PERÍODO 2019

PROYECTO DE INGENIERÍA

Aplicación de control de mesas en restaurante

ELABORADO POR:

MIGUEL BAILÓN

LUIS MOYÓN

GUIDO RAMIREZ

ROBERTO VALAREZO

MATERIA: PROGRAMACIÓN DE SISTEMAS TELEMÁTICOS

CARRERA DE INGENIERÍA EN MECATRÓNICA

FIEC - ESPOL

Resumen ejecutivo

El presente proyecto de ingeniería plantea mejora la eficacia y eficiencia de un restaurante, permitiendo el control en tiempo real de las sillas ocupadas mediante una aplicación. Esta permitirá la correcta distribución de las mesas y permitirá usar todas las sillas disponibles a la vez.

Introducción

Para todo tipo de negocios, los conceptos de eficacia y eficiencia son de vital importancia para la productividad del mismo. Eficacia, se refiere a “hacer las cosas correctas”, es decir hacer lo que se tenga que hacer para lograr los objetivos planteados. Eficiencia, en cambio es “hacer correctamente las cosas”, en otras palabras, usar correctamente los recursos, para lograr los objetivos lo más rápida posible. Estos conceptos deben ser unidos por un buen emprendedor, ya que ambos conceptos no siempre se logran al mismo tiempo. Por lo que el enfoque del proyecto es que un restaurante logre ambas cualidades.

En este caso es de suma importancia la atención al cliente que brinda. Y en muchas ocasiones, las esperas muy largas, dejan un mal sentimiento de satisfacción en el cliente. Lo cual puede influir en su decisión de volver en otra ocasión. Adicionalmente, hay ocasiones en las que le restaurante tiene sillas disponibles, pero están en mesas ocupadas, por lo que no se las toma en consideración, esto baja su eficiencia. Por lo tanto, la correcta distribución de los clientes en las mesas y el uso de todas las sillas disponibles al mismo tiempo son de vital importancia para mejorar la eficiencia y la eficacia. Al lograr ubicar los clientes en sus mesas de manera eficaz y utilizar todos los recursos posibles a la vez sin desperdiciar permitirá aumentar la clientela del lugar.

Descripción del problema

Un restaurante requiere mantener un control en tiempo real de los asientos que se encuentran desocupados, para poder determinar de forma rápida las mesas disponibles. Se propone usar un sensor en los asientos y mediante una aplicación móvil mapear los lugares disponibles.

Objetivos específicos

- Mejorar la eficiencia de la asignación de asientos a los clientes de un restaurante.
- Optimizar el uso de la cantidad total sillas en un mismo lapso de tiempo.

Funcionamiento de la solución

La solución propuesta requiere que a cada silla se le debe incorporar un Arduino acoplado con un sensor de presión. Este sensor manda una señal de que la silla está siendo ocupada cuando algún cliente está sentado en la silla. Gracias al uso de un módulo de internet, el Arduino manda esta información a una base de datos remota en la nube en tiempo real.

Cada silla actúa como un objeto individual; sin embargo, cada mesa, dependiendo de su tamaño, tiene asignado como mínimo una silla. Por lo tanto, con tal de que una silla asignada a una mesa este ocupada, la mesa será tomada como ocupada con asientos libres.

Para el mapeo del restaurante se usará un aplicativo móvil, el cual está conectado a la base de datos y mediante el uso de tres colores (verde, amarillo y rojo) mostrará, en tiempo real, si la mesa está disponible u ocupada (con sillas libres o completamente ocupada) respectivamente.

La aplicación permitirá también mover las sillas de una mesa cualquiera y asignarlas a otra. Para así lograr, que se usen todas las sillas posibles a la vez. También muestra el número de sillas disponibles y ocupadas en tiempo real.

Solución propuesta

Se propone implementar un dispositivo IoT de bajo costo capaz de detectar la variación de presión sobre una superficie, en este caso, una silla. Con el uso una base de datos remota para el almacenamiento digital de las mesas y sillas registradas en el restaurante, se planea monitorear el estado de estas. Cabe destacar que dicho servicio remoto de base de datos debe ser eficiente al momento de mostrarlos, es decir, ser capaz de transmitir en tiempo real.

Por último, se plantea diseñar un aplicativo móvil agradable con el usuario para mostrar en vivo los estados cambiantes de las mesas y sillas de un restaurante previamente registrado con nosotros.

Recursos de hardware y software

Hardware:

- Arduino UNO
- Módulo de internet ESP8266/Tarjeta NodeMCU
- Módulo HX711
- Sensor de presión

Software

- Android Studio
- Arduino IDE
- Firebase

Implementación del proyecto

En el presente proyecto se procedió a implementar un sistema conformado por un microprocesador, un sensor, dos módulos, una base de datos remota y una aplicación móvil. Para lograr montar el sistema se realizó el siguiente procedimiento:

1. Se crea la base de datos remota en Firebase de acuerdo con el diagrama entidad-relación planteado previamente. De esta página se obtienen las credenciales para acceder a esta base de datos desde el celular y el módulo ESP8266.
2. Se cortan dos pedazos de madera para colocarlos en la parte superior e inferior de la celda de carga mediante pernos. Esto se lo realiza para volver al sensor en un tipo de balanza y medir mejor el peso aplicado.
3. Conectar la celda de carga al módulo HX711 de la siguiente forma: alimentación de la celda en el pin E+ del módulo, fuente de la celda en el pin E- del módulo, señal positiva en el pin A- del módulo y señal negativa en el pon A+ del módulo.
4. Proceder a conectar el módulo HX711 con el Arduino Uno de la siguiente forma: pin de voltaje VCC del módulo en la ranura de 5V del Arduino, pin GND del módulo en la tierra común con el Arduino, pin de entrada digital (SCK) en el pin digital 8 del Arduino y la salida digital (DT) en el pin digital 7 del Arduino.
5. Teniendo este circuito armado, se procede a cargar el sketch de ejemplo “calibration” de la librería “HX711_ADC” en el Arduino Uno. Este sketch es para encontrar el valor de calibración de nuestra celda de carga con el módulo. Entonces se coloca encima de la balanza, creada en el paso dos, un objeto de masa conocida y esta masa se la ingresa al monitor serial (en gramos) para que el programa nos imprima el valor de calibración que se usará más adelante.

6. Ahora se procede a agregar el ESP8266 al circuito. Para esto se realizan las siguientes conexiones: el pin 3V3 del módulo con la ranura 3.3V del Arduino, el pin GND a la tierra común del circuito, el pin TX del módulo con el pin digital 3 del Arduino y el pin RX del módulo con el pin digital 4 del Arduino.
7. Con el circuito terminado, se procede a cargar los sketches correspondientes. En el Arduino se carga el sketch llamado “Sensor_Silla_Arduino_HX711.ino” donde la variable calValue será el valor obtenido en el paso cinco. Para el ESP8266 se cargará el sketch llamado “Sensor_Silla_ESP8266.ino” que tendrá la conexión con la base de datos remota y modificará el estado del asiento en tiempo real.
8. En este momento se tiene a todo el sistema funcionando, así que se diseña la aplicación móvil en Android Studio con conexión a la base de datos en Firebase para lo cual se crea un login para los usuarios y una página donde se podrá visualizar las mesas y sillas ocupadas en tiempo real.

Diseño de la interfaz de usuario

La interfaz de usuario fue enfocada a los meseros o recepcionistas de los restaurantes por lo tanto se realizó una interfaz simple pero llamativa y que presente la información de tal manera que pueda se leída de manera rápida.

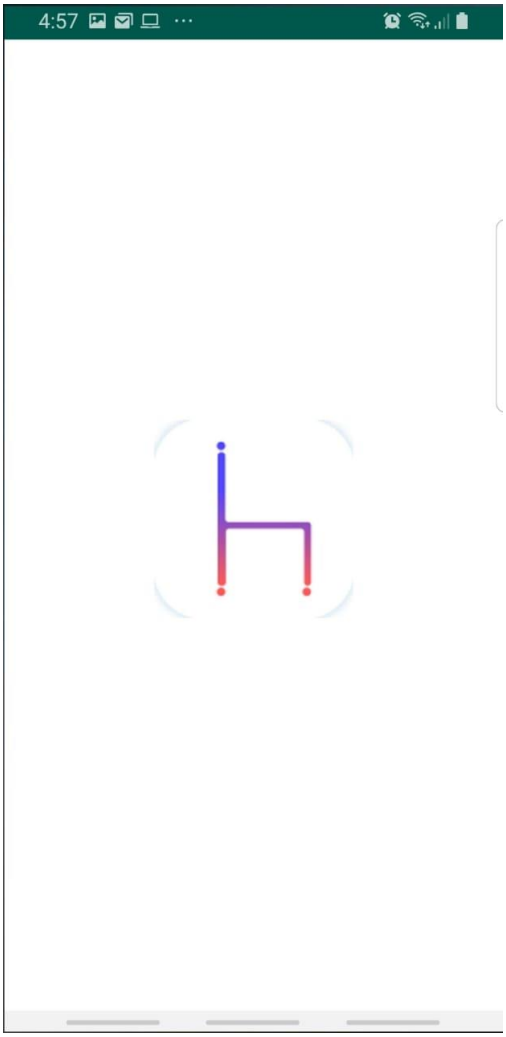

Ilustración 1: Pantalla inicial.

Esta es la pantalla que el usuario observará al momento de que inicie la aplicación móvil. Es solo el logo de la misma mientras se cargan los componentes necesarios para su funcionamiento.

Ilustración 2: Pantalla de Login.

Esta pantalla es la que sigue del anterior y permite el ingreso del usuario con su contraseña para poder entrar en la aplicación y visualizar los asientos disponibles. Este login también es una medida de seguridad para que personas sin autorización no puedan modificar la ubicación de las sillas en una mesa o acciones parecidas.

Ilustración 3: Pantalla principal.

En esta pantalla tenemos la acción principal de la aplicación que es la visualización de mesas con las sillas ocupadas de las mismas. Estas tienen un código de colores de semáforo. El verde significa que la mesa completa esta disponible, el amarillo que aquella mesa contiene sillas disponibles y el rojo que todas las sillas de la mesa están siendo ocupadas. Adicionalmente, se puede visualizar un contador para las sillas disponibles y otro para las mesas disponibles.

Diagramas

Diagrama de usos

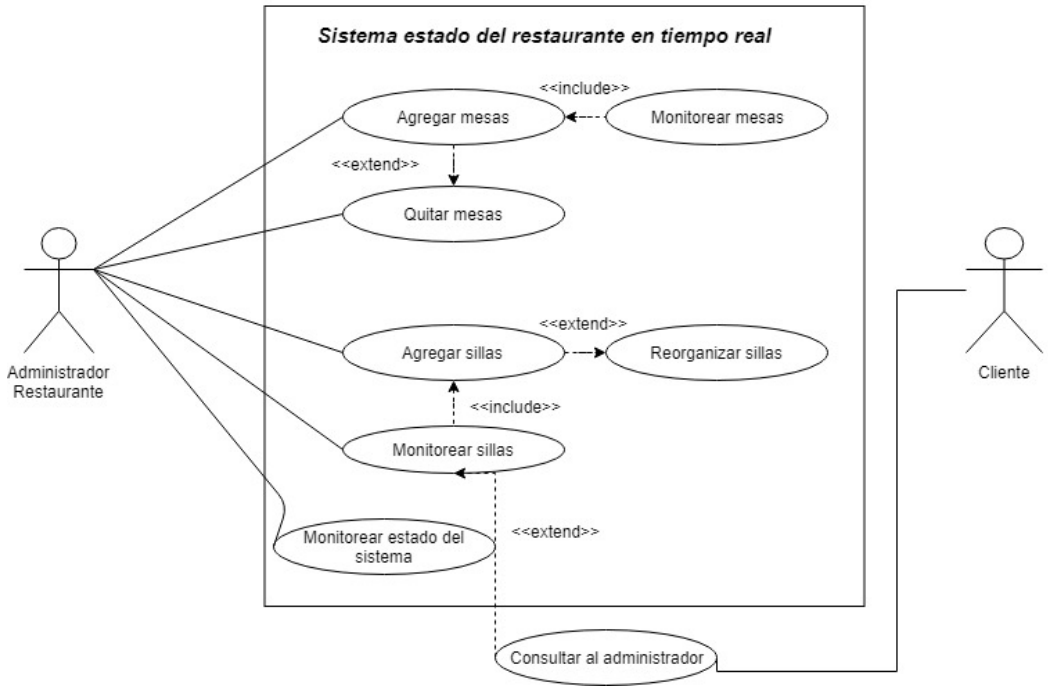

Ilustración 4: Diagrama de usos

En este diagrama de usos, planteamos a nuestro actor principal al administrador del restaurante. Este administrador es el encargado de monitorear la mesas y sillas disponibles, para poder realizar la distribución de los clientes. En la aplicación solo el administrador tiene permitido, agregar mesas, la cual estarán incluidas en el inventario, en caso de querer ampliar la capacidad del local. Junto con las mesas se pueden agregar sillas, de igual forma. Y es importante que también tiene la posibilidad de reorganizar las sillas, caso de que los clientes quieran agregar sillas a su mesa.

Diagrama entidad-relación

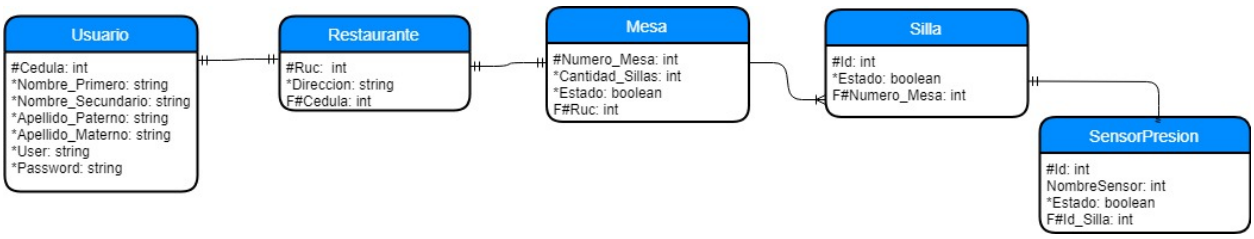

Ilustración 5: Diagrama entidad-relación

El diagrama entidad relación se colocó las entidades principales que conforman el sistema para en base a este esquema crear la base de datos.

Diagrama de despliegue

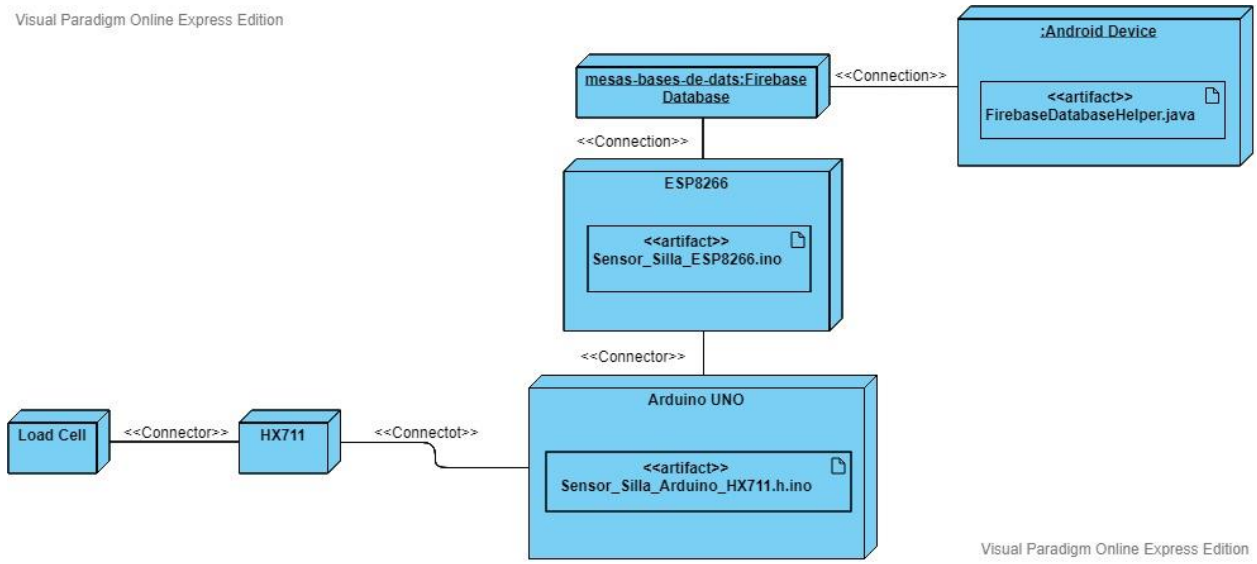

Ilustración 6: Diagrama de despliegue

En base al diagrama de despliegue se pudo reconocer todos los componentes de hardware que nuestro sistema va a requerir para solucionar el problema planteado.

Diagrama de red

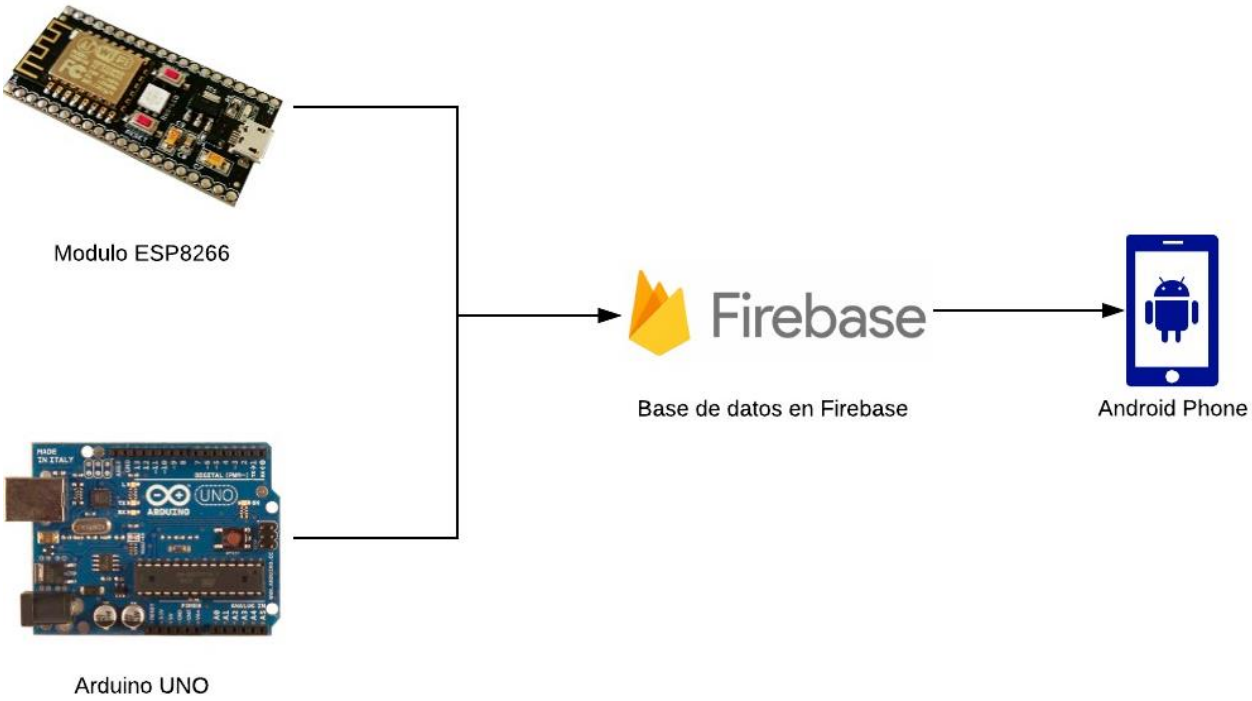

Ilustración 7: Diagrama de red

El proyecto implementa un Arduino Uno y un módulo ESP8266 para la actualización del estado del asiento en la base de datos remota Firebase. El Arduino Uno envía un dato al ESP8266 que puede ser un 1 o un 0 y en base a esto el módulo procede a cambiar la información almacenada en Firebase. Este resultado podrá ser visible a través de la aplicación móvil en tiempo real.

Diagrama de circuito

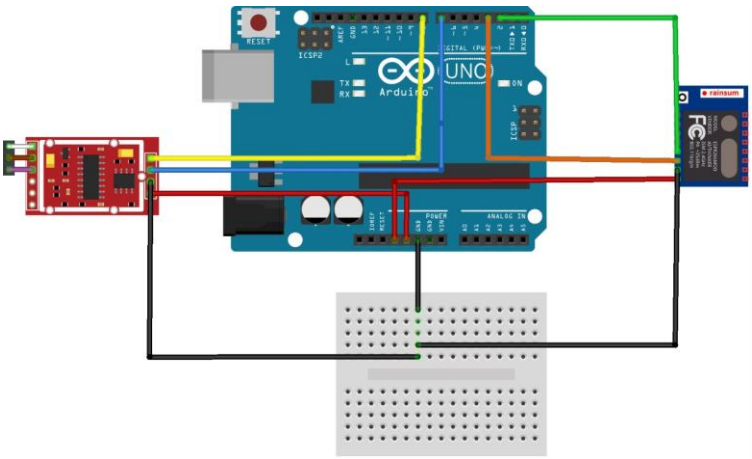

Ilustración 8: Diagrama de circuito

El diagrama de circuito nos presenta las conexiones que tiene los componentes de hardware que son el Arduino Uno, el módulo HX711 y el ESP8266 para conformar el sistema propuesto en este proyecto.

Diagrama UML de clases

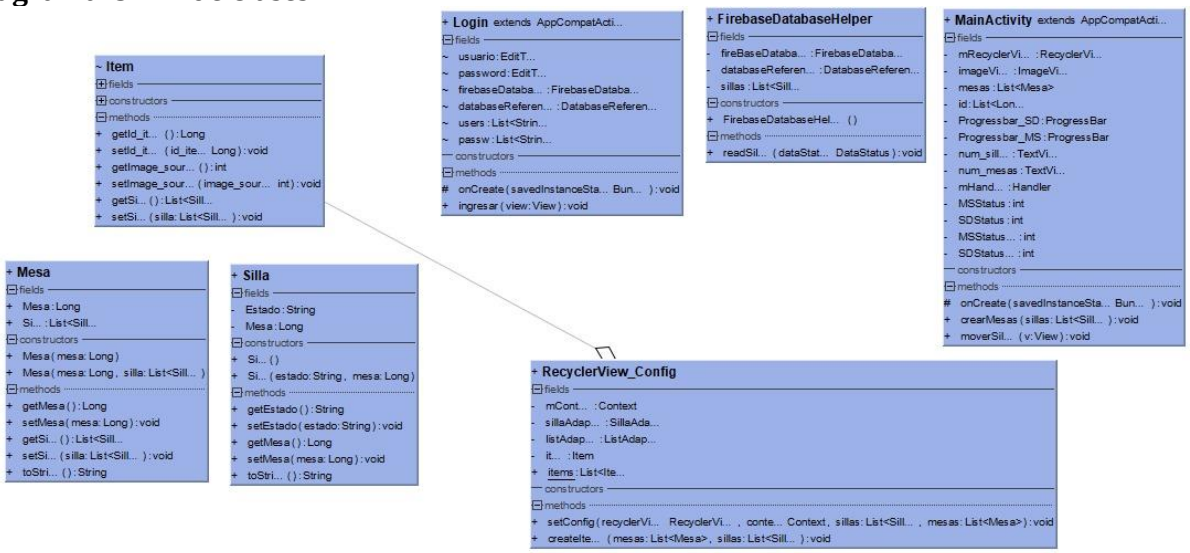

Ilustración 9: Diagrama de clases

El diagrama UML contiene las clases que fueron implementadas en la creación de la aplicación móvil del proyecto.

Descripción de la base de datos

Para diseñar de manera correcta la base de datos a utilizar se utilizó el modelo relacional Entidad-Relación (Ilustración 5) para organizar de mejor manera la información necesaria que nos provee el cliente y relacionar los campos entre sí para un optimó desempeño que a través de la lógica empleada de manera correcta, pueda ser sencillo su modificación en el futuro.

Para obtener el mejor rendimiento de base de datos, libre y gratuito, en tiempo real se utilizó la base de datos Realtime Database de Firebase, plataforma de desarrollo móvil en la nube de Google, que al ser "NoSQL" no trabaja con comandos SQL, sino con objetos JSON (Java Script Object Node), construyendo las tablas a partir de una relación entre nodos e hijos, como si de un árbol se tratara y asignándole cada conjunto de valores a una clave especifica.

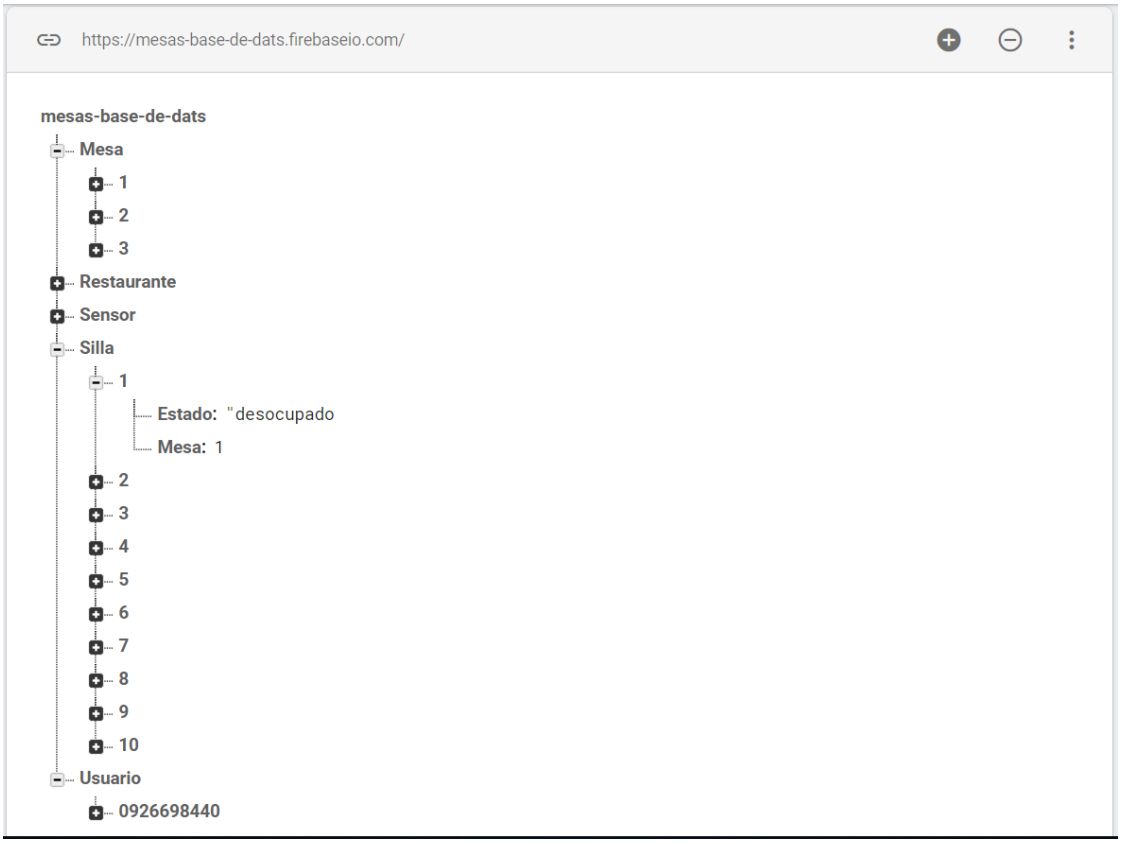

Ilustración 10: Base de datos remota en Firebase.

Explicación del código fuente

Para el proyecto se usaron dos tipos de códigos: para los módulos/sensores (hardware) y para el aplicativo móvil.

En el caso del código para el hardware se crearon dos archivos o sketch. El primer código fue cargado al Arduino para leer la información enviada por el módulo HX711 para luego ser tratada y enviada al ESP8266. El segundo fue cargado al ESP8266 el cual acorde a lo recibido por parte del Arduino modificaba la base de datos de Firebase.

A continuación, se detalla el código del sketch contenido en el Arduino:

```
//Sensor_Silla_Arduino_HX711.ino
#include <HX711_ADC.h>
#include <SoftwareSerial.h>
HX711_ADC LoadCell(7,8); //Se establece los pines que recibirán la lectura del HX711.
SoftwareSerial BT1(3,2); //Se establece la conexión serial entre el Arduino y el ESP8266.
int hxreading; //Se define la variable que contendrá la lectura del sensor HX711.
boolean ocupado = false; //Se define una variable booleana para definir el estado actual de la silla, al iniciar estará en false.
int normal = 10; //Se define el valor por default que el sensor va a presentar a una carga igual a cero.

void setup() {
  float calValue = 1120.48; //Factor de calibración del HX711.
  Serial.begin(115200);
  BT1.begin(115200); //Se abre el puerto serial a 115200 baudios.
  LoadCell.begin();
  long stabilisingtime = 2000; // tare precision can be improved by adding a few seconds of stabilising time.
  LoadCell.start(stabilisingtime);
  if(LoadCell.getTareTimeoutFlag()) {
    Serial.println("Tare timeout, check MCU>HX711 wiring and pin designations");
  }
  else {
    LoadCell.setCalFactor(calValue); // set calibration value (float).
    Serial.println("Startup + tare is complete");
  }
}
```

```
void loop() {
  LoadCell.update();
  hxreading = LoadCell.getData(); //Se toma la lectura actual del sensor.
  Serial.print("Load_cell output val: ");
  Serial.println(hxreading);
  if(hxreading>normal && ocupado==false){ //Si la lectura actual sobrepasa el
límite normal y el asiento esta desocupado(false) entra al bucle.
    Serial.println("- Asiento ocupado");
    Serial.println(hxreading);
    ocupado = true; //El estado del asiento cambia a ocupado (true).
    BT1.print("1\n"); //Se envia el valor de 1 al ESP por medio de la conexión
serial.
    Serial.flush();
  }
  else if(hxreading<=normal && ocupado==true){ //Si la lectura actual es menor al
límite normal y el asiento está ocupado(true) entra al bucle.
    Serial.println("- Asiento desocupado");
    Serial.println(hxreading);
    ocupado = false; //El estado del asiento cambia a desocupado (false).
    BT1.print("0\n"); //Se envia el valor de 1 al ESP por medio de la conexión
serial.
    Serial.flush();
  }
  delay(1000);
}
```

Este código que fue cargado al Arduino consiste en que se definen primero los pines que servirán para la transmisión de información con el módulo HX711 y con el ESP8266. Además, se define una variable para saber el estado (ocupado o desocupado) de la silla, de esta forma no siempre se mandará información al ESP salvo que este estado pueda cambiar. Debido a que el módulo HX711 envía un valor cuando no existe carga, se toma este valor como un punto de referencia para conocer en cual momento se ejerce presión en la silla. A continuación, en la función setup() se empieza la comunicación con el HX711, definiendo un valor de calibración que se obtuvo previamente, y la comunicación serial con el módulo wifi. En la función loop() se toma el valor enviado por el HX711, si este valor supera al valor de carga cero y el asiento estaba previamente desocupado entonces procede a enviar al ESP8266 el valor de “1” modificando el

valor de la variable de estado de la silla por ocupado (true). Por otro lado, si el valor recibido por el HX711 es menor al de carga cero y el asiento se encontraba previamente ocupado, se envía al ESP8266 el valor de “0” modificando el valor de la variable de estado de la silla por desocupado (false). Finalmente, antes de repetir el proceso se coloca un delay de un segundo para darle tiempo al buffer de vaciarse y evitar su sobrecarga.

El sketch que se encuentra cargado en el ESP8266 es el siguiente:

```
//Sensor_Silla_ESP8266.ino
#include <ESP8266WiFi.h>
#include <SoftwareSerial.h>
#include <FirebaseArduino.h> //Se incluye la librería para conectar con Firebase

#ifndef STASSID
#define STASSID "AndroidAPabef" //Se define la red wifi a la cual se quiere
conectar.
#define STAPSK "1234prueba" //Se define la contraseña de la red wifi.
#endif

#define FIREBASE_HOST "mesas-base-de-dats.firebaseio.com" //Link del API del
proyecto en Firebase
#define FIREBASE_AUTH "0f7BsbmAnZtZ6xvLBCUm4BPz8o2MlrPrwrOsSoDt" //Database
secret del proyecto en Firebase

const char* ssid     = STASSID; //Variable que contiene la red wifi.
const char* password = STAPSK; //Variable que contiene la contraseña de la red
wifi.
const String silla = "Silla/1/Estado"; //Se define la silla a la cual pertenece
este sensor.
String estado; //Se define la variable el cual contendrá el estado de la silla
sea ocupado o desocupado.

void setup() {
  Serial.begin(115200);
  //Se comienza la conexión a la red wifi.
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
```

```
  /* Explicitly set the ESP8266 to be a WiFi-client, otherwise, it by default,
     would try to act as both a client and an access-point and could cause
     network-issues with your other WiFi-devices on your WiFi-network. */
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) { //Se crea un bucle del cual no se
saldrá hasta que se logre conectar con la red wifi definida.
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

  Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH); //Se conecta a la base de datos
de Firebase.
  Serial.println("Conexión a Firebase exitosa");
  delay(1000);
}

void loop() {
  Serial.flush();
  String hxreading = "" ;
  if (Serial.available() ){
    char c = Serial.read(); //Se lee cada caracter enviado por comunicación
Serial
    while ( c != '\n' ){ //Hasta que el caracter sea diferente a "\n".
      hxreading = hxreading + c ; //Se completa el String enviado por
comunicación Serial.
      c = Serial.read();
    }
     Serial.println(hxreading);
  }
  else {return;}
  delay(500);
  if(hxreading == "1" || hxreading == "0"){
    if(hxreading=="1"){ //Si el mensaje recibido es 1 entonces estado será
ocupado.
      estado = "ocupado";
    }
    else{ //Si el mensaje recibido es 0 entonces estado será desocupado.
      estado = "desocupado";
```

```
    }
    Serial.println(estado);
    Firebase.setString(silla,estado); //Se modifica el estado de la silla en la
base de datos de Firebase.
  }
  delay(1000);
}
```

El código se apretura con la definición de variables de suma importancia puesto que estas contienen la información que permitirán la conexión con la red wifi y la base de datos de Firebase. Además, se define una variable string que contiene la ruta que será modificada en la base de datos, pero también contiene el id de la silla a la cual se encuentra asociada este sistema. En la función setup() se realiza la conexión a la red wifi y a la base de datos remota. Por otro lado, en la función loop() se lee cada carácter que llega por la comunicación serial con el Arduino y se va añadiendo a un string hasta que llegue el carácter "\n". Dependiendo del string final el ESP8266 va a modificar la base de datos de Firebase. Si el valor que llegó es "1" entonces el asiento al cual está asignado va a tener una modificación en su estado a "ocupado". Sin embargo, si el valor es "0" el asiento verá modificado su valor de estado a "desocupado".

Para la programación de la aplicación móvil se emplearon las siguientes clases:

Silla.java

```
package com.iChair.mesas;

public class Silla {
    private String Estado;
    private Long Mesa;

    public Silla(){
    }

    public Silla(String estado, Long mesa) {
        this.Estado = estado;
        this.Mesa = mesa;
    }

    public String getEstado() {
        return Estado;
    }

    public void setEstado(String estado) {
        this.Estado = estado;
    }


    public Long getMesa() {
        return Mesa;
    }

    public void setMesa(Long mesa) {
        this.Mesa = mesa;
    }

    @Override
    public String toString() {
        return "Silla{" +
                "Estado='" + Estado + '\'' +
                ", Mesa='" + Mesa + '\'' +
                '}';
    }
}
```

Mesa.java

```
package com.iChair.mesas;

import java.util.List;

public class Mesa {
    public Long Mesa;
    public List<Silla> Silla;
    //public int Estado;

    public Mesa(Long mesa){
        Mesa = mesa;
    }
    public Mesa(Long mesa, List<com.iChair.mesas.Silla> silla) {
        Mesa = mesa;
        Silla = silla;
    }

    public Long getMesa() {
        return Mesa;
    }

    public void setMesa(Long mesa) {
        Mesa = mesa;
    }

    public List<Silla> getSilla() {
        return Silla;
    }

    public void setSilla(List<com.iChair.mesas.Silla> silla) {
        Silla = silla;
    }

    @Override
    public String toString() {
        return "Mesa{" +
                "Mesa=" + Mesa +
                ", Silla=" + Silla +
                '}';
    }
}
```

Item.java

```
package com.iChair.mesas;

import java.util.List;

class Item {
    private Long id_item;
    private int image_source;
    private List<Silla> silla;

    public Item() {

    }

    public Item(Long id_item, int image_source, List<Silla> silla) {
        this.id_item = id_item;
        this.image_source = image_source;
        this.silla = silla;
    }

    public Long getId_item() {
        return id_item;
    }

    public void setId_item(Long id_item) {
        this.id_item = id_item;
    }

    public int getImage_source() {
        return image_source;
    }

    public void setImage_source(int image_source) {
        this.image_source = image_source;
    }

    public List<Silla> getSilla() {
        return silla;
    }

    public void setSilla(List<Silla> silla) {
        this.silla = silla;
    }
```

```
}
```

FirebaseDatabaseHelper.java

```
package com.iChair.mesas;
import android.util.Log;
import androidx.annotation.NonNull;
import com.google.android.material.snackbar.Snackbar;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import java.util.ArrayList;
import java.util.List;
public class FirebaseDatabaseHelper {
    private FirebaseDatabase fireBaseDatabase;
    private DatabaseReference databaseReference;
    private List<Silla> sillas = new ArrayList<>();

    /**
     * Constructor para el gestor de base de datos de Firebase.
     * Nos permite instanciar las variables creadas para gestionar las claves y
valores de los nodos
     * en la base de datos de Firebase.
     */
    public FirebaseDatabaseHelper(){
        fireBaseDatabase = FirebaseDatabase.getInstance();
        databaseReference = fireBaseDatabase.getReference();
    }

    /**
     * Interfaz para la sincronización entre el main Thread y la actividad
asincronica del Listener
     * que escucha los cambios en la base de datos.
     */
    public interface DataStatus{
        void DataisLoaded(List<Silla> sillas,List<String> keys);
        void DataisUpdated();
    }

    /**
```

```
     * Metodo que lee las sillas del nodo respectivo "Silla" del RealtimeDatabase
de Firebase que
     * recorre las claves y valores de cada objeto, guardandola en una lista.
     * @param dataStatus Interfaz a implementar para sincronización en tiempo
real.
     */
    public void readSillas(final DataStatus dataStatus){
        databaseReference.child("Silla").addValueEventListener(new
ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                sillas.clear();
                List<String> keys = new ArrayList<>();
                for(DataSnapshot keyNode: dataSnapshot.getChildren()){
                   keys.add(keyNode.getKey());
                   Silla silla = keyNode.getValue(Silla.class);
                   sillas.add(silla);
                }
                dataStatus.DataisLoaded(sillas,keys);
            }


            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {

            }
        });
    }

}
```

RecyclerView_Config.java

```
package com.iChair.mesas;

import android.content.Context;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ImageView;
import android.widget.ListView;
```

```
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import java.util.ArrayList;
import java.util.List;


public class RecyclerView_Config {
    private Context mContext;
    private SillaAdapter sillaAdapter;
    private ListAdapter listAdapter;
    private Item item;
    public static List<Item> items = new ArrayList<>();

    /**
     * Se setea la configuración necesaria para mostrar la información en tiempo
real de la pantalla
     * correspiende a los datos de sillas y mesas.
     * @param recyclerView
     * @param context
     * @param sillas
     * @param mesas
     */
    public void setConfig(RecyclerView recyclerView, Context context, List<Silla>
sillas, List<Mesa> mesas){
        mContext = context;
        createItems(mesas,sillas);
        sillaAdapter = new SillaAdapter(items);
        recyclerView.setLayoutManager(new LinearLayoutManager(context));
        recyclerView.setAdapter(sillaAdapter);
    }

    /**
     * Se crean los items correspondientes de acuerda a las mesas y las sillas
asignadas a cada mesa.
     * @param mesas
     * @param sillas
     */
    public void createItems(List<Mesa> mesas,List<Silla> sillas){
        items.clear();
        for(Mesa mesa:mesas){
            Item item = new Item();
```

```
            item.setId_item(mesa.getMesa());
            item.setImage_source(R.mipmap.ic_logo);
            List<Silla> sillaList = new ArrayList<>();
            for(Silla silla: sillas){
                if(silla.getMesa().equals(item.getId_item())){
                    sillaList.add(silla);
                }
            }
            item.setSilla(sillaList);
            items.add(item);
        }

    }

    /**
     * Clase interna diseñada para crear una View correspondiente para cada Mesa.
     */
    class SillaView extends RecyclerView.ViewHolder{
        private TextView mMesa;
        private ListView listView;
        private ImageView imageView;
        private String key;

        /**
         * Constructor para la instancias de widgets utilizadas en el layout
silla_list_item.
         * @param parent
         */
        public SillaView(ViewGroup parent){

super(LayoutInflater.from(mContext).inflate(R.layout.silla_list_item,parent,false
));
            mMesa = (TextView) itemView.findViewById(R.id.txt_mesa);
            listView = (ListView) itemView.findViewById(R.id.list_silla);
            imageView = (ImageView) itemView.findViewById(R.id.imageView);

        }

        /**
         * Metodo bind para designar a cada widget el valor correspondiente segun
el Item creado.
         * @param mesa
         * @param image
         * @param key
         * @param sillas
```

```
         */
        public void bind(Long mesa,int image,String key,List<Silla> sillas){
            String id = "Mesa " + mesa + " :";
            imageView.setImageResource(image);
            listView.setAdapter(new ListAdapter(sillas));
            mMesa.setText(id);
            this.key = key;
        }
    }

    /**
     * Adaptador del recycler view para la lista creada de Items dentro de la
vista, que recorre
     * la lista de items y la va agregando a la vista principa.
     */
    class SillaAdapter extends RecyclerView.Adapter<SillaView>{
        private List<Mesa> Mesas;
        private List<Silla> Sillas;
        private List<Item> items;

        public SillaAdapter(List<Mesa> Mesas,List<Silla> Sillas) {
            this.Mesas = Mesas;
            this.Sillas = Sillas;
        }

        public SillaAdapter(List<Item> items) {
            this.items = items;
        }


        @NonNull
        @Override
        public SillaView onCreateViewHolder(@NonNull ViewGroup parent, int
viewType) {
            return new SillaView(parent);
        }

        @Override
        public void onBindViewHolder(@NonNull SillaView holder, int position) {

holder.bind(items.get(position).getId_item(),items.get(position).getImage_source(
),String.valueOf(items.get(position).getId_item()),items.get(position).getSilla()
);
        }
```

```
        @Override
        public int getItemCount() {
            return items.size();
        }
    }

    /**
     * Adaptador de ListView que recorre la lista de sillas de un Item para
colocarlos de manera
     * consecutiva dentro del widget correspondiente.
     */
    class ListAdapter extends BaseAdapter{
        private List<Silla> sillas;

        public ListAdapter(List<Silla> sillas) {
            this.sillas = sillas;
        }

        @Override
        public int getCount() {
            return sillas.size();
        }

        @Override
        public Object getItem(int i) {
            return sillas.get(i);
        }

        @Override
        public long getItemId(int i) {return 0;}

        @Override
        public View getView(int i, View view, ViewGroup viewGroup) {
            Silla silla = (Silla) getItem(i);
            view =
LayoutInflater.from(mContext).inflate(R.layout.list_view,null);
            TextView txt_estado = (TextView)
view.findViewById(R.id.txt_silla_estado);
            String estado = "Silla: " + silla.getEstado();
            txt_estado.setText(estado);

            return view;
        }
    }
}
```

Login.java

```
package com.iChair.mesas;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

import java.util.ArrayList;
import java.util.List;

public class Login extends AppCompatActivity {
    EditText usuario;
    EditText password;
    FirebaseDatabase firebaseDatabase;
    DatabaseReference databaseReference;
    List<String> users;
    List<String> passw;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTheme(R.style.AppTheme_NoActionBar);
        setContentView(R.layout.activity_login);
        users = new ArrayList<>();
        passw = new ArrayList<>();
        usuario = findViewById(R.id.editText2);
        password = findViewById(R.id.editText);
        obtenerUsuarios();
    }

    /**
     * Metodo que guarda los usuarios y clave en un arrayList del todos los nodos
en la tabla "nodos".
     */
```

```
    public void obtenerUsuarios(){
        firebaseDatabase = FirebaseDatabase.getInstance();
        databaseReference =
firebaseDatabase.getReference("Usuario").child("0926698440");
        databaseReference.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                users.add(dataSnapshot.child("usuario").getValue().toString());
                passw.add(dataSnapshot.child("clave").getValue().toString());
            }

            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {

            }
        });
    }

    /**
     * Metodo del Botón ingresar para comparar las credenciales obtenidas en el
meotod obtenerUsuarios()
     * y los ingresados por el usuario a traves de los cuadros edit text.
     * @param view
     */
    public void ingresar(View view) {
        String user = usuario.getText().toString();
        String pass = password.getText().toString();
        if(user.equals(users.get(0)) && pass.equals(passw.get(0))){
            Intent i = new Intent(this,MainActivity.class);
            startActivity(i);
        }
     }
}
```

MainActivity.java

```
package com.iChair.mesas;

import androidx.annotation.RequiresApi;
import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
```

```
import android.content.Intent;
import android.os.Build;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.view.View;
import android.widget.ImageView;
import android.widget.ListView;
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.List;

public class MainActivity extends AppCompatActivity {
    private RecyclerView mRecyclerView;
    private ImageView imageView;
    private List<Mesa> mesas = new ArrayList<>();
    private List<Long> id = new ArrayList<>();
    //PROGRESS BAR DE LAS SILLAS DISPONIBLES
    private ProgressBar Progressbar_SD;
    private ProgressBar Progressbar_MS;
    //Numero de sillas disponibles
    private TextView num_sillas;
    //Numero de mesas disponibles
    private TextView num_mesas;
    //PROGRESS BAR DE LAS SILLAS EN USO
    private Handler mHandler= new Handler();
    //PROGRESS BAR DE LAS MESAS DISPONIBLES
    private int MSStatus;
    private int SDStatus;
    private int MSStatusfin;
    private int SDStatusfin;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTheme(R.style.AppTheme_NoActionBar);
        setContentView(R.layout.activity_main);
        mRecyclerView = (RecyclerView) findViewById(R.id.recyclerview_sillas);
        imageView = (ImageView) findViewById(R.id.imageView);
        new FirebaseDatabaseHelper().readSillas(new
FirebaseDatabaseHelper.DataStatus() {
            @Override
            public void DataisLoaded(List<Silla> sillas, List<String> keys) {
```

```
                MSStatus=0;
                SDStatus=0;
                MSStatusfin=0;
                SDStatusfin=0;
                crearMesas(sillas);
                new
RecyclerView_Config().setConfig(mRecyclerView,MainActivity.this,sillas,mesas);
                estadoMesas(mesas,RecyclerView_Config.items);
                /**
                 * Se actualiza la cantidad de sillas y mesas disponibles a
medida que existen cambios
                 * en la base de datos.
                 * Se setea los valores maximos correspondientes de mesa y silla
disponible.
                 */
                Progressbar_MS= findViewById(R.id.progress_mesas);
                //decimos cual es el maximo numero de mesas
                Progressbar_MS.setMax(RecyclerView_Config.items.size());
                Progressbar_SD=(ProgressBar) findViewById(R.id.progress_sillas);
                //decimos cual es el maximo de la barra de sillas
                Progressbar_SD.setMax(sillas.size());
                num_sillas=findViewById(R.id.num_sillas);
                num_mesas=findViewById(R.id.num_mesas);
                //numero de sillas disponibles
                //Setea el numero de sillas disponibles
                num_sillas.setText(String.valueOf(SDStatusfin));
                //Setea el numero de mesas disponibles
                num_mesas.setText(String.valueOf(MSStatusfin));

                new Thread(new Runnable() {
                    @Override
                    public void run() {
                        while(SDStatus<SDStatusfin){
                            SDStatus++;
                            android.os.SystemClock.sleep(50);
                            mHandler.post(new Runnable() {
                                @RequiresApi(api = Build.VERSION_CODES.N)
                                @Override
                                public void run() {
                                    Progressbar_SD.setProgress(SDStatus,true);
                                }
                            });
                        }
                    }
                }).start();
```

```
                new Thread(new Runnable() {
                    @Override
                    public void run() {
                        while(MSStatus<MSStatusfin){
                            MSStatus++;
                            android.os.SystemClock.sleep(50);
                            mHandler.post(new Runnable() {
                                @RequiresApi(api = Build.VERSION_CODES.N)
                                @Override
                                public void run() {
                                    Progressbar_MS.setProgress(MSStatus,true);
                                }
                            });
                        }
                    }
                }).start();
            }
            @Override
            public void DataisUpdated() {
            }
        });
    }

    /**
     * Crea los objetos mesas de acuerdo al numéro diferentes de id que se
obtienen de las sillas y
     * se setea una lista de sillas en el objeto mesa correspondiente al mismo.
     * @param sillas
     */
    public void crearMesas(List<Silla> sillas){
        List<Silla> sillasl = new ArrayList<>();
        for(Silla silla :sillas){
            if(!id.contains(silla.getMesa())){
                id.add(silla.getMesa());
                Mesa mesa = new Mesa(silla.getMesa());
                mesas.add(mesa);

            }
        }
        for(Long id_mesa:id){
            for(Silla silla: sillas){
                if(silla.getMesa().equals(id_mesa)){
                    sillasl.add(silla);
                }
            }
```

```
            mesas.get(id.indexOf(id_mesa)).setSilla(sillasl);
            sillasl.clear();
        }
    }

    /**
     * Analisis en tiempo real del estado de silla de las correspodientes mesas
para establecer el
     * estado de una mesa y asignarle una respectiva imagen que represente
visualmente si esta
     * ocupada.
     * @param mesas
     * @param items
     */
    public void estadoMesas(List<Mesa> mesas,List<Item> items){
        for (Mesa mesa: mesas) {
            int estado=0;
            for (Silla silla : items.get(mesas.indexOf(mesa)).getSilla()) {
                if (silla.getEstado().equals("ocupado")) {
                    estado++;
                } else{
                    SDStatusfin ++;
                }
            }
            if(estado==0){
                MSStatusfin ++;
                for (Item item : items) {
                    if (mesa.getMesa().equals(item.getId_item())) {
                        item.setImage_source(R.mipmap.mesa_desocupada);
                    }
                }
            }
            else if (estado==items.get(mesas.indexOf(mesa)).getSilla().size()){
                for (Item item : items) {
                    if (mesa.getMesa().equals(item.getId_item())) {
                        item.setImage_source(R.mipmap.mesa_ocupada);
                    }
                }
            }
            else {
                for (Item item : items) {
                    if (mesa.getMesa().equals(item.getId_item())) {
                        item.setImage_source(R.mipmap.mesa_semiocupada);
                    }
                }
```

```
            }
        }

    }
}
```

Empezando el desarrollo del aplicativo móvil en el ambiente de desarrollo de Android se debe vincular el proyecto con Firebase, siguiendo los pasos proporcionados por el propio servicio de Google para el uso de la base de datos en tiempo real, seremos capaces de actualizar la base de datos de manera remota gracias a la implementación de sensores en cada una de las sillas del restaurante, utilizando celdas de carga que enviarán una señal a través del módulo HX711 hacia el microcontrolador Arduino que al interpretarla se enviara una actualización a la base de datos por medio del módulo de Wi-Fi ESP8266, posteriormente, se debe tener claves de acceso para la base de datos de Firebase.

Para poder acceder al sistema funcional del aplicativo móvil se implementó un sistema de ingreso por clave y usuario que al ser validados se despliega una vista del restaurante con el qué está vinculado el usuario, donde le aparecen todas las mesas y sillas correspondientes. En la parte superior de la aplicación se muestran el total de lugares y asientos disponibles, que se actualizan a medida que se llena el restaurante, lo llamaremos barras de disponibilidad.

En la aplicación, las sillas tienen estados: “ocupado” o “desocupado” y las mesas poseen un ícono representativo con distintos fondos: “verde”, disponible; “amarillo”, semi ocupada;” rojo”, llena. Si al menos una silla de cierta mesa está ocupada, se actualizará el estado de la silla, el ícono representativo de la mesa y la barra de disponibilidad. De esta manera, quien administre la aplicación tendrá a su disponibilidad toda la información en tiempo real sobre los lugares disponibles, ocupados, sillas libres, entre otros, en la palma de su mano.

Análisis del código de Android

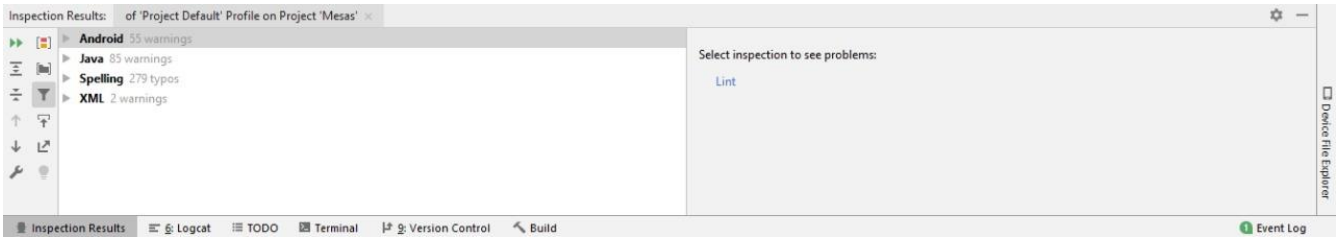

Ilustración 11: Análisis del código de la aplicación móvil

Análisis de presupuesto

(1) Arduino UNO: $20

(2) Módulo HX711: $2,25

(3) Módulo ESP8266: $12,5

(4) Sensor Load Cell: $4,5

Costo total: $39.25

Podemos notar que el componente más costoso es el Arduino UNO, sin embargo, otras placas que cumplen la misma función son más costosas, como Raspberry Pi. Por otra parte, se puede adquirir un Arduino no de la compañía UNO, si no de una marca genérica, lo cual aumenta el riesgo de que este falle o dure menos para desempeñar sus funciones.

El módulo ESP8266 Representa un costo considerable, se puede usar una versión anterior, sin embargo, el módulo ESP8266 cuenta con la implementación de librerías más eficientes para conectar con la base de datos Firebase.

Por último, se puede considerar adquirir un sensor de presión mucho más sofisticado que la celda de carga. Esto nos evitaría adquirir un módulo HX711 por cada celda.

En conclusión, para un proyecto de índole universitario, un costo menor a $50 es considerable aceptable considerando los ingresos que puede tener un universitario.

Análisis de pruebas realizadas

Como se puede observar en la Ilustración 3, el estado de una silla es ocupado para la Mesa#2, por lo tanto, esta cambió de ícono, mostrando una mesa con fondo de color amarillo, que representa el estado de “semi ocupada” para las mesas. El cambio de estado sucedió en el instante en que la celda de carga “sintió” una presión sobre él, por lo que inmediatamente cambia el estado de la silla, de igual manera, el estado de la mesa. Una vez que no hay presión sobre la celda, el estado cambia automáticamente de “ocupado” a “desocupado”, refrescando la interfaz del usuario en su estado original.

Además, se comprobó qué cambiando los datos de manera manual la base se actualizaba instantáneamente, así mismo la figura de la mesa y el estado de las sillas, sin embargo, las barras de disponibilidad se reiniciaban en cada cambio de estado provocando una sensación de reinicio por parte de la vista en donde se observan como se llena una y otra vez cada que se modifica un dato relevante dentro de la base de datos de Firebase. Para un modelo real, esta funcionalidad de en tiempo real resulta factible puesto que permite conocer con exactitud cuantas sillas se encuentran disponibles para un comensal.

Adicionalmente, debido al uso de una base de datos remota, es posible acceder a esta información de sillas disponibles en cualquier momento y en cualquier parte con una conexión a internet siempre y cuando se posea un usuario registrado en la aplicación.

Conclusiones

- La aplicación de bases remotas en Firebase nos permiten obtener los datos en tiempo real por lo que facilitan la aplicación del concepto “Internet de las cosas” porque las reacciones de los dispositivos y la visualización de la información será en el menor tiempo posible.
- La aplicación del microprocesador Arduino Uno con el módulo ESP8266 permite la conectividad con una red lo cual facilita el tráfico de información entre los distintos dispositivos de un sistema.
- La correcta distribución del tiempo durante un proyecto de trabajo lleva a que el mismo evolucione de manera progresiva y ordenada, por lo tanto, puede tener un mayor valor agregado en el desarrollo y puede divergir en más funcionalidades, siendo atractivo para el usuario final.
- La implementación de sistemas telemáticos en los proyectos de hoy en día están ganando terreno de manera descomunal ya que el internet de las cosas, aprendizaje automático y aprendizaje profundo están formando parte de la cuarta revolución industrial la cual junta la tecnología básicamente en cualquier ámbito: investigación, economía, seguridad, entre otros. Además, que este tipo de proyectos son progresivos y actualizables por lo cual siempre pueden ser mejorados.

Referencias bibliográficas

GreatScott! (3 de Diciembre de 2017). *Electronic Basics #33: Strain Gauge/Load Cell and how to use them to measure weight*. Obtenido de Youtube: https://www.youtube.com/watch?v=lWFiKMSB_4M&feature=youtu.be

Rashmin, R. (22 de Julio de 2018). *Arduino to Android , Real Time Communication For IoT with Firebase*. Obtenido de Coinmonks: https://medium.com/coinmonks/arduino-to-android-real-time-communication-for-iot-with-firebase-60df579f962

rgmosqueda007. (29 de Noviembre de 2016). *¿Cómo conectar a Arduino con NodeMCU por serial?* Obtenido de Euritium: https://euritium.wordpress.com/2016/11/29/2-como-conectar-a-arduino-con-nodemcu-por-serial/

Sparkfun. (s.f.). *24-Bit Analog-to-Digital Converter (ADC) for Weigh Scales.* Obtenido de Sparkfun: http://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf

Anexos

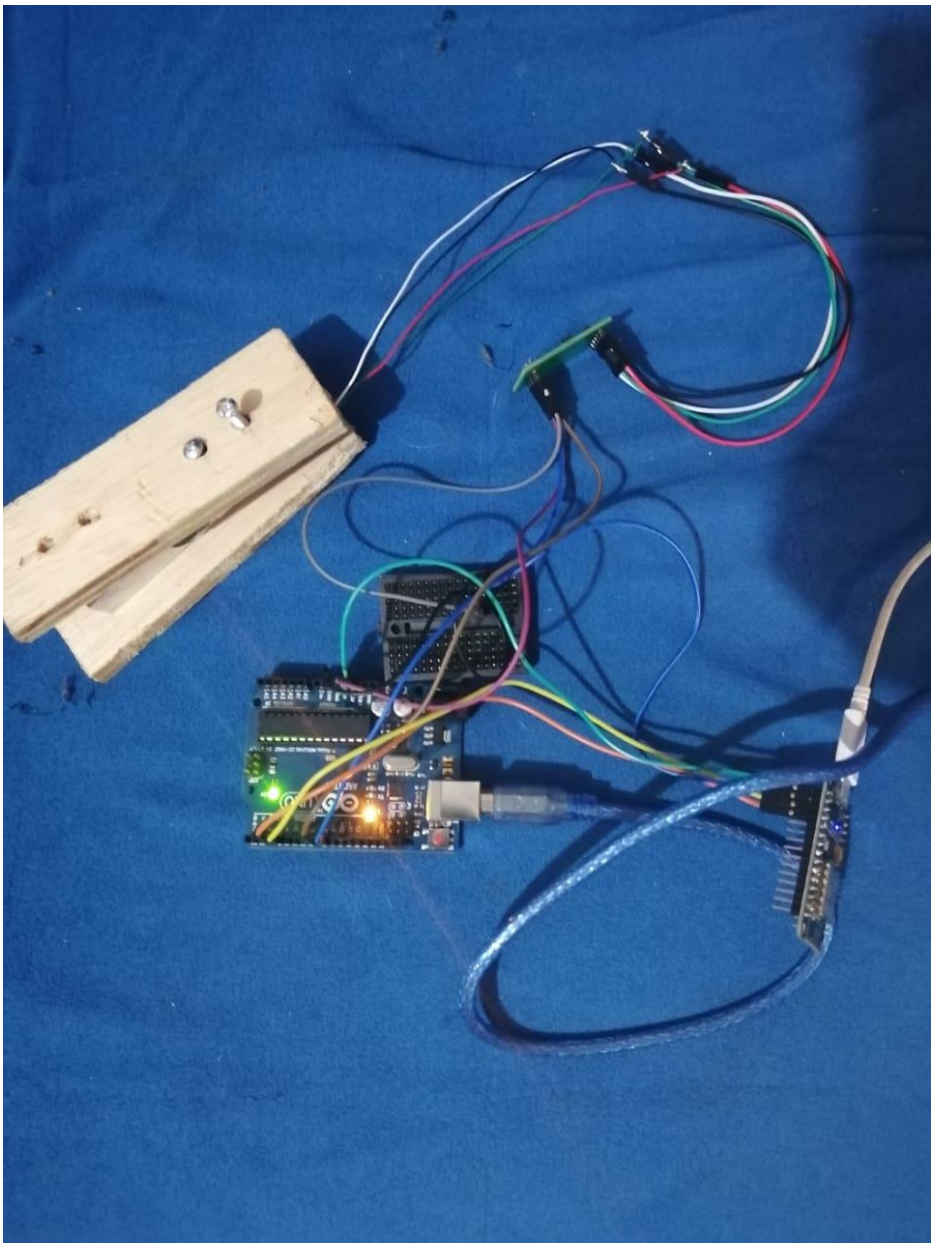

Ilustración 12: Prototipo del sistema propuesto.