ESCUELA POLITECNICA - JOSE TOSCHE

TECNICATURA ELECTRONICA - 7mo AÑO

Proyecto Final

Internet Of Things

Implementación para un Hogar

Integrantes:

Bonaccini Alejandro

Laime David

Agustín Poza

Profesores:

Mariaca Omar

Pistonesi Alejandro

Venanzoni Mauricio

18/11/2020

INDICE

1. Introducción

El proyecto presenta el diseño y la implementación de un sistema IoT para las funciones comunes de un hogar, como la medición de temperatura, el control de una alarma, medición de monóxido de carbono, control sobre un control de riego y la posibilidad de controlar las luces de determinados ambientes de la casa.

1.1 Objetivos Generales

Diseñar e implementar un sistema IoT (Internet Of Things) para un hogar convencional

1.1.1 Objetivos Específicos

- Utilizar los siguientes dispositivos para la comunicación: Xbee y el Rasberry
- Monitorear los siguientes datos: la temperatura, el monóxido de carbono, las luces, controlar el riego y controlar la alimentación de la alarma.
- Lograr una comunicación fluida entre los dispositivos (sensores, Xbee, Arduino, Raspberry, Firebase y Android).
- Guardar los datos en la nube.
- Implementar los lenguajes de programación necesarios para cada dispositivo
- Poder observar y controlar los datos de los sensores a través de un dispositivo Android.

2. Primeras Pruebas

Realizamos nuestras primeras pruebas tratando de abarcar cada nodo de comunicación para luego poder unirlos todos.

2.1 Comunicación entre Rasberry y Arduino

Decimos empezar por entablar una comunicación entre el Rasberry y Arduino

2.1.1 Prueba LED

La comunicación se realizará a través de un USB. Y la primera prueba consistirá en encender un LED, el Rasberry enviara un dato y mientras tanto, el Arduino leerá el dato que le fue enviado y actuara dependiendo del dato.

```python
import serial

#nombre del dispositivo serial : dmesg | grep -v disconnect | grep -Eo
arduino = serial.Serial('/dev/ttyACM0', 9600)


while True:
	comando = input('Introduce un comando: ') #Input
	arduino.write(comando.encode()) #Mandar un comando hacia Arduino
	if comando == 'A':
		print('LED ENCENDIDO')
	elif comando == 'a':
		print('LED APAGADO')


arduino.close() #Finalizamos la comunicacion
```

```arduino
int LED = 13;
const int Sensor = A0;

void setup() {
  Serial.begin(9600);
  pinMode(LED, OUTPUT);
}

void loop() {
  char VE = Serial.read();
  switch(VE) {
    case 'A': digitalWrite(LED, HIGH);
    break;
    case 'a': digitalWrite(LED, LOW);
    break;
  }
}
```

No hubo demasiados problemas, funciono como lo esperábamos, así que decidimos probar enviando un dato análogo desde el Arduino.

2.1.2 Prueba con un dato análogo

En esta prueba el Arduino estará recolectando un dato analógico del LM35 para luego enviárselo al Rasberry y que lo muestre. Por suerte, no hubo mayores complicaciones.

```
int LED = 7;
const int Sensor = A0;

void setup() {
  Serial.begin(9600);
  pinMode(LED, OUTPUT);
}

void loop() {
  LeerLM35();
  delay(1000);
}

void LeerLM35() {
  int value = analogRead(Sensor);
  float millivolts = (value / 1023.0) * 5000;
  float celsius = millivolts / 10;
  Serial.println(celsius);
}
```

2.1.3 Prueba con un LED y un dato análogo

Una vez que por singular pudimos enviar y recibir datos entre estos, era momento de que pongamos a prueba las dos pruebas a la vez. Que el Arduino sea capaz de leer el dato que le envía el Rasberry, y que a su vez el Arduino envié una actualización de la temperatura. Aun no pudimos lograrlo.

```
int LED = 7;
const int Sensor = A0;

void setup() {
  Serial.begin(9600);
  pinMode(LED, OUTPUT);
}

void loop() {
  char VR = Serial.read();
  switch(VR) {
    case 'A': digitalWrite(LED, HIGH);
    break;
    case 'a': digitalWrite(LED, LOW);
    break;
  }
  LeerLM35();
  delay(1000);
}

void LeerLM35() {
  int value = analogRead(Sensor);
  float millivolts = (value / 1023.0) * 5000;
  float celsius = millivolts / 10;
  Serial.println(celsius);
}
```

2.2 Desarrollo de la aplicación Android

En la parte de la comunicación del usuario, con lo que vendría siendo la base de datos, tenemos, a la aplicación Android, la cual es desarrollada con el IDE de Android Studio, utilizando principalmente el lenguaje de programación Kotlin para el desarrollo de back-end y XML para el desarrollo de la UI en el front-end.

Para el desarrollo de la aplicación hemos tenido que aprender el lenguaje de Kotlin desde cero, el cual es similar en algunos aspectos a Java. También hemos tenido que aprender a utilizar XML el cual tiene ciertas similitudes con HTML. Todo esto sería el ecosistema de Android Studio junto a sus emuladores

2.2.1 Comunicación entre el Android Studio y Firebase

Para la comunicación de la aplicación android con Firebase lo primero que necesitamos hacer es descargar un archivo con las credenciales para que nuestra aplicación pueda acceder a la base de datos en tiempo real, luego configurar los permisos en nuestra aplicación, importar librerías y plugins. Una vez conectada nuestra aplicación con la base de datos de Firebase, podemos enviar y recibir datos, así como organizar una jerarquía de los mismos, en nuestros últimos avances hemos logrado enviar exitosamente los datos al Firebase, los problemas que hemos tenido han sido al momento de recibir datos del Firebase a nuestra aplicación, pero es algo lo cual pronto esperamos resolver, y lo cual sería un punto clave para terminar la parte del back-end de nuestra aplicación, ya que con el envío y la escucha de datos, tanto Booleanos para las luces y los dispositivos que solo necesitan de 2 estados posibles, como datos Strings como vendrían siendo la temperatura, el nivel de CO2, entre otros.

```
class MainActivity : AppCompatActivity() {
    val database = Firebase.database

    var refCasa = database.getReference( path: "home")
   lateinit var refLuz : DatabaseReference
   lateinit var refLuzCocina : DatabaseReference
    lateinit var refLuzCocinaEstado : DatabaseReference
```

En esta sección inicializamos y clasificamos a nuestros datos

```
refLuz = refCasa.child( pathString: "luces")
refLuzCocina = refLuz.child( pathString: "luz_cocina")
refLuzCocinaEstado = refLuzCocina.child( pathString: "estado")
```

Luego definimos la jerarquía en la que se van a estructurar nuestros datos

```
val toggle: ToggleButton = findViewById(R.id.toggleButton)
toggle.setOnCheckedChangeListener { _, isChecked ->
    if (isChecked) {
        refLuzCocina.setValue(Boolean)
    } else {
        refLuzCocina.setValue(Boolean)
    }
}
val postListener = object : ValueEventListener {
    override fun onDataChange(dataSnapshot: DataSnapshot) {
        // Get Post object and use the values to update the UI
        val check = dataSnapshot.getValue<Boolean>()
        // ...
    }

    override fun onCancelled(databaseError: DatabaseError) {
    }
}
refLuzCocina.addValueEventListener(postListener)
```

Y finalmente asociamos un botón a una luz y le damos la función para que cuando hagamos un click en el botón cambie de estado y ese estado lo envíe a nuestra base de datos, lo que vemos abajo es un EventListener, que se utiliza para recibir los datos de nuestra base de datos, por ahora no lo hemos podido hacer funcionar.

2.3 Comunicación entre el Arduino y Xbee

La comunicación estará compuesta por, un Arduino conectado con un Xbee que estará actuando como un coordinador, y los end-device que serán compuestos por un Xbee conectados de forma directa a los sensores (al MQ7, para el monóxido de carbono, y al AM2302, para la temperatura) mientras que las luces y demás alimentaciones que se controlen (como el de la alarma y el control de riego) estarán representados por un LED al que nosotros controlaremos,

encendiéndolos y apagándolos. La comunicación será Punto a Multipunto en modo API.

Una vez ya que decidimos el modo de funcionamiento procedemos a elegir a que pins del Xbee se conectaran los sensores/LEDs. En nuestro caso a modo de ejemplo para darnos una idea utilizamos estos Pins:

Xbee A

Pin 20: estará conectado un LED representando una luz de una casa.

Pin 19: estará conectado un LED representando la alarma.

Pin 18: estará conectado la pata digital del MQ7.

Pin17: estará conectado la pata analógica del MQ7.

Xbee B

Pin 20: estará conectado un LED representando una luz de una casa.

Pin 19: estará conectado un LED representando el control de riego.

Pin17: estará conectado la pata analógica del AM2302.

(Esto está sujeto a cambios y es tan solo para declarar los pins, y así poder formar las tramas necesarias para la comunicación del coordinador con el Xbee)

Una vez que dejamos en claro el hardware proseguimos a armar las tramas que enviara el coordinador y así formar una comunicación entre los end-device. En nuestro caso no tenemos ni un Xbee, ni mucho menos una picaro para conectarlos y configurarlos, por lo tanto, simulamos una creación de tramas. Lo hicimos con la aplicación XCTU, en la parte superior, en la sección “Tools” se pueden generar y leer los frames, generamos los frames dependiendo de que necesitamos. En nuestro caso utilizamos comando AT, mas específicamente el comando IS (para leer los estados de los end-device) y los comandos D0 para modificar las salidas. Ya obtenidos los frames procedemos a programar el coordinador. (En lo siguiente que comentaremos son “pruebas” que realizaremos, por el momento, con un solo Xbee, el Xbee A).

Declaramos el Xbee, algunas variables que luego utilizaremos para guardar las tramas, y por último, iniciamos la comunicación.

```
#include <SoftwareSerial.h>
SoftwareSerial Xbee(2,3);    //RX, TX

char VR;            //Tecla Pulsada
byte ISA[19];       //Estado del XbeeA
bool EstDigA;       //Estado Dig A
int EstAnalogA;     //Estado Analog A
byte ALED[20];      //Arreglo LED A
//-----------------------------------------------------
void setup() {
  Serial.begin(9600);   //Iniciamos la comunicacion
  Xbee.begin(9600);     //Iniciamos el Xbee
}
```

Consultamos si el Rasberry está disponible y en caso de que si, leemos el dato que nos envía para llamar a la función que encienda/apague el LED. Luego consulta al Xbee si está disponible y llama a la función para leer los estados de los pines, más tarde mostraremos las funciones. Por último, ubicamos un delay de 10 segundos, y aquí es donde nos presenta un problema, pues para encender el LED nos da un delay y las pruebas que realizamos con solo el Rasberry y Arduino enviar un dato análogo y que a su vez nos lea el dato que el Rasberry le envía al Arduino, nos da errores. Por lo tanto, esta sección probablemente este mal.

```
void loop() {
  if(Serial.available()){
    VR = Serial.read();
    switch(VR) {
      case 'A': ONALED;
      break;
      case 'a': OFFALED;
    }
  }
  if(Xbee.available()) {
    ISXbeeA();
  }
  delay(10000);
}
```

Iniciemos con la función para prender y apagar el LED, que son bastante similares. Creamos la trama y la guardamos en un arreglo ya antes declarado, para luego enviárselo. Esta funcion es llamada para prender un LED cada que recibe una “A”, en caso que reciba una “a”, lo apagara y para ello la única diferencia es que byte en vez de ser “35”, pasara a ser “34”. Y eso es todo.

```
94  void ONALED() {
95    ALED[0] = 0x7E;
96    ALED[1] = 0x00;
97    ALED[2] = 0x10;
98    ALED[3] = 0x17;
99    ALED[4] = 0x01;
100   ALED[5] = 0x00;
101   ALED[6] = 0x00;
102   ALED[7] = 0x00;
103   ALED[8] = 0x00;
104   ALED[9] = 0x00;
105   ALED[10] = 0x00;
106   ALED[11] = 0x00;
107   ALED[12] = 0x00;
108   ALED[13] = 0xCA;
109   ALED[14] = 0xCA;
110   ALED[15] = 0x02;
111   ALED[16] = 0x44;
112   ALED[17] = 0x30;
113   ALED[18] = 0x35;
114   ALED[19] = 0xE4;
115   if(Xbee.available()) {
116     long OnLED = ALED;
117     Xbee.write(OnLED);
118   }
```

Realizamos lo mismo, armamos la trama, la guardamos en un arreglo para que luego sea enviada.

```
30  void ISXbeeA() {
31    ISA[0] = 0x7E;
32    ISA[1] = 0x00;
33    ISA[2] = 0x0F;
34    ISA[3] = 0x17;
35    ISA[4] = 0x01;
36    ISA[5] = 0x00;
37    ISA[6] = 0x00;
38    ISA[7] = 0x00;
39    ISA[8] = 0x00;
40    ISA[9] = 0x00;
41    ISA[10] = 0x00;
42    ISA[11] = 0x00;
43    ISA[12] = 0x00;
44    ISA[13] = 0xCA;
45    ISA[14] = 0xCA;
46    ISA[15] = 0x02;
47    ISA[16] = 0x49;
48    ISA[17] = 0x53;
49    ISA[18] = 0xF1;
50
51    if(Xbee.available()){
52      long ISAX = ISA;
53      Xbee.write(ISAX);
54    }
```

Una vez enviada la trama leemos la trama que nos envía el Xbee (o sea la respuesta), guardamos los bytes que no necesitamos en descartar hasta llegar al byte que necesitamos, la máscara de los estados digitales. Leemos dicho byte y lo guardamos en un arreglo para luego compararlo con otro byte, en caso de que sea 1 o 0, enviara un String al Rasberry y con eso sería todo para leer los estados digitales.

```
55    if(Xbee.available()) {
56      for(int i = 0; i < 12; i++) {
57        byte descartar = Xbee.read();
58      }
59      byte EstDig = Xbee.read();
60      EstDigA = 0x01 && EstDig;
61      switch(EstDigA) {
62        case 0: Serial.write("Luz A Apagada");
63        break;
64        case 1: Serial.write("Luz A Encendida");
65        break;
66      }
67      EstDigA = 0x02 && EstDig;
68      switch(EstDigA) {
69        case 0: Serial.write("Alarma Apagada");
70        break;
71        case 1: Serial.write("Alarma Alimentada");
72        break;
73      }
74      EstDigA = 0x04 && EstDig;
75      switch(EstDigA) {
76        case 0: Serial.write("Alcohol Detectado");
77        break;
78        case 1: Serial.write("Sin Presencia de Alcohol");
79        break;
```

Lo siguiente seria leer los estados analógicos, en nuestro caso es solo uno y procedemos a realizar lo mismo que hicimos antes. Leemos todo y lo descartamos hasta llegar a la máscara de los estados analógicos, esta vez no es un byte, sino dos, y una vez que agarramos los bytes del analógico, los pasamos a Volts y luego se lo enviamos al Rasberry.

```
83    if(Xbee.available()) {
84      for(int i = 0; i < 13; i++) {
85        byte descartar = Xbee.read();
86      }
87      int AnalogRead1 = Xbee.read();
88      int AnalogRead2 = Xbee.read();
89      EstAnalogA = AnalogRead1 + (AnalogRead2 * 256);
90      Serial.write(EstAnalogA);
91    }
92  }
```

Todo lo que mencionamos y mostramos, compila, pero dudamos si es realmente la forma en la que se tiene que hacer, o si incluso no cumple con lo que nosotros necesitamos. No hemos hecho pruebas de esto, y aun así hay un error que debemos arreglar, en términos generales lo que necesitaríamos seria realizar multi-tasks en el arduino, que sea capaz de realizar dos acciones a la vez.

3. Programa Final

3.1 Aclaraciones

Toda la sección “2. Primeras Pruebas” fue en un contexto distinto al actual, era un contexto en donde cada uno tenía que trabajar de forma individual y sin poder juntarnos. En dicha situación no podíamos verificar los errores, y teníamos la fe de que en un futuro próximo podríamos juntar todas las herramientas y componentes para entregar el proyecto final de forma apropiada.

Actualmente el contexto es distinto, incluso mejor, pero todavía con precariedades que a lo largo de la sección “3. Proyecto Final” se iran presentando, aun así, daremos algunos de los cambios más grandes antes, los cuales son:

- Las funciones, pues actualmente nuestro proyecto es capaz de encender, apagar y mostrar el estado de un LED. El LED trabaja de forma digital asi que necesitábamos alguna función que trabaje de forma analógica, por ello se mantuvo el sensor de temperatura. Por lo tanto, es capaz de trabajar con un LED y un sensor de temperatura (LM35), todas las demás funciones se tuvieron que descartar.

- Las comunicaciones, nosotros pudimos conseguir algunos componentes tanto para las comunicaciones, como para las funciones, aun asi uno de los componentes, específicamente para las comunicaciones no pudimos conseguir fue el Xbee. Se tuvo que descartar el Xbee tanto en el planteamiento de las comunicaciones como en la ejecución y práctica; lo único que se trabajó alrededor del Xbee fue lo visto en la sección “2. Primeras Pruebas”.

Aclarado todo lo anterior presentaremos los programas, funciones y resultados de nuestro proyecto.

3.2 Etapas de la comunicación

Los dispositivos utilizados para nuestro proyecto fueron (las flechas indican el sentido de comunicación):

Dispositivo Android ⇆ Firebase ⇆ Rasberry ⇆ Arduino ⇆ Sensores/LEDS

Los sensores/LEDS se conectan de forma directa al Arduino, y el Arduino está conectado de forma directa a través de un cable USB al Rasberry. Lo demás se comunica de forma remota.

3.3 Funcionamiento de cada función del IOT

Como nuestro IOT tiene dos funciones detallare como actuaria cada etapa.

El LED se encuentra de forma directa conectado con el Arduino, este estará constantemente esperando recibir un dato específico para encender/apagar el LED (como una “A” para encenderlo). Dicho dato será enviado por parte del Rasberry, que se encontrara conectado de forma directa con un USB al Arduino, pero para que el Rasberry envié ese dato lo primero que debe de hacer es verificar en la nube (Firebase) que dato está guardado en el mismo. Y, por último, el Firebase guardara el dato que hayamos enviado a través nuestro celular de forma remota.

Es decir, que nosotros como usuarios enviaríamos una cadena de datos al Firebase, el Firebase guardaría el dato para que luego más tarde sea leído por el Rasberry, y dependiendo de que dato haya leído, el Rasberry le enviara una “A” o “a” al Arduino para que encienda o apague el LED, respectivamente.

El LM35 o sensor de temperatura trabaja de forma distinta al LED, no tan solo estamos trabajando con datos analógicos, sino que la función que tiene nuestra red de comunicación no es modificar el estado del dato, sino más bien mostrarlo. El sensor estará conectado de forma directa con el Arduino, este lo que hara es agarrar el dato cada cierto tiempo y enviarlo al Rasberry (que se encontrara conectado de forma directa con un USB), luego el Rasberry envía el dato al

Firebase donde se imprime para que por último el dispositivo Android agarre el dato y lo imprima en la pantalla del móvil.

3.4 Datos del Firebase

Algo a destacar es que cada dispositivo que utilizamos para la comunicación funciona de forma distinta, tiene distintos lenguajes y por lo tanto distintas reglas sobre que es apto para la sintaxis de cada acción que queramos realizar. Por ejemplo, un “switch” en un Arduino tan solo es capaz de funcionar por números o caracteres, mas no por string (cadena de caracteres), pero en un Rasberry si es posible. Entonces debíamos encontrar un punto en común para que todos se comuniquen de forma más o menos similar.

Nuestro “switch” en Arduino tan solo acepta caracteres, específicamente una “a” o “A”, entonces para que el Rasberry se pueda comunicar con el Arduino lo que hacíamos es que leía el dato del Firebase y luego ese dato lo “traduzco” de forma que el Arduino pueda interpretarlo. Pues algo importante a mencionar es que cuando hablamos de datos en el Firebase, básicamente estamos hablando de Strings, todo lo que se ve en el Firebase es texto, no es número, entonces la idea es agarrar esos textos y compararlos con lo que nosotros necesitamos.

3.5 Programación y Programas Finales

3.5.1 Arduino

La programación no ha cambiado demasiado comparándolo con la anterior, la principal diferencia que hay es el timer que pusimos para la temperatura, la idea es simular algún tipo de “multi-task”, donde el arduino sea capaz de leer que es lo que le envía el Rasberry, y al mismo tiempo enviar el dato de la temperatura sin tener que perjudicar la lectura/envió de datos.

El timer que nosotros programamos lo que permite es que este contando desde un inicio, sin que tenga que llegar a dicha parte del programa, y cada que completa un segundo llama a la función, “LeerLM35” que hace el cálculo de la temperatura y la envía al Rasberry. Actualmente estamos escribiendo esto debido a que funciona, pero en su momento fue uno de los obstáculos que tuvimos que resolver, pues cuando utilizamos el comando “delay” no tan solo

pausaba el envió de datos, sino también la lectura, y como ya mencionamos era algo que no queríamos, deseábamos algún tipo de “multi-task”.

```
unsigned long time;
unsigned long t = 0;
int TPT = 1000;
int LED = 7;
const int Sensor = A0;
//----------------------------------------
void setup() {
  Serial.begin(9600);
  pinMode(LED, 'OUTPUT);
}
//----------------------------------------
void loop() {
  if (Serial.available() > 0) {
    char VR = Serial.read();
    switch(VR) {
      case 'A': digitalWrite(LED, HIGH);
      break;
      case 'a': digitalWrite(LED, LOW);
      break;
    }
  }
  time = millis();
  if (time - t > TPT){
    t = time;
    LeerLM35();
  }
}
```

```
void LeerLM35() {
  int value = analogRead(Sensor);
  float millivolts = (value / 1023.0) * 5000;
  float celsius = millivolts / 10;
  Serial.println(celsius);
}
```

3.5.2 Rasberry

El siguiente desafío ahora era lograr meter todo lo anterior en un solo programa, tuvimos varios problemas, uno de esos era que trabajaran como multitareas y tuvimos que recurrir a fuentes para que nos ayuden, luego de varias pruebas y errores logramos que funcionara con la función **class** que nos permite mantener el estado en este caso el led y el sensor que tenían que enviar los datos a la vez.

Primero importamos todas las librerías que vamos a utilizar y establecemos la comunicación de la Rasberry y el Arduino

```
## CODIGO TEMP + LED Funcionando (Final, We did it)
import serial
import firebase_admin
from firebase_admin import credentials
from firebase_admin import db

arduino = serial.Serial('/dev/ttyACM0', 9600)

```

Luego establecemos la comunicación entre la Rasberry y el Firebase, poniendo los archivos json y el URL que te los da el mismo Firebase y generamos la estructura de cómo se va a visualizar en el Database

```
class IOT():
    def __init__(self):
        cred = credentials.Certificate('/home/pi/Desktop/IoT/cred2.json')
        firebase_admin.initialize_app(cred, {
            'databaseURL': 'https://electronica-703.firebaseio.com/'
        })

        self.refHome = db.reference("Casa")
        #self.Inicial()
        self.refTemp = self.refHome.child("Temperatura")
        self.refLuz  = self.refHome.child("Luz")
        self.Enviar()

    def Inicial(self):
        self.refHome.set({
            "Luz": True,
            "Temperatura": True
            })

```

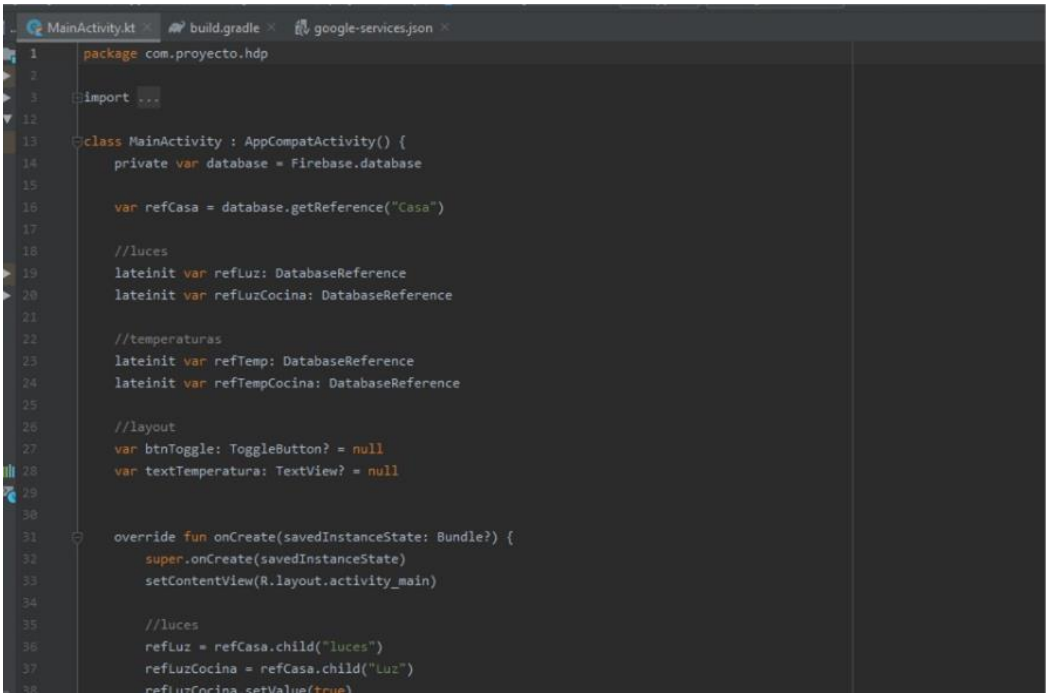
Por último, enviamos todos los datos que le envía el Arduino a la Rasberry y de ahí al Firebase.

```
    def Enviar(self):
        while True:
            leer = self.refLuz.get()
            print (leer)
            if leer == True:
                Envio = 'A'
            else:
                Envio = 'a'

            arduino.write (Envio.encode ())
            self.EnviarTemp()

    def EnviarTemp(self):
        temp = arduino.readline()
        temperatura = temp.decode('utf-8').strip()
        self.refTemp.set(temperatura)

print ("Start!")
iot = IOT()
```

3.5.3 Imagen del Firebase

Es una captura de nuestro RealTime Database de nuestro Firebase que actúa como la nube. Sirve para dar una mejor idea, y es aquí donde nosotros imprimimos los datos que necesitábamos, ya sea para leerlos (como el caso de la temperatura y el LED) o para cambiarlos (como el caso del LED). Y a esto es cuando nos referíamos de que el texto (o string) “false” del LED no nos servía para el Arduino, entonces es cuando el Rasberry debe de convertir ese false en una “a”, que significa apagar el LED.

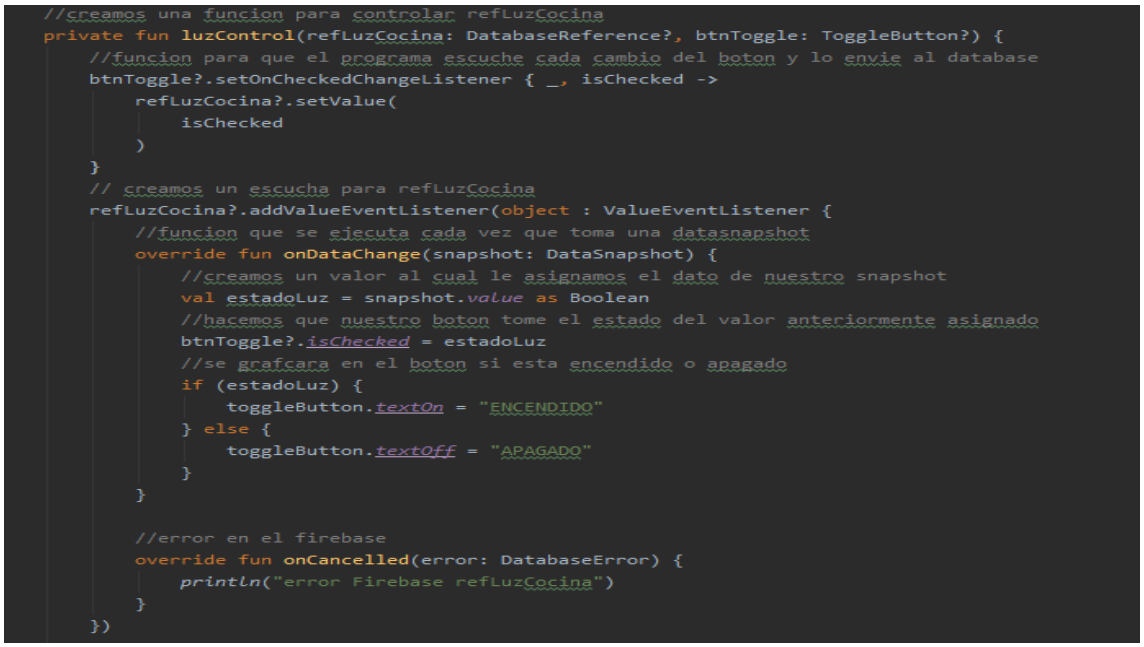
3.5.3 Android Studio



La programación no ha cambiado mucho desde la anterior presentada, en esta captura se muestran las importaciones de las librerías y la creación de las distintas variables que utilizaremos más adelante en el código.

```
    // refLuz = refCasa.child("luces")
    refLuz = refCasa.child( pathString: "Luz")
    refLuz.setValue(true)

    //temperaturas
    //refTemp = refCasa.child("Temperaturas")
    refTempCocina = refCasa.child( pathString: "Temperatura")
    refTempCocina.setValue("0")

    //layout
    btnToggle = findViewById<ToggleButton>(R.id.toggleButton)
    textTemperatura = findViewById(R.id.textTemperatura)

    luzControl(refLuz, btnToggle)
    tempControl(refTempCocina, textTemperatura)


}

private fun tempControl(refTempCocina: DatabaseReference, textTemperatura: TextView?) {
    refTempCocina.addValueEventListener(object : ValueEventListener {
        override fun onDataChange(snapshot: DataSnapshot) {
            val Temperatura = snapshot.value as String
            textTemperatura?.text = Temperatura
        }

        override fun onCancelled(error: DatabaseError) {}
    })

}
```

En esta imagen creamos las funciones principales del programa (Control de Temperatura y Control de Luz). También vemos que se define la función de temperatura, y lo que esta hace es recibir y enviar datos desde el Firebase y los grafique en pantalla.



```
//creamos una funcion para controlar refLuzCocina
private fun luzControl(refLuzCocina: DatabaseReference?, btnToggle: ToggleButton?) {
    //funcion para que el programa escuche cada cambio del boton y lo envie al database
    btnToggle?.setOnCheckedChangeListener { _, isChecked ->
        refLuzCocina?.setValue(
            isChecked
        )
    }
    // creamos un escucha para refLuzCocina
    refLuzCocina?.addValueEventListener(object : ValueEventListener {
        //funcion que se ejecuta cada vez que toma una datasnapshot
        override fun onDataChange(snapshot: DataSnapshot) {
            //creamos un valor al cual le asignamos el dato de nuestro snapshot
            val estadoLuz = snapshot.value as Boolean
            //hacemos que nuestro boton tome el estado del valor anteriormente asignado
            btnToggle?.isChecked = estadoLuz
            //se grafcara en el boton si esta encendido o apagado
            if (estadoLuz) {
                toggleButton.textOn = "ENCENDIDO"
            } else {
                toggleButton.textOff = "APAGADO"
            }
        }

        //error en el firebase
        override fun onCancelled(error: DatabaseError) {
            println("error Firebase refLuzCocina")
        }
    })
```

En esta captura lo que hacemos es crear un botón para prender y apagar la luz, este botón lo situamos dentro de la función (Control de Luz), en la cual hacemos lo mismo que en la de temperatura, y tomamos los datos que nos da Firebase y hacemos que dependiendo si el dato es False o True, el botón este encendido o apagado, y si esta variable cambia de valor, ese valor lo aviemos al Firebase.

3.6 Problemas Principales

Tuvimos nuestros errores, principalmente cuando intentábamos que todo sucediera a la vez teníamos muchos errores, un obstáculo que nos detuvo mucho fue que intentábamos prender el LED mientras que la temperatura se actualizaba, éramos capaces de hacerlos trabajar de forma separada pero cuando intentábamos unificar todas las cosas no salían muy bien, fue cuestión de prueba y error e ir probando incluso sintaxis o comandos no correspondientes a lo que queríamos pero que nos daban el resultado que necesitábamos. Una herramienta, o mejor dicho una página que funcionaban como algún tipo de blogs donde posteabas tu consulta o problemas y con suerte alguno te podía dar una mano, nos ayudó mucho cuando teníamos errores en la sintaxis que no éramos capaz de identificar. Creo que lo más negativo que podríamos decir sobre nuestro proyecto es que le falta pulir, y que tal vez actualmente nos funciona, pero habría que ver cuantas funciones seriamos capaz de implementar hasta que deje de funcionar.

3.7 Resultados y Conclusiones

Creemos que los resultados fueron más que positivos, todo lo mostrado funciona a la perfección. La situación fue muy precaria, y si no fuera porque la situación fue mejorando con el tiempo tal vez no hubiéramos podido lograr buenos resultados, juntarnos con las herramientas que disponíamos y poder ser capaces de probar y corregir los errores que encontrábamos fue de muchísima ayuda. Es verdad que tuvimos que bajar nuestra expectativa y descartar funciones, pero viéndolo en el contexto… nos sentimos contentos con el resultado. En todo caso tal vez hubiéramos deseado tener todas las herramientas y hacer un proyecto final mucho más completo y gratificante.