
}  
</style>
```

Viéndose así:

Lista de canciones

Song Name

Song 1
Song 2
Song 3
Song 4
Song 5

Ahora añadiremos slots, que son pequeños bloques que nos permiten definir un lugar del componente en el cual poder añadir más elementos de forma externa.

En SongList.vue incluimos

```
</table>  
<slot></slot>
```

que es donde añadiremos el elemento nuevo.

Y en app.vue

```
<song-list>  
  <button class="btn">Añadir</button>  
</song-list>
```

Añadiremos un botón al componente en el hueco que hay en el slot.

Ahora vamos a diferenciar diferentes elementos para diferentes slots.

Le damos un nombre al slot:

```
<slot name="header"></slot>
```

Y el elemento que queremos poner en ese slot:

```
<h1 slot="header" class="title is-1">Lista de canciones</h1>
```

E incluso puedo definir un template para varios elementos:

```

<template slot="header">
  <h1 class="title is-1">Lista de canciones</h1>
  <p>Mis canciones</p>
</template>

```

A continuación vamos a ver scoped slot, pasar información hacia fuera.
en el songlist.vue

```

<slot></slot>
<slot :textFromChild="text"></slot>
</div>
</template>

```

```

<script>
export default {
  props: ['playList'],
  data () {
    return {
      text:'Hola desde el hijo',
    }
  }
}

```

Ahora puede ver el texto y mandarlo al padre, el cual tendrá:

```

<template scope="props">
  <p>{{props.textFromChild}}</p>
</template>

```

Tiene que ir en un template obligado. Mostrándose así:

Lista de canciones

Mis canciones

Song Name

Song 1

Song 2

Song 3

Song 4

Song 5

Hola desde el hijo