

Altair Centro Educativo

# Vue.js & Firebase

Proyecto final



Sergio Raúl Bueno Martínez  
[Fecha]

# Índice

<b>1.- ¿Qué es Vue.js y Firebase?</b>	<b>2</b>
<b>2.- Primeros pasos</b>	<b>3</b>
<b>3.- Vue.js</b>	<b>5</b>
<b>3.1.- Mostrar la lista de tareas</b>	<b>5</b>
<b>3.2.- Agregar una nueva tarea</b>	<b>7</b>
<b>3.3.- Eliminar una tarea</b>	<b>9</b>
<b>3.4.- Modificar una tarea</b>	<b>10</b>
<b>3.5.- Marcar como completada o desmarcar una tarea</b>	<b>12</b>
<b>3.6.- Creación de un template</b>	<b>15</b>
<b>4.- FireBase</b>	<b>18</b>

## 1.- ¿Qué es Vue.js y Firebase?

**Vue.js** es un framework progresivo que sirve para construir interfaces de usuario. La biblioteca principal se enfoca solo en la capa de la vista, y es muy simple de utilizar e integrar con otros proyectos o bibliotecas existentes.

Para entenderlo de una forma más clara vemos un ejemplo rápido. Imagínate que tienes una fuente de datos, un modelo, con una información y ese modelo está reflejado en diferentes elementos de tu página web que están mostrando esa información.

Ahora, ¿qué pasa cuando cambia o actualiza la información de ese modelo? Gracias al data binding que posee Vue.js veremos que cambia automáticamente los valores de los elementos de tu página web cambian en cuanto cambia el modelo.

Un framework muy similar a Vue.js es por ejemplo Angular.

**Firebase** es un conjunto de herramientas que te ayudan a desarrollar, complementar, añadir características a tu aplicación tanto para web, Android y en iOS. Una de sus muchas características que vamos a utilizar y puede ser la más conocida de Firebase es la base de datos (NoSQL) a tiempo real (*RealTime DataBase*). Firebase fue una startup que ha sido adquirida por Google actualmente.

Gracias a esta herramienta puedes prescindir de tener un backend propio en tu servidor. Esto tiene su parte buena y mala, la mala es que pierdes un poco de control, pero la buena es que ganas bastante tiempo.

## 2.- Primeros pasos

Ahora vamos a empezar el ejercicio para ver la utilidad que tienen estas dos herramientas unidas.

- Primero tenemos que crear una estructura de carpetas para nuestro proyecto. Dentro de la carpeta donde empezaremos el proyecto crearemos la carpeta css, fonts y js.

- Instalamos bootstrap en nuestro proyecto.

<https://getbootstrap.com/docs/3.3/getting-started/>

Descargamos de aquí la versión SOURCE CODE y cogemos del rar que se descarga: de la carpeta docs/dit los siguientes archivos y los metemos en el proyecto:

- > /css: bootstrap.css

- > /fonts: TODO

- > /js: Bootstrap.js

Asociamos a nuestro index.html dichos archivos para poder utilizarlo.

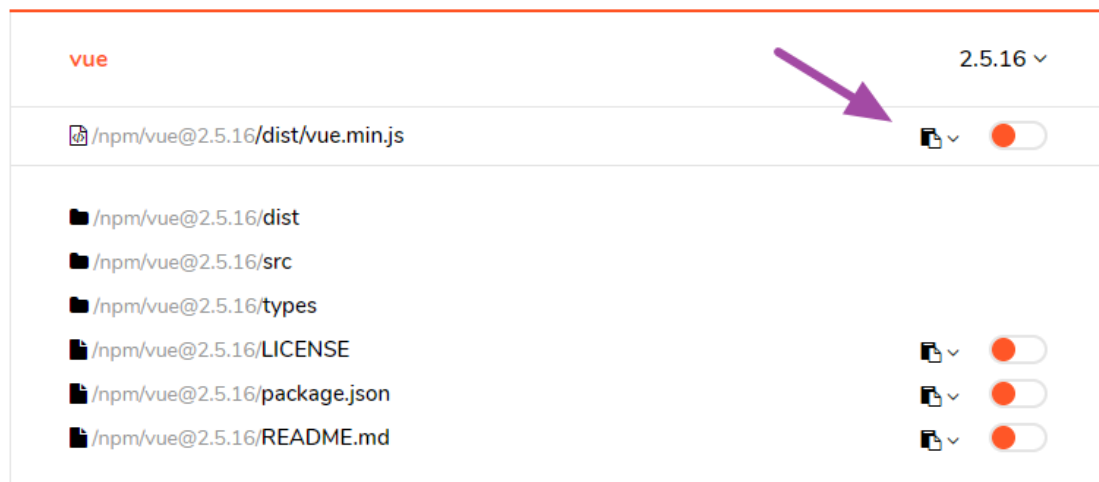
- Creamos ahora la estructura base de nuestro index.html. (Archivo base se encuentra en git)

- Creamos en la carpeta CSS un archivo que se llame main.css donde pondremos algunas directivas de estilo.

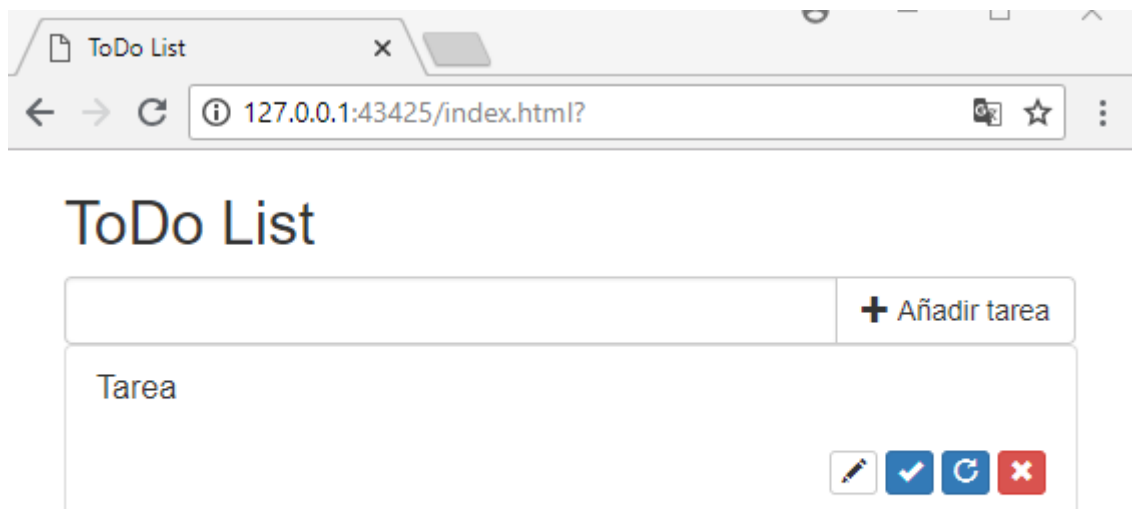
- Tenemos que introducir en <script> de Vue.js para que podamos trabajar con él. Para ello nos vamos a:

<https://www.jsdelivr.com/package/npm/vue> y copiamos la URL de vue.min.js.

## npm CDN Files



- Una vez que tengamos todo instalado y asociado al index.html nos quedará una página tal que así:



## 3.- Vue.js

Vamos a empezar a ver las utilidades que tiene este framework.

Primero crearemos el archivo **main.js** (también hay que asociarlos al HTML con su <script> correspondiente al index.html).

### 3.1.- Mostrar la lista de tareas

Dentro del archivo main.js crearemos la instancia Vue, por ejemplo la lista de tareas que queremos añadir:

```
new Vue({  
  el: '#main',  
  data:{  
    tareas:[  
      {titulo:'Salir a correr', completo: false },  
      {titulo:'Ir al gimnasio', completo: false },  
      {titulo:'Lavar el coche', completo: false },  
      {titulo:'Hacer la compra', completo: false }  
    ]  
  }  
});
```

“el:” -> Se trata del elemento del DOM dónde vamos a renderizar o hacer uso de Vue. En nuestro caso a la etiqueta que contiene el id main





“data:” -> En él estarán todas las propiedades y/o modelos de datos que queramos utilizar en la vista. En nuestro caso es una lista de tareas.

Si queremos ver la información del modelo a través del html pero en forma de json, escribiremos la etiqueta:

```
<pre>{{ $data | json }}</pre>
```

Resultado:

## ToDo List

<input type="text"/>	<a href="#">+ Añadir tarea</a>
Tarea	
<div></div>	

```
{
  "tarefas": [
    {
      "titulo": "Salir a correr",
      "completo": "false"
    },
    {
      "titulo": "Ir al gimnasio",
      "completo": "false"
    },
    {
      "titulo": "Lavar el coche",
      "completo": "false"
    },
    {
      "titulo": "Hacer la compra",
      "completo": "false"
    }
  ]
}
```

Ahora vamos al index.html y recorreremos este modelo de datos con la directiva v-for.

















```
<!-- Listado de tareas -->
<ul class="list-group">
  <li v-for = "(tarea, index) in tareas"
    class="list-group-item clearfix">
```

Su estructura es prácticamente igual que un foreach en c#, es decir, “tarefas” en la lista de datos y “tarea” es el nombre que se le da a cada elemento de la lista.

Para ver los datos de cada elemento “tarea” tenemos que hacer una interpolación con doble llave `{{ tarea.titulo }}`.

Resultado:

# ToDo List

	+ Añadir tarea
Salir a correr	   
Ir al gimnasio	   
Lavar el coche	   
Hacer la compra	   

## 3.2.- Agregar una nueva tarea

Lo primero que vamos a hacer para insertar la funcionalidad de agregar una nueva tarea es irnos al main.js y dentro de data crearnos un nuevo elemento que hará referencia a la nueva tarea y estará inicializado como nulo -> **nuevaTarea: null**,

Ahora debajo de data creamos otro elemento que se llame **methods** y dentro de él irán todos los métodos que queramos meter en nuestro caso ahora añadiremos el método **agregarTarea**:

```
methods:{
  agregarTarea: function(tarea){
    this.tareas.unshift({
      titulo: tarea, completado: false
    });
    this.nuevaTarea = '';
  }
}
```



A continuación vamos a añadir una directiva que enlace con ese método. Se puede poner directamente en el botón de “Añadir tarea”, pero perderíamos la opción de darle a enter y que se guarde, por ello creamos esa directiva en la etiqueta del formulario:

```
<form class="form" @submit.prevent="agregarTarea(nuevaTarea)">
```





Añadiendo .prevent hacemos que no se recargue la página

Resultado:

## ToDo List

+ Añadir tarea

Estudiar para el PREGUNTEITOR

Salir a correr

```
{
  "nuevaTarea": "",
  "tareas": [
    {
      "titulo": "Estudiar para el PREGUNTEITOR",
      "completado": "false"
    },
    {
      "titulo": "Salir a correr",
      "completo": "false"
    },
  ],
}
```

### 3.3.- Eliminar una tarea

Como en el paso anterior, nos vamos al main.js y creamos otro nuevo método: **eliminarTarea**.

```
eliminarTarea: function(indice){  
    this.tareas.splice(indice, 1);  
}
```

Splice se utiliza para navegar dentro de una lista y los parámetros que recibe son (posición donde se coloca, cuantos elementos coge a partir de la posición indicada).

Posteriormente nos vamos al index.html y modificamos el botón de eliminar:

```
<button @click="eliminarTarea(index)"  
    class="btn btn-danger btn-xs">  
    <span class="glyphicon glyphicon-remove"></span>  
</button>
```





@click hace referencia a la acción de hacer click en dicho botón.

Resultado:

## ToDo List

+ Añadir tarea

Ir al gimnasio

```
{  
  "nuevaTarea": null,  
  "tareas": [  
    {  
      "titulo": "Ir al gimnasio",  
      "completo": "false"  
    }  
  ]  
}
```

### 3.4.- Modificar una tarea

Para editar el nombre de la tarea haremos lo siguiente. Primero aparecerá un input de tipo texto donde aparecerá el nombre de la tarea y lo editaremos y lo guardaremos tanto con el botón de guardar como cuando quitemos el foco de ese input (efecto blur).

Para ello lo primero creamos el input en el index.html justo debajo del párrafo del título. Para que solo se muestre el input de la tarea que queremos editar pinchando en el botón, guardaremos el índice del elemento que queremos editar. Entonces cuando el índice corresponda con la tarea en sí es cuando se muestra ese input específicamente.

Para ellos necesitamos una nueva variable en el modelo que se llame **editandoTarea** y que sea iniciado en nulo.

Ahora nos vamos al botón de editar y añadimos un nuevo evento al hacer click el cual guarda en **editandoTarea** el index de la tarea señalada:

```
<button @click="editandoTarea = index"
        class="btn btn-default btn-xs">
  <span class="glyphicon glyphicon-pencil"></span>
</button>
```

```
{
  "nuevaTarea": null,
  "editandoTarea": 3,
  "tareas": [
    {
      "titulo": "Salir a correr",
      "completo": "false"
    },
  ],
}
```

Para que se muestre el input añadimos en la etiqueta:

**v-show="editandoTarea === index"**

Esto hace que solo se muestre el input cuando el index que se obtiene cuando pinches el botón coincide con el de la tarea, es decir, que ahora conseguimos que se vea el input de la tarea que pinchemos el botón de editar.

El siguiente paso es que salga en el input el nombre de la tarea ya precargado, para ello añadimos al input **v-model="tarea.titulo"**:



Ahora vamos a insertar la funcionalidad blur en el input:

**@blur="editandoTarea = null, editarTarea(tarea)"**

Primero ponemos la variable editandoTarea a nulo para desaparezca el input y ejecutamos un método, que crearemos a continuación, que se le pasa la tarea. Para ello vamos al main.js y creamos el método:

```
editarTarea: function(tarea){  
    console.info(tarea);  
},
```

Esto devuelve un objeto a la consola que ahora puede que no le veas utilidad pero se utilizará para conectar con FireBase y darle esa información.






Por último vamos a añadir un botón de guardado, al cual solo se mostrará cuando estemos editando una tarea y la función que va a tener cuando pinchemos en ese botón de guardar va a ser la misma que la del efecto blur:

```
<button v-show="editandoTarea === index"  
    @click="editandoTarea = null, editarTarea(tarea)"  
    class="btn btn-default btn-xs">  
    <span class="glyphicon glyphicon-floppy-saved"></span>  
</button>
```

Resultado:

Salir a correr por el parque del Alamillo

Salir a correr por el parque del Alamillo



### 3.5.- Marcar como completada o desmarcar una tarea

Creamos en el fichero main.css una clase para darle cuando marquemos como completada la tarea:

```
.completado {  
  text-decoration: line-through;  
  font-style: italic;  
  color: gray;  
}
```


Nos vamos a la etiqueta `<p>` donde se muestra el título y le añadimos lo siguiente:

```
<p class="lead" :class="{completado: tarea.completado}">{{tarea.titulo}}</p>
```

**:class** aplica una clase css pero con una condición, en este caso aplica la clase **completado** si **tarea.completado**, que es el elemento del modelo de la tarea que te indica si está o no completado dicha tarea, es true o false.

Para ver que funciona podemos ir al modelo de main.js y modificamos una tarea para ponerlo a true saldrá la tarea tachada.

# ToDo List

<input type="text"/>	<b>+ Añadir tarea</b>
<i>Salir a correrssss</i>	
 	

```
{
  "nuevaTarea": null,
  "editandoTarea": null,
  "tareas": [
    {
      "titulo": "Salir a correrssss",
      "completado": true
    }
  ],
}
```

Ahora vamos a añadir al botón de check (completar una tarea):

**@click="tarea.completado = true"**

Y al botón de descompletar la tarea:

**@click="tarea.completado = false"**






Ahora con esto se completará o no dando a los botones correspondientes.

Para mejorar un poco la vista vamos añadir algunas validaciones para que se muestren los botones en las situaciones que deben y si no se ocultan:

```
<button @click="editandoTarea = index"
        v-show="tarea.completado === false"
        class="btn btn-default btn-xs">
    <span class="glyphicon glyphicon-pencil"></span>
</button>
<button v-show="editandoTarea === index"
        @click="editandoTarea = null, editarTarea(tarea)"
        @click="editandoTarea = index"
        class="btn btn-default btn-xs">
    <span class="glyphicon glyphicon-floppy-saved"></span>
</button>
<button @click="tarea.completado = true"
        v-show="tarea.completado === false"
        class="btn btn-primary btn-xs">
    <span class="glyphicon glyphicon-ok"></span>
</button>
<button @click="tarea.completado = false"
        v-show="tarea.completado === true"
        class="btn btn-primary btn-xs">
    <span class="glyphicon glyphicon-repeat"></span>
</button>
<button @click="eliminarTarea(index)"
        class="btn btn-danger btn-xs">
    <span class="glyphicon glyphicon-remove"></span>
</button>
```

Resultado:

## ToDo List

	+ Añadir tarea
Salir a correrssss	 
Ir al gimnasio	  

### 3.6.- Creación de un template

Por último vamos a crear un template la estructura de lo que llevamos.

Para ello cogemos todo el contenido de dentro del `<div class="container">` y creamos una etiqueta `<template id="todo-template"></template>` y aquí dentro pegamos todo el contenido.

```
<div class="container" id="main">

</div>
<template id="todo-template">
  <div class="col-md-12">
    <h2>ToDo List</h2>
    <!-- Formulario -->
    <form class="form" @submit.prevent="agregarTarea(nuevaTarea)"> ... </form>
    <!-- Listado de tareas -->
    <ul class="list-group"> ... </ul>
  </div>
</template>
```

Ahora nos vamos al main.js y creamos un nuevo componente llamado 'todo-list' al cual se le pasará por parámetro un objeto con la configuración siguiente:

- Template: que indica donde está el template `id="todo-template"`
- Data: contiene una función anónima que retorna el objeto, donde pasamos a esta parte las dos variables: nuevaTarea y editandoTarea.
- Methods: le pasamos todos los métodos de la instancia principal.

En la instancia de Vue solo dejaremos los que son los datos en sí.

Ahora creamos una propiedad llamada por ejemplo tareas, para pasarle datos de:

**props: ['tareas'],**



Resultado:

```
Vue.component('todo-list', {
  template: '#todo-template',

  data: function(){
    return{
      nuevaTarea: null,
      editandoTarea : null,
    }
  },
  props: ['tareas'],
  methods:{
    agregarTarea: function(tarea){
      this.tareas.unshift({
        titulo: tarea, completado: false
      });
      this.nuevaTarea = '';
    },

    editarTarea: function(tarea){
      console.info(tarea);
    },

    eliminarTarea: function(indice){
      this.tareas.splice(indice, 1);
    },
  }
});

new Vue({
  el: '#main',

  data:{
    tareas:[
      {titulo:'Salir a correr', completado: false},
      {titulo:'Ir al gimnasio', completado: false},
      {titulo:'Lavar el coche', completado: false},
      {titulo:'Hacer la compra', completado: false}
    ]
  },
});
```

Nos vamos ahora al index.html creamos dentro del `<div class="container">` una etiqueta llamada como el nombre del componente que acabamos de crear: ("todo-list"). `props: ['tareass']` es una propiedad se transforma en un *custom attribute* pasándole la lista de tareas llamadas `tareass`.

Una cosa a tener en cuenta que si ponemos `tareass="tareass"` estamos indicando que se le pasa un string y no queremos eso, para ello ponemos ":" al principio para que coja el contenido de esa palabra que es una lista.

Resultado:

```
<body>
<div class="container" id="main">

  <todo-list :tareass="tareass"></todo-list>
  <!--<pre>{{ $data | json }}</pre>-->

</div>
<template id="todo-template">
  <div class="col-md-12">
    <h2>ToDo List</h2>
    <!-- Formulario -->
    <form class="form" @submit.prevent="agregarTarea(nuevaTarea)"> ... </form>
    <!-- Listado de tareas -->
    <ul class="list-group"> ... </ul>
  </div>
</template>
<script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.min.js"></script>
<script src="js/main.js"></script>
</body>
```

## 4.- FireBase