

Altair Centro Educativo

# Vue.js & Firebase

Proyecto final



Sergio Raúl Bueno Martínez  
20-6-2018

# 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>
<b>4.1.- Firebase DataBase Realtime</b>	<b>18</b>
<b>4.2.- Firebase Authentication</b>	<b>23</b>
<b>4.3.- Firebase Hosting</b>	<b>27</b>

## 1.- ¿Qué es Vue.js y Firebase?

**Vue.js** se trata de un framework progresivo, esto quiere decir que las funcionalidades principales (el renderizado y el sistema de componentes) están en una pequeña biblioteca, sin embargo es posible añadir todas las funcionalidades que tienen otros frameworks completos como el build systems como webpack o systemjs añadiendo bibliotecas adicionales.

Sirve para construir interfaces de usuario.

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 VueJS veremos que cambia automáticamente los valores de los elementos de tu página web en cuanto cambia el modelo y viceversa.

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.

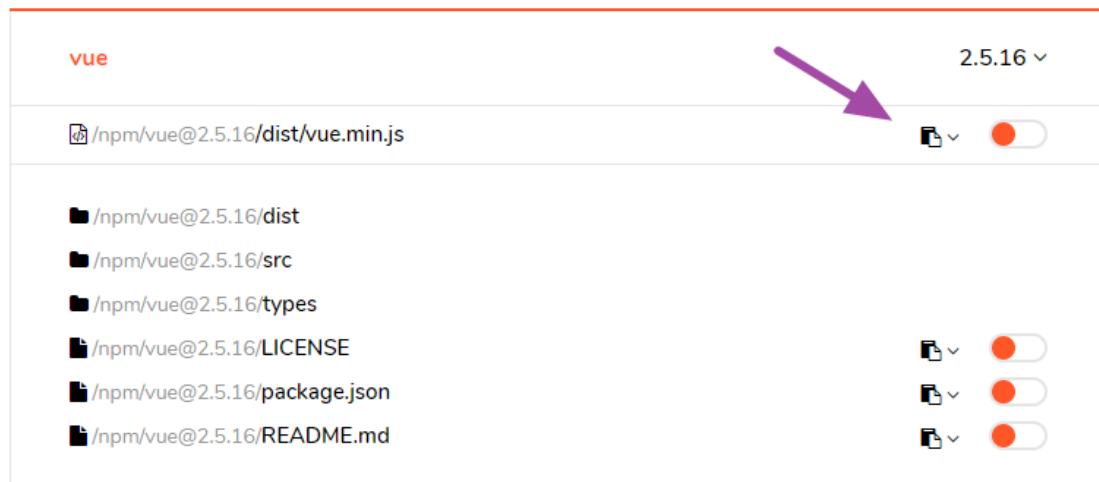
- Creamos ahora la estructura base de nuestro index.html. (Archivo base se encuentra en git)

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

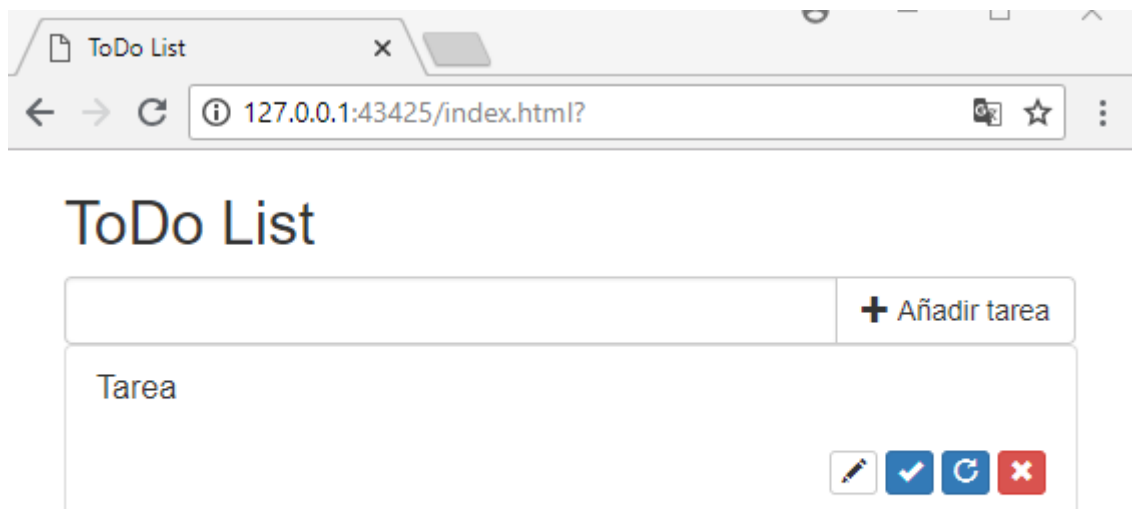
- Tenemos que introducir en <script> de Vue.js para que podamos trabajar con él. Para ello nos vamos a:

<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', completado: false},
      {titulo:'Ir al gimnasio', completado: false},
      {titulo:'Lavar el coche', completado: false},
      {titulo:'Hacer la compra', completado: 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>	

```
{
  "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"
    }
  ]
}
```

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

















```
<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, “tareas” 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

<input type="text"/>	<button>+ Añadir tarea</button>
Estudiar para el PREGUNTEITOR	<div></div>
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 quitamos 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 pinchamos 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>+</b> Añadir tarea
<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
<del>Salir a correr</del>	 
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.

```
<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

Firebase es un conjunto de herramientas que te ayudan a desarrollar, complementar y añadir características a tu aplicación tanto para web, Android e iOS.

Una de sus muchas características que vamos a utilizar, pudiendo ser la más conocida de Firebase, es la base de datos (NoSQL) a tiempo real (RealTime DataBase). También utilizaremos Firebase Authentication y Firebase Hosting.

### 4.1.- Firebase DataBase Realtime

Firebase Realtime Database es una base de datos alojada en la nube. Los datos se almacenan en formato JSON y se sincronizan en tiempo real con cada cliente conectado.

Cuando compilas APPs multiplataforma con los SDK de iOS, Android y JavaScript, todos los clientes comparten una instancia de Realtime Database y reciben actualizaciones automáticamente con los datos más recientes.

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.

Para comenzar nos dirigimos a la página oficial de Firebase: <https://firebase.google.com>. Al ser una aplicación de Google es muy fácil iniciar sesión ya que vale con alguna cuenta de Google.

Una vez iniciada sesión, nos tenemos que ir a la consola y crear un nuevo proyecto.

**Agregar un proyecto**

Nombre del proyecto  
VueFirebase

ID del proyecto  
vuefirebase-4087f

País/Región  
España

☒ Utilizar la configuración predeterminada para el uso compartido de datos de Google Analytics for Firebase

- ✓ Comparte los datos de Analytics con Google para mejorar nuestros productos y servicios.
- ✓ Comparte los datos de Analytics con Google para habilitar la asistencia técnica.
- ✓ Comparte los datos de Analytics con Google para habilitar las comparativas.
- ✓ Comparte los datos de Analytics con los especialistas en cuentas de Google.

☒ Acepto los [Términos relativos a los controladores](#). Esto es obligatorio cuando se comparten datos de Analytics para mejorar los Productos y Servicios de Google. [Más información](#)

☒ Reconozco que uso servicios de Firebase en mi app y acepto las [condiciones](#) aplicables.

CANCELAR **CREAR PROYECTO**

Ahora pinchamos en “Añade Firebase a tu aplicación web” y copiaremos el código:

- La primera línea la copiamos y la añadimos en la sección de scripts del index.html.
- El contenido del otro script son variables de configuración en base al proyecto que se ha creado, es decir conectar nuestra aplicación con Firebase. Se copia y se pone arriba del todo del main.js

Lo primero es añadir la funcionalidad de añadir una tarea pero ahora en lugar de guardarla en la matriz existente vamos a guardarla en la base de datos de Firebase.

Tenemos que eliminar la lista de tareas que tenemos en local, pero hay que crear una misma variable “tareas: []” inicializada como una matriz vacía y será Firebase la que se encargará de ir modificando y rellenando.

Añadimos la referencia del servicio de base de datos en tiempo real de Firebase:

```
var config = {
  apiKey: "AIzaSyB1uG5_GdLNU8_P_UzRus_HUwF_12",
  authDomain: "v-u-51-b1-1227f-firebase.com",
  databaseURL: "https://v-u-51-b1-1227f.firebaseio.com",
  projectId: "v-u-51-b1-1227f",
  storageBucket: "v-u-51-b1-1227f-firebase.com",
  messagingSenderId: "706662115146"
};
firebase.initializeApp(config);

var db = firebase.database();
```

Ahora tenemos que irnos a la instancia principal de Vue lo asignamos a una variable:

```
var vm = new Vue ({
  ...
})
```

Después añadimos un nuevo método de la instancia Vue:

```
mounted: function(){
  // Cargar los datos de BBDD
  db.ref('tareas/').on('value', function(snapshot) {
    vm.tareas = [];
    var objeto = snapshot.val();
    for (var propiedad in objeto) {
      vm.tareas.unshift({
        'key': propiedad,
        completado: objeto[propiedad].completado,
        titulo: objeto[propiedad].titulo
      });
    }
  });
},
```

-> .ref('tareas/') : hace referencia al directorio donde se encuentra todos los elementos que va a obtener dentro de la base de datos.

-> .on('value', function(snapshot){ : value hace referencia a un evento que salta cuando hay algún cambio (en nuestro caso por ejemplo cuando se modifica una tarea, cuando se elimina una tarea, cuando se añade una tarea) junto a una snapshot con el estado específico de ese momento determinado. Otra característica positiva de este evento es que salta cuando se arranca la aplicación.

-> Dentro tenemos que primero vaciar la lista de tareas. Después asignamos a una variable los valores de la snapshot que se le pasa con los datos que hay. Por último recorreremos con un for (igual que un foreach) donde se le pasa los nuevos elementos del modelo que va a tener cada tarea. Aquí es donde metemos dentro del modelo la key del elemento que lo necesitaremos para hacer futuras operaciones con él.

### **AgregarTarea:**

**Push()** : Es un método que se utiliza para escribir datos en Firebase Realtime Database. Es muy parecido a set pero vamos a utilizar push porque este añade una id basada en un timestamp que lo que permiten es que sean única cada nueva entidad que se crea y tendrá una id única porque así no importa la gente que esté trabajando sobre la misma bbdd, siempre se mantendrán una integridad en los datos.

Nos vamos ahora al método “agregarTarea” y borramos el contenido y ponernos el siguiente:

```
agregarTarea: function(tarea){
    db.ref('tareass/').push({
        titulo: tarea, completado: false
    });
    this.nuevaTarea = "";
},
```

Dentro del método se coloca los elementos que se quieren actualizar. Después de ejecutarse el método ponemos el elemento nueva tarea a vacío.

## EditarTarea:

Editamos el evento que ya existía y lo modificamos. Realiza la actualización de un elemento de la base de datos:

```
editarTarea: function(tarea){
    db.ref('tareas/' + tarea['.key']).update({
        titulo: tarea.titulo
    });
},
```

En “.ref(...)” se encuentra la referencia del elemento que se ha modificado, tanto la carpeta donde se encuentra “tareas/” como el identificador del elemento específico “.key”. Después se pone dentro de la función update los elementos que queremos actualizar.

## ActualizarEstadoTarea:

Creemos un nuevo método para actualizar el estado de la tarea al cual se le pasa el estado y la tarea. Para ello cogemos de base el método que hemos modificado antes (editarTarea) y nos basamos de él. La diferencia es que en el interior del método de actualizar tenemos que poner un operador ternario para poner el estado completado en true o false:

```
actualizarEstadoTarea: function(estado, tarea){
    db.ref('tareas/' + tarea['.key']).update({
        completado: estado ? true : false,
    })
},
```

También tenemos que irnos al evento @click del botón “completar tarea” y modificar para que ahora ejecute el método actualizarEstadoTarea pasándole el estado en true y la tarea que hace referencia. Y en el evento @click del botón “descompletar tarea” y hacer lo mismo pero el estado en false y pasándole la tarea.

```
<button @click="actualizarEstadoTarea(true, tarea)"
    v-show="tarea.completado === false" class="btn btn-primary btn-xs">
    <span class="glyphicon glyphicon-ok"></span>
</button>

<button @click="actualizarEstadoTarea(false, tarea)"
    v-show="tarea.completado === true" class="btn btn-primary btn-xs">
    <span class="glyphicon glyphicon-repeat"></span>
</button>
```

## EliminarTarea:

Para realizar la eliminación de un elemento de la lista de tarea tenemos que poner en la referencia el elemento que queremos borrar, como en los métodos anterior: `.ref('tareas/' + tarea['.key']).remove()`:

```
eliminarTarea: function(tarea){  
    db.ref('tareas/' + tarea['.key']).remove();  
},
```

Ahora nos vamos al index, concretamente al evento `@click` del botón de borrado y en vez de pasar el index, ahora le pasamos la tarea que queremos borrar:

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

## 4.2.- Firebase Authentication

Vamos a crear un acceso de la aplicación a través de Google por el cual si no está logueado no podrá ni crear ni editar tareas.

Añadiremos a una barra superior con bootstrap, en la cual tendremos el botón de conectar, y un menú desplegable con el usuario y dentro un botón para cerrar sesión.

Vamos a Firebase, al módulo de "**Auth**" e indicar los métodos de inicio de sesión. En esta versión tenemos que activar aparte del inicio a través de Google, el inicio por usuario y contraseña aunque no lo utilicemos.

Ahora vamos al código (main.js) y añadimos dos nuevos elementos al modelo de datos principal:

```
autenticado: false,  
usuarioActivo: null,
```



Posteriormente tenemos que añadir en el evento “**mounted**” la función de **onAuthStateChanged**. Es un eventListener que estará escuchando cambios en auth. Cuando el usuario esté logueado pondrá el elemento “**autenticado**” a true y en el elemento “**usuarioActivo**” guardará toda la información del usuario logueado y cuando el usuario no lo esté, pondrá “**autenticado**” a false y “**usuarioActivo**” a null:

```
firebase.auth().onAuthStateChanged(function(user) {  
  if (user) {  
    console.log('Conectado', user);  
    vm.autenticado = true;  
    vm.usuarioActivo = user;  
  } else {  
    console.warn('No conectado');  
    vm.autenticado = false;  
    vm.usuarioActivo = null;  
  }  
});
```

Lo siguiente que tenemos que hacer es crear en la instancia Vue principal crear dos nuevos métodos: conectar y desconectar:

-> conectar: creamos arriba del documento, donde está la variable db, una nueva variable que sea: **var provider = new firebase.auth.GoogleAuthProvider();** . Ahora utilizamos esta variable, la cual crea una instancia del objeto del proveedor Google. Se invoca **signInWithPopup** que hace que salga una ventana emergente para pedirles a los usuarios que accedan con sus cuentas de Google. Únicamente dejamos el catch ya que sólo se capturará información cuando se produzca un error.

-> desconectar: Se invoca **signOut()** que se utiliza para salir de la sesión de un usuario. Únicamente dejamos el catch ya que sólo se capturará información cuando se produzca un error.

```

methods: {
  conectar: function(){
    provider.addScope('https://www.googleapis.com/auth/plus.login');
    firebase.auth().signInWithPopup(provider).catch(function(error){
      console.log('Error haciendo login: ', error);
    });
  },
  desconectar: function(){
    firebase.auth().signOut().catch(function(error){
      console.log('Error haciendo logout: ', error);
    });
  }
},
},

```

Para poder hacer el inicio de sesión y el cierre del mismo, tenemos que asociar esos métodos a algunos botones de nuestro documento:

```

<a @click="conectar()" href="#">Conectar</a>
<a @click="desconectar()" href="#" class="btn btn-danger btn-block">
Cerrar Sesión </a>

```

Para mostrar el nombre de usuario que se conecta cambiamos el literal de “Nombre de usuario” por **{{usuarioActivo.displayName}}**

Ahora tenemos que ocultar el desplegable con el nombre de usuario cuando no tengamos iniciada sesión. Para ello tenemos que ponerle la siguiente directiva al dropdown correspondiente:

```

<li v-if="autenticado" class="dropdown">

```

Y para el caso contrario, de ocultar el botón de conectarse, cuando estemos logueados, tenemos que añadir la siguiente directiva al elemento:

```

<li v-if="!autenticado" class="active">

```

Vamos a ponerle la propiedad de **required** al input donde escribimos la nueva tarea para que no se pueda enviar una tarea vacía. Además ahora vamos a aplicar de forma dinámica el estado disabled al botón de “Añadir tarea” para que cuando no estés logueado no puedas añadir ninguna.

Para ello tenemos que crear dos propiedades más en el elemento “**props**” del main.js que sean **autenticado** y **usuarioActivo**, exactamente igual que las propiedades del modelo:

```
props: ['tareas', 'autenticado', 'usuarioActivo']
```

Ahora nos vamos al componente del index.html y añadimos las dos nuevas propiedades, es decir pasarle a este componente, desde la instancia padre, estas dos nuevas propiedades:

```
<todo-list :tareas="tareas" :autenticado="autenticado" :usuario-  
activo="usuarioActivo"> </todo-list>
```

Con esto podemos ahora poner de forma dinámica la propiedad **disabled** al botón de añadir tarea:

```
<button class="btn btn-default" type="submit" :disabled="!autenticado">
```

Esta propiedad se les pone a los botones de edición para que si no has iniciado sesión no puedas hacer usos de ellos.

Lo último que vamos a hacer es poner a lado del nombre de la tarea, el nombre del usuario que ha realizado la creación de la misma. Nos vamos al main.js y en el evento de **agregarTarea** añadimos el nombre del usuario y el uid que es la id del usuario:

```
nombre: vm.usuarioActivo.displayName,  
uid: vm.usuarioActivo.uid,
```

Ahora tenemos procesarlas, para ello en el evento de mounted tenemos que crear dos nuevos elementos que tendrá una tarea:

```
vm.tareas.unshift({  
  '.key': propiedad,  
  completado: objeto[propiedad].completado,  
  titulo: objeto[propiedad].titulo,  
  nombre: objeto[propiedad].nombre,  
  uid: objeto[propiedad].uid,  
});
```

Para concluir tenemos que añadir esta información en index para poder verlo:

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

### 4.3.- Firebase Hosting

Ahora vamos a ver Firebase Hosting. Se trata de un servicio de Firebase para poder hospedar aplicaciones o recursos estáticos en sus propios servidores. Ofrecen un certificado SSL por defecto y un CDN para servir esos recursos estáticos de diferentes zonas del planeta.

Primero tenemos que instalar Firebase CLI. La CLI (interfaz de línea de comandos) de Firebase que necesita Node.js y npm, que se pueden instalar según las instrucciones que aparecen en <https://nodejs.org/>. Si instalas Node.js, también se instala npm. (Para usar Firebase CLI, necesitas Node.js versión 5.10.0 o una más reciente.).

Una vez que instales Node.js y npm, puedes instalar Firebase CLI a través de npm:

```
npm install -g firebase-tools
```

Esto instala el comando firebase disponible de manera global.

-> Aquí tenemos que indicar que proyecto de firebase va a coger para implementar.

Tenemos que crear una nueva carpeta en el directorio donde se encuentra el proyecto y meter ahí todo el contenido de ficheros y carpetas que tenga tu proyecto.

Para empezar tenemos que indicar con que cuenta nos logueamos para hacer las operaciones posteriores:

## firebase login

```
C:\VueJSHosting>firebase login
? Allow Firebase to collect anonymous CLI usage and error reporting information? Yes

Visit this URL on any device to log in:
https://accounts.google.com/o/oauth2/auth?client_id=563584335869-fgrhgmd47bqnekij5i8b5pr03ho849e6.apps.googleusercontent.com&scope=email%20openid%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloudplatformprojects.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Ffirebase%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloud-platform&response_type=code&state=671995202&redirect_uri=http%3A%2F%2Flocalhost%3A9005

Waiting for authentication...
+ Success! Logged in as sergioraul92@gmail.com
```

Ahora nos vamos al cmd otra vez y nos dirigimos a la ruta del proyecto y ejecutamos el comando:

## firebase init

```
C:\VueJSHosting>firebase init

##### 
##      ##      ##      ##      ##      ##      ##      ##      ##      ##      ##
##### 
##      ##      ##      ##      ##      ##      ##      ##      ##      ##      ##
##      ##      ##      ##      ##      ##      ##      ##      ##      ##      ##
##      ##      ##      ##      ##      ##      ##      ##      ##      ##      ##
##### 

You're about to initialize a Firebase project in this directory:

  C:\VueJSHosting

Before we get started, keep in mind:

  * You are currently outside your home directory
  * You are initializing in an existing Firebase project directory

? Are you ready to proceed? Yes
? Which Firebase CLI features do you want to setup for this folder? Press Space to select features, then Enter to confirm your choices. Hosting: Configure and deploy Firebase Hosting sites 1
=== Project Setup

First, let's associate this project directory with a Firebase project.
You can create multiple project aliases by running firebase use --add,
but for now we'll just set up a default project.

i .firebaseerc already has a default project, skipping

=== Hosting Setup

Your public directory is the folder (relative to your project directory) that
will contain Hosting assets to be uploaded with firebase deploy. If you
have a build process for your assets, use your build's output directory.

? What do you want to use as your public directory? VueJsHost 2
? Configure as a single-page app (rewrite all urls to /index.html)? No 3
+ Wrote VueJsHost/404.html
? File VueJsHost/index.html already exists. Overwrite? No
i Skipping write of VueJsHost/index.html

i Writing configuration info to firebase.json...
i Writing project information to .firebaseerc...
+ Firebase initialization complete!
```

Si ejecutas el comando `firebase init`, se crea un archivo de configuración `firebase.json` en la raíz del directorio del proyecto.

En el proceso de ejecución del comando tenemos que elegir:

- 1- La funcionalidad queremos implementar: Hosting
- 2- El directorio donde se encuentra tu proyecto: (Nombre de la carpeta que creamos antes y que contiene todo el proyecto): `VueJsHost`
- 3- Ahora nos pedirán que si quieres sustituir los `index.html` que ya existen por unos que te pone por defecto y tenemos que decir a todo que no: `N`

Una vez realizado la inicialización de la aplicación tenemos que implementarlo:

## firebase deploy

```
C:\VueJSHosting>firebase deploy
=== Deploying to 'vuefirebase-4087f'...
i  deploying hosting
i  hosting: preparing VueJsHost directory for upload...
+  hosting: 13 files uploaded successfully
+  Deploy complete!

Project Console: https://console.firebase.google.com/project/vuefirebase-4087f/overview
Hosting URL: https://vuefirebase-4087f.firebaseio.com
```

Cuando termine nos pondrá cual es la URL de tu aplicación. También podemos ver la URL en el módulo de Hosting de Firebase.

Dominios	
Dominio	Estado
vuefirebase-4087f.firebaseio.com	Predeterminado