

Procesos Asíncronos

Un proceso asíncrono se realiza cuando la respuesta no es bloqueante para el flujo de la aplicación, lo que implica que cuando un proceso asíncrono es ejecutado, uno tranquilamente puede seguir navegando en la aplicación web, entrar a otra página de la aplicación web o incluso retirarse de la misma página web sin ningún problema.

Los procesos asíncronos también conocidos como ajax son ampliamente utilizados cuando se requiere información del lado del servidor y esta se requiere mostrar en la aplicación web sin refrescar la actual vista, así que de esa manera no se vea afectado el flujo actual de la aplicación web. Un ejemplo claro de estos procesos asíncronos es cuando en el buscador de Facebook comenzamos a introducir nombres en el buscado y el buscador carga una lista de coincidencias, esta carga es asíncrona ya que la aplicación web no se queda esperando el retorno de las coincidencias sino que solamente mantiene un símbolo de carga pero uno puede navegar a través de la web sin ningún problema e incluso cambiar de página.



Diagrama 01: Sincrónico vs Asíncrono (ajax)

Un claro ejemplo de los mismos es cuando realizamos búsquedas en google y google nos realiza sugerencias mientras escribimos cierta palabra

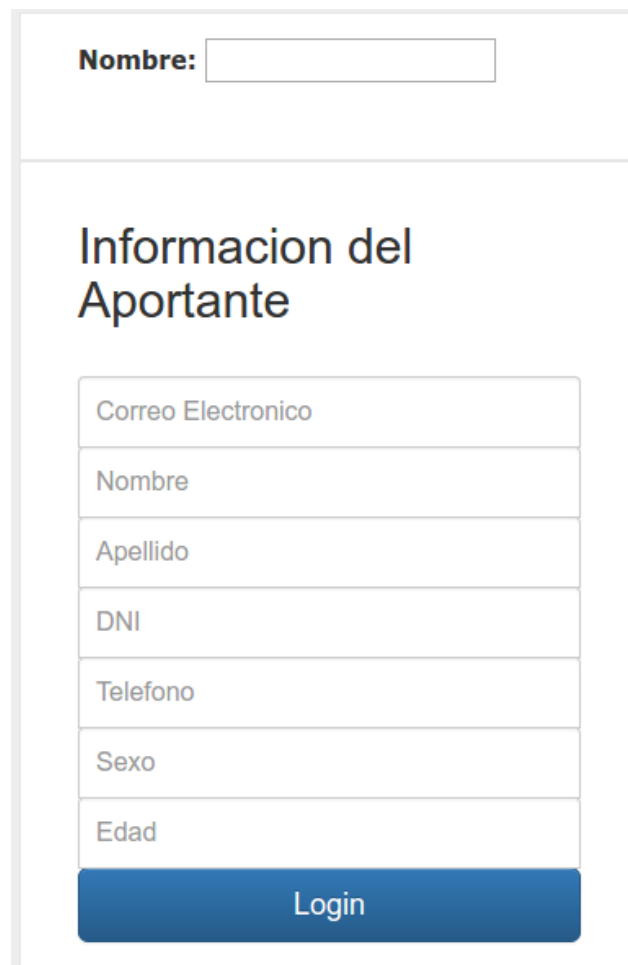
soporte
soporte pedagogico
soporte tecnico
soporte universal
soporte

Ejemplo de Prueba:

Realizaremos un código de prueba, para ello requeriremos el Código de ejemplo 02

Tienes una empresa con una cantidad diversa de empleados, lo que tu quieres realizar es mostrar un reporte de todos los empleados con su información personal.

En este caso se tendrá la siguiente vista:



Nombre:

Informacion del Aportante

Correo Electronico
Nombre
Apellido
DNI
Telefono
Sexo
Edad

Login

Imagen 01: Formulario de empresa

CONFIGURACION

Las configuraciones de las reglas de acceso de la base de datos deberán de ser configuradas de la siguiente manera:

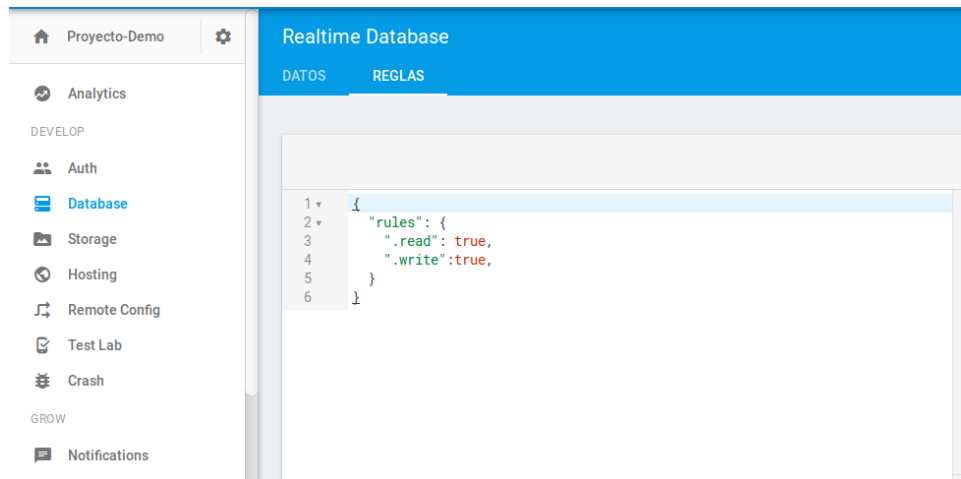


Imagen 02: Permisos de Acceso a Database

En la sección de la base de datos se debe de comenzar con la siguiente estructura, haciéndole un click se agrega.

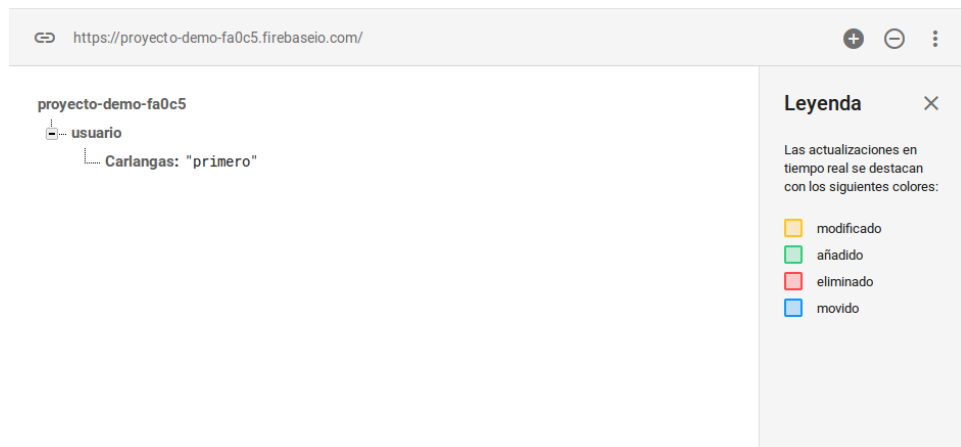


Imagen 03: Esquema inicial de la base de datos

Se procede a realizar un click al usuario y posteriormente se realiza un click en el botón derecho superior y luego a import json e importar el archivo database.json que se encuentra dentro del código de ejemplo

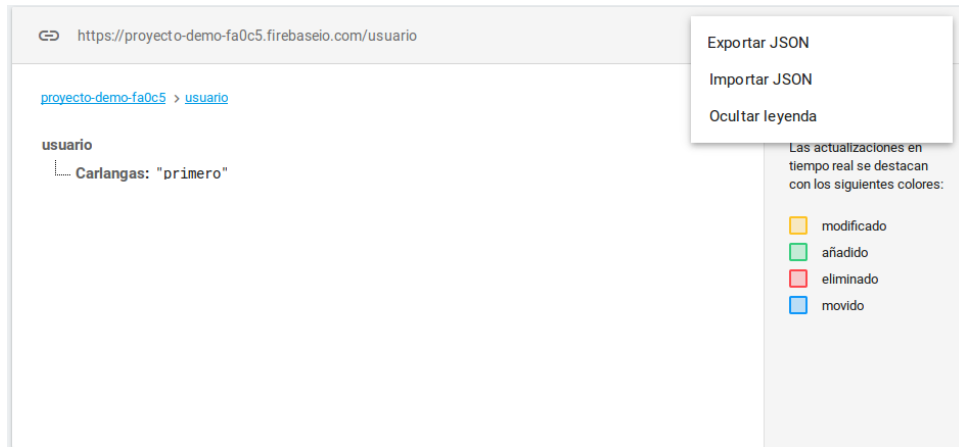


Imagen 04: Importar JSON

Al final de todo la estructura de la base de datos del proyecto debería de quedar de la siguiente manera

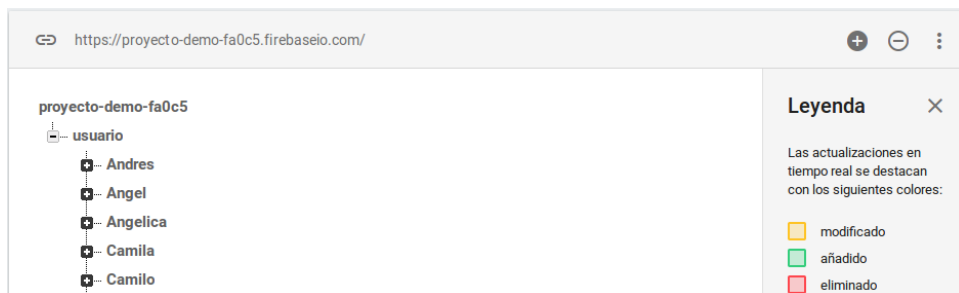


Imagen 05: Esquema final de base de datos

EJECUCION

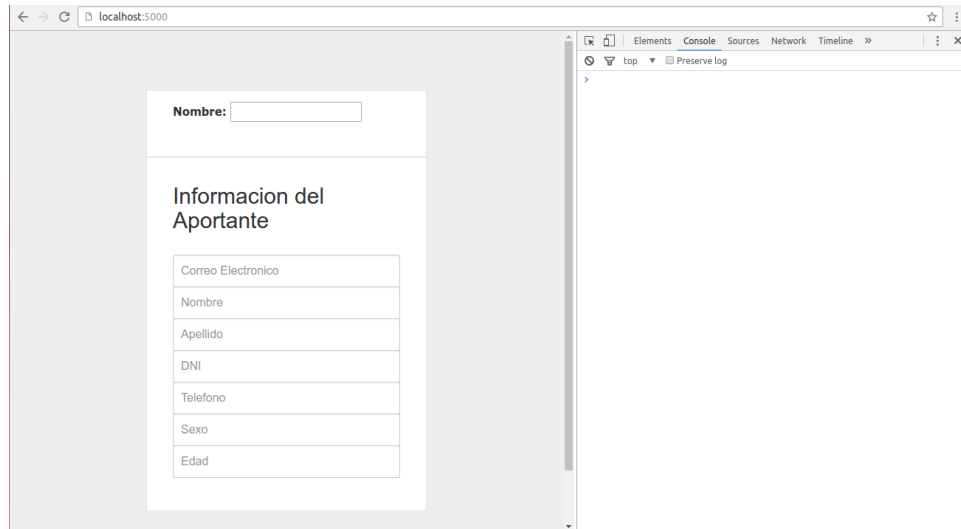
Al descomprimir el archivo posicionarse en la carpeta y ejecutar por el Git Bash el comando siguiente:

```
→ asincrono_hackspace firebase serve
Starting Firebase development server...

Project Directory: /home/carlos/asincrono_hackspace
Public Directory: ./

Server listening at: http://localhost:5000
```

Luego escribir en una ventana del navegador la siguiente ruta puesta en la consola se podrá visualizar lo siguiente



El pequeño proyecto de manera sencilla busca al escribir un nombre de un empleado en un campo mostrar una lista de coincidencias con el texto ingresado, mostrar sugerencias y una vez escogida una sugerencia mostrar la información del empleado

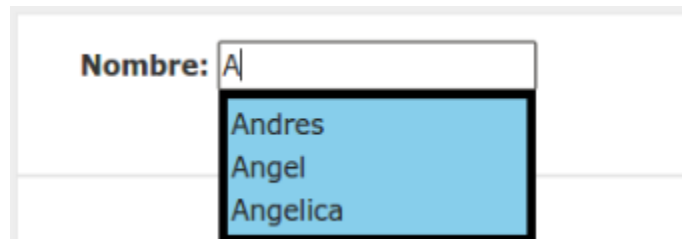


Imagen 06: Autocomplete de ejemplo

ca-tech@dps.centrin.net.id
Andres
Brendan Ballance
7575577
9110520355
Masculino
36

Imagen 07: Rellenado del formulario de ejemplo

EJECUCIÓN

A nivel de código, nos centraremos en el archivo main.js

Para completar nuestros requerimientos tendremos que realizar las siguientes acciones:

- Listar los usuarios
- Mostrar información del usuario

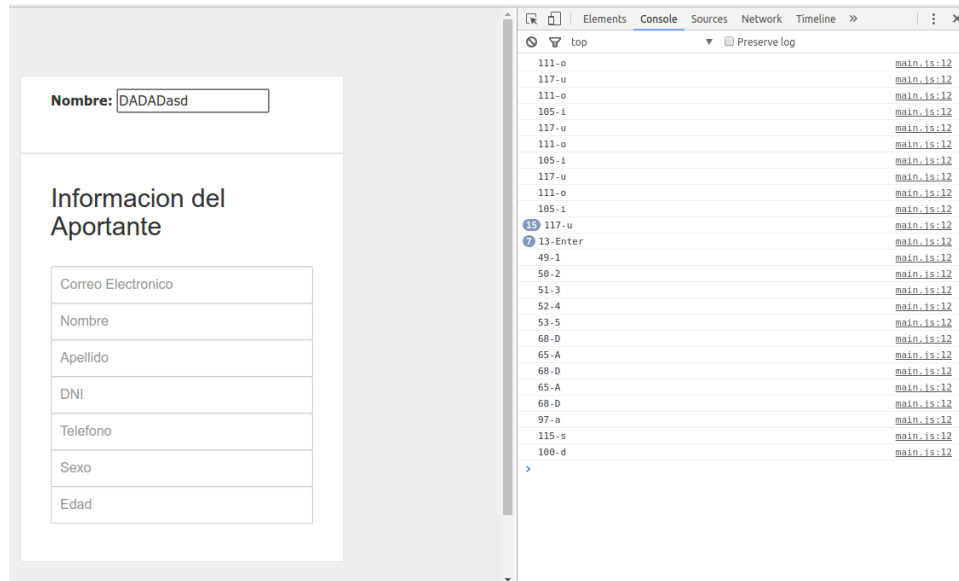
Comenzaremos agregando el archivo de configuración del firebase, propio de cada proyecto

```
var config = {  
  apiKey: "AIzaSyDtxS_3-dszy3_jjz1JCr8v9xtbwBIpdSs",  
  authDomain: "proyecto-demo-fa0c5.firebaseio.com",  
  databaseURL: "https://proyecto-demo-fa0c5.firebaseio.com",  
  storageBucket: "proyecto-demo-fa0c5.appspot.com",  
};  
firebase.initializeApp(config);
```

Comenzaremos con la función que agregará un evento a un elemento HTML, y solamente mostraremos el key y el texto de la tecla presionada

```
$(document).ready(function(){  
  $('#tags').on('keypress',function(e){  
    console.log(e.which+"-"+e.key);  
  });  
});
```

A continuación comenzaremos actualizando el navegador (presionando F5), y veremos la siguiente pantalla y si comenzamos a digitar podremos ver al lado derecho el carácter y el código propio



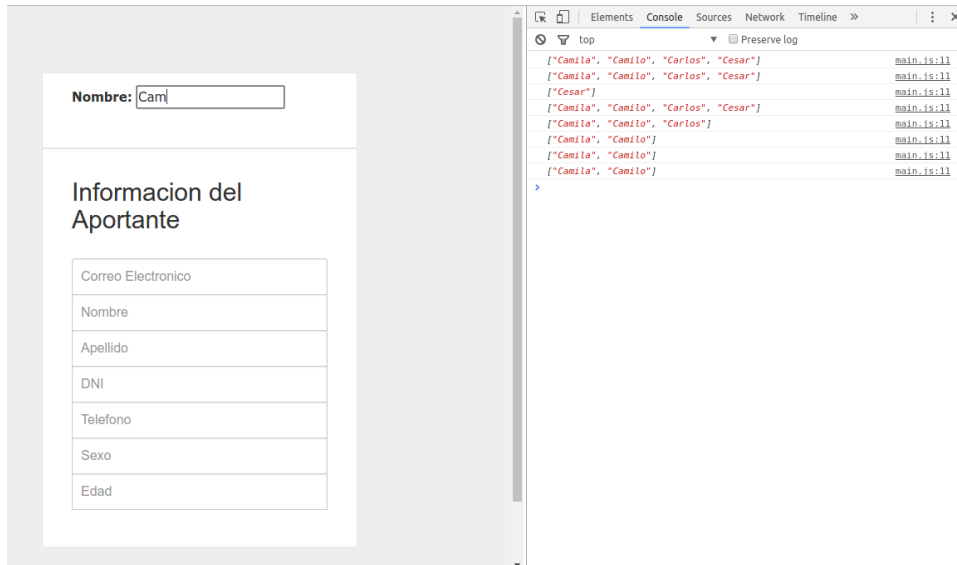
Comenzaremos a refactorizar el código que tenemos para proveer la funcionalidad deseada, primero restringiremos si en el texto de autocomplete, se comienza a teclear cualquier tecla distinta al ENTER se comenzara a cargar las opciones del autocomplete sino se cargara toda la data del usuario en el formulario, se cambió del evento keypress a keyup puesto que el primero retornaba data cuando era presionada una tecla pero no retornaba lo que realmente existía en el elemento del HTML

```
function getAutoCompleteElements(substring){
    console.log(substring);
}
function loadForm(data){
    console.log(data);
}
$(document).ready(function(){
    $('#tags').on('keyup',function(e){
        if(e.which==13){
            loadForm($(this).val());
        }else{
            getAutoCompleteElements($(this).val());
        }
    });
});
```

Continuaremos con el flujo de mostrar el autocomplete por ello realizaremos la funcionalidad de la función `getAutoCompleteElements`, en la función pasa lo siguiente: se hace referencia a los usuarios en general y se piden todos los valores, lo que retorna está en la variable `snapshot` y de la cual solo nos importan sus valores por ello le realizamos un `val()` y obtendremos todos los nombres de los usuarios, posteriormente los recorremos en un bucle a través de la variable `key`. Posteriormente realizamos una validación, si es que el `substring` enviado es parte del nombre entonces la función `indexOf` retornara un número entre 0 y la dimensión del nombre por ello es la validación de que sea mayor a `-1` y la siguiente validación de menor a 5 es para que no se muestren más de 5 elementos en el autocomplete.

```
function loadAutoComplete(data){  
  ...console.log(data);  
}  
function getAutoCompleteElements(substring){  
  ...var val = substring;  
  ...var i=0;  
  ...var names = [];  
  ...firebase.database().ref('usuario/').on('value', function(snapshot){  
  ...for(key in snapshot.val()){  
  ...  if(key.indexOf(val)> -1){  
  ...    if(i<5){  
  ...      names.push(key);  
  ...      i++;  
  ...    }  
  ...  }  
  ...}  
  ...loadAutoComplete(names);  
  ...});  
}
```

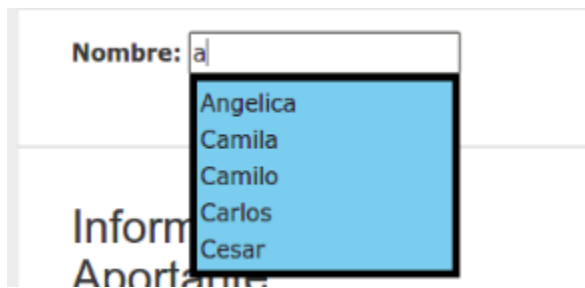
Volviendo al navegador, realizando una recarga del mismo para actualizar el main.js al que hace su referencia y escribiendo data dentro del campo del autocomplete se mostrar lo siguiente en nuestro navegador, lo que demostrara que está filtrando por coincidencia



Posteriormente se realizara el seteo de esas coincidencias en el texto que se desplegara al tipear algún texto

```
function loadAutoComplete(data){  
  ...$("#tags").autocomplete(  
    ...{  
      ...source: data,  
    ...});  
}
```

Posteriormente recargando en el navegador y tipeando alguna vocal dentro del campo se debería de presentar el siguiente efecto



A continuación proseguiremos con el flujo de cargar toda la data del usuario, modificando el código para que luzca de la siguiente forma, como ya escribimos el nombre la base de datos de firebase busca la ruta de usuario el nombre que pasas.

```
function setDataIntoHTML(data){
  ... console.log(data);
}
function loadForm(data){
  ... firebase.database().ref('usuario/'+data).on('value', function(snapshot){
  ... var data = snapshot.val();
  ... setDataIntoHTML(data);
  ... });
}
```

Ahora nos dirigimos al navegador presionamos F5 para recargar los recursos y debería de suceder lo siguiente, mostrar toda la información del usuario en forma de array

The screenshot shows a web application with a form on the left and a browser console on the right. The form has a label "Nombre:" followed by a text input containing "Jonan". Below this is a section titled "Informacion del Aportante" containing a table with the following fields: Correo Electronico, Nombre, Apellido, DNI, Telefono, Sexo, and Edad. The browser console on the right shows a log of an array of user objects. The first object is {apellido: "Sherise Shortt", dni: "2943522", edad: "64", email: "stone@meekness.com", nombre: "Angel"}. The second object is {apellido: "Martha Perry", dni: "4689018", edad: "34", email: "rotary.bali.kutagmail.com", nombre: "Angelica"}. The third object is null. The fourth object is {apellido: "Diana Brooks", dni: "5611074", edad: "41", email: "pitamahagindosat.net.id", nombre: "Jonan"}. The console log also shows the main.js:33 line number for each object.

Finalmente modificaremos la función final para que se pinte adecuadamente en el formulario la data del usuario a partir de su nombre, se le agregara el texto correspondiente a cada elemento HTML.

```
function setDataIntoHTML(data){
  ... $("#email").val(data.email);
  ... $("#last").val(data.apellido);
  ... $("#dni").val(data.dni);
  ... $("#age").val(data.edad);
  ... $("#name").val(data.nombre);
  ... $("#sex").val(data.sexo);
  ... $("#telephone").val(data.telefono);
}
```

Finalmente volveremos al navegador y recargaremos la página y escribiremos un nombre existente y deberá de cargar lo siguiente:

Nombre:

Informacion del Aportante

pitamaha@indosat.net.id
Jonan
Diana Brooks
5611074
8007079723
Masculino
41

CONCLUSIONES

Al terminar este tutorial uno podrá recuperar la data de la base de datos de firebase