UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

TRABAJO FINAL DE CARRERA

Creación de sistema domótico para Mascotas mediante Android Studio y Arduino

GRADO EN INGENIERÍA ELÉCTRICA

Autor: Jhonatan Muñoz Carrasco

Director: Antoni Pérez Poch

Departamento: Ciencias de la Computación

Resum

Aquest treball de final de grau tracta sobre la creació de una aplicación amb finalitat de controlar una unitat de domòtica.

Aquest sistema domòtic es creat mitjançant una placa de Arduino, aquesta placa controlara diferents elements de una sala per cuidar i mantener mascotas. En aquest cas l'aplicació controlarà el sistema de llums de la sala, la climatització depenent dels paràmetres que hi hagin a la sala, aquests paràmetres estarán en controlats en tot moment mitjançant el sensor de Temperatura i humitat, que anirà informant el usuari l'estat de la sala. La pantalla del calendari es una mica mes simple ja que per registrar la informació d'alguna acció, es guarda directament en el propi sistema operatiu sense fer servir cap plataforma externa per guarda-la. La pantalla de dispensador de menjar i aigua te dos pantalles separades en la de l'aigua tindrà un sensor ultra Sonic per controlar la quantitat d'aigua que hi ha en el recipient. En la del menjar estarà formada per un LED amb un sensor LDR que simularà el sensor de pes si hi ha menjar el LED estarà apagat, si el sensor esta sense tapar vol dir que no hi ha menjar llavors l'usuari podrà encendre el LED simulant la obertura o funcionament del motor per on passa el menjar una vegada arribi a estar al nivell desitjat de ple en aquest cas de que la LDR estigui tapada l'usuari podrà apagar el LED. Indican que ja este ple el recipient de menjar.

Resumen

Este trabajo de final de grado trata sobre la creación de una aplicación con finalidad de controlar una unidad de domótica.

Este sistema domótico es creado mediante una placa de Arduino, dicha placa controlara diferentes elementos de una sala para cuidar y mantener las mascotas. En este proyecto la aplicación controlará un sistema de iluminación, de climatización, dispensador de comida y tendrá un calendario para guardar posibles eventos. El usuario podrá ver en todo momento la temperatura y la humedad de la sala, si el usuario desea podrá acceder al sistema de encendido y apagado de la ventilación o calefacción, en la pantalla de iluminación solo hay un botón que permitirá encender y apagar. En el sistema de dispensador hay dos botones uno para el agua que si el usuario pica en el botón accederá a una pantalla donde el usuario indica la profundidad en este caso del recipiente que contiene el agua y lo verifica aparecerá en la pantalla la cantidad de agua que queda en dicho recipiente. En el dispensador de comida se hará mediante un sistema de LDR para simular un sensor de peso y un led indicando que el usuario a encendido el motor y lo para una vez llega a lo deseado.

Abstract

The final degree Project deals with the crationj of an application in order to control a home automation unit.

This home automation system is created using an Arduino board, said board will controldifferent elements of a room to care for and maintain pets. In this project, the application will control a lighting system, air conditioning, food dispenser and will have a calendar to sabe posible events. The user will be able to see at all times the temperatura and humidity of the room, if the user wishes, they will be able to acces the ventilation or heating system, on the lighting screen there is only one button that will allow turning on and off. In the dispenser system there are two buttons, one for wates, if the user clicks on the button, he will acces a screen where the user indicates the depth in this case of the containesr that contains the wáter and verifies it, the amount of wáter will appear on the screen the remaining wáter. In the food screen, it will be done by a LDR system to simulate a weight sensor and a LED indicating that the user has started the engine and stops it once reaches what is desired.

Agradecimientos

Gracias al profesor que imparte la asignatura optativa de Programación de dispositivos móviles, pude realizar este proyecto, ya que con el pude descubrir que me gusta la programación y que puede llegar a ser muy entretenida, también le agradezco a mis padres por darme idea para poder desarrollar alguna aplicación.

Índice

1. Prefacio

1.1. Origen del trabajo

El origen de este proyecto viene dado por buscar una app que permita domotizar la estancia para mascotas. La idea surgió cuando hablaba con mi madre sobre que se podría hacer el trabajo y me propuso diferentes ideas. Me decante por este tipo de trabajo porque me gusta la optativa de programación de móviles que curse el cuatrimestre pasado.

1.2. Motivación

Mi motivación viene dada por el hecho de que tengo una mascota en casa y quería buscar un sistema que me facilitara el cuidado de mi mascota en casa sin tener que estar pendiente de ciertas funciones, como alimentarlo o ponerle agua las demás funciones son un plus, si yo no me encuentro en casa hacer un poco más confortable la espera de mi mascota dependiendo de la temporada ambiental a la que estemos

1.3. Requerimientos previos

Para este proyecto se ha hecho falta tener un conocimiento en programación mínimo para poder llevar a cabo el trabajo. En mi caso la idea de crear esta aplicación me llevo a tener un reto ya que mi conocimiento era lo que había aprendido en la universidad, pero gracias a cómo nos enseñaron a programar me veía con ganas de intentarlo por mi cuenta y con la guía del tutor.

2. Introducción

Hoy en día la gente busca comodidad y rapidez a la hora de hacer las cosas, estas facilidades vienen dada por el uso de aplicaciones como, por ejemplo; pagar con el móvil, controlar la climatización cuando sales, la luz en caso de que quieras simular que hay alguien en casa cuando no lo hay para evitar entrada no deseada o simplemente para no tener que levantarse cada vez que se quiera encender o apagar una luz, etc….

En mi caso quiero crear una aplicación que pueda controlar un sistema domótico para una sala con mascotas. El sistema estaría formado por una placa de Arduino como central de control sobre los diferentes sistemas de acción y la aplicación móvil como central de control sobre el Arduino en este caso.

La aplicación será creada mediante softwares gratuitos que hay en la red como, AppInventor o AndroidStudio. Usare uno de estos dos programas para crear y diseñar mi aplicación.

2.1. Objetivos del trabajo

Mi objetivo es crear una aplicación funcional que controle diferentes situaciones de una estancia, y probar dicha aplicación en una placa de Arduino como placa de control. Las pantallas que se programara serán las siguientes, dispensador de agua y comida, control de luz, climatización y control de temperatura y humedad y por ultimo una pantalla con calendario donde el usuario puede registrar eventos.

2.2. Alcance del trabajo

La idea del proyecto es crear y validar una aplicación real que controle un sistema domótico diseñado para cualquier tipo de sala que tenga mascotas.

El primer paso fue buscar información de aplicaciones similares a la que quiero desarrollar, pero no vi una parecida a la que quiero llevar a cabo sino partes de diferentes aplicaciones. Entonces busque la forma de unificarlas en una sola aplicación y poder controlarlas con ese módulo de Arduino.

El segundo paso fue pensar cómo se desarrollaría la aplicación y en qué orden estaría programada cada función.

En el tercer paso sería la programación de cada pantalla y por ultimo crear el sistema domótico que registre y envié datos a Firebase que luego la aplicación recibirá.

3. Estado del Arte

Encontré proyectos donde el usuario controla dispensadores de agua y comida, los productos que vi solo tienen una aplicación únicamente para ese fin, mi idea es integrar esa idea en una aplicación con diferentes funciones de una estancia. Lo mismo pasa para el sistema de control de temperatura y humedad de la estancia, y la luz lo mismo el calendario está dentro del dispositivo móvil por defecto. Pero la idea de crear esta aplicación es que facilitaría al usuario controlar con solo una aplicación diferentes acciones de una estancia sin tener que abrir diferentes aplicaciones para llevarlo a cabo.

4. Metodología

Hoy en día a la hora de programar nos encontramos con que java ya no es el único lenguaje que se usa para programar aplicaciones, también nos podemos encontrar con Mit App 2, Kotlin+jetbrains y Android Studio con java, XML y Kotlin.

El Mit App 2 es una herramienta mejorada de programación creada por el (Instituto Tecnológico de Massachusetts) este programa facilita a los usuarios programar de forma sencilla aplicaciones para dispositivos Android este programa fue adoptado por Google. Este servicio lo puede usar usuarios que ni sepan programar ya que el propio Mit te facilita dicha programación ya que es muy visual.

Kotlin y jetbrains el lenguaje de programación de Kotlin es estático y de código abierto que admite la programación funcional y que se le puede aplicar a objetos. Los conceptos y sintaxis son parecida al lenguaje C++, Java y Scala entre otros. Jetbrains es una compañía que tiene muchos lenguajes integrados también conocido como (IDE). El sistema de Kotlin permite desarrollar aplicaciones en Android, es un lenguaje moderno, haciendo que el usuario escriba menos a la hora de programar y trabajar con él.

Cogí el Sistema de Android Studio con lenguaje Kotlin porque lo vi más fácil a la hora de programar, ya que esta todo de forma más reducido, el problema llega cuando buscas algún código en java y lo pasas a Kotlin el propio sistema de Android Studio te facilita ese cambio de código si no sabes cómo escribirlo en Kotlin.

En el mercado podemos encontrar diferentes tipos de circuitos impresos con componentes discretos: Raspberry Pi y Arduino.

Raspberry Pi es una placa de ordenador simple compuesto por un SoC, CPU, memoria RAM, puertos de entrada y salida de Audio y video, conectividad de red, ranura SD para más almacenamiento, reloj, una toma de alimentación, conexiones para periféricos de bajo nivel, para programar la Raspberry Pi tiene diferentes sistemas operativos, pero el más usado es el Raspbian que usa un sistema operativo libre basado en Linux.

Arduino es una placa de ordenador simple también pero no lleva incorporados puertos de entrada y salida de audio y video, como la Raspberry Pi, Ambas placas tienen diferentes modelos dependiendo del uso que se les quiera dar. En este proyecto se hará uso de Arduino ya que anteriormente la he usado y más o menos sabia como programarla y subir librerías en caso de que haga falta.

5. Ingeniería de concepción

El proyecto está divido en dos partes una de programación y diseño de una aplicación móvil, y una segunda parte que consiste crear un sistema de control mediante una placa de Arduino que sea capaz de registrar y enviar los datos a nuestra aplicación y a la misma vez que sea capaz de recibir la señal que el usuario le envié a la aplicación.

5.1. Programación de Arduino y Móvil

En este apartado se organizará de cómo estará programado y en qué orden cada pantalla y sensor o actuador usado en la aplicación que vamos a simular. El lenguaje usado en las pantallas es el que se ha nombrado anteriormente mediante lenguaje Kotlin. En cambio, la placa de

Arduino se programó mediante lenguaje Java. Los códigos se verán en el apartado siguiente ya que cada pantalla tiene su propio código y funcionalidad. Lo mismo pasa con el Arduino cada sensor se programa de una forma, y la forma en que registra los datos y los recibe igual.

5.2. Conexión a la placa de Arduino

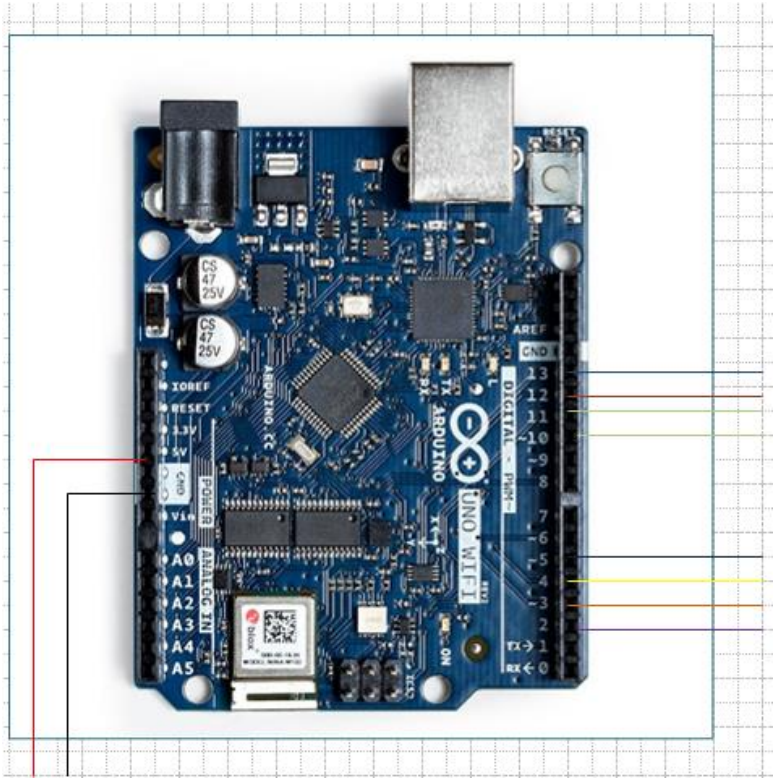


Imagen 1 Conexión elementos en los pin del Arduino

EL cable de color rojo que podemos observar en la parte izquierda de la placa es el cable de alimentación que está conectado a la tensión de salida de 5 V. El cable negro que esta abajo del rojo es el cable de la tierra necesario para algunos elementos que tienen más pin de conexión.

El pin numero 13 está conectado como pin de salida a nuestro actuador LED de dos colores, que simulara el sistema de iluminación de la sala.

El pin numero 12 está conectado como pin de salida del LED RGB, que en este caso el pin 12 permitirá encender la luz roja del LED. El pin 11 permitirá del mismo LED RGB, permitirá encender la luz azul del LED, la luz roja simulará el sistema de calefacción y la luz azul el sistema de ventilación.

El pin numero 10 tendremos otro LED RGB, pero tipo SMD (dispositivo de montaje superficial), este led estará conectado a la luz verde, este led permitirá simular el encendido de la válvula de apertura y cerrado del dispensador de comida.

En el pin numero 5 está conectado al sensor ultrasónico, en el pin de ECHO permitiendo la salida de la señal recibida, en el pin numero 4 tendremos conectado el pin Triq (triangulación)

Es el pin de entrada de señal que permite recibir la señal enviada por el pin echo una vez la señal rebota en alguna superficie indicándonos la profundidad de dicho recipiente si está vacío o lleno. Este sensor permitirá simular la cantidad de agua que tendrá el recipiente en todo momento.

En el pin numero 3 conectamos el sensor de fotorresistencia este sensor simulara la cantidad de comida que queda en el recipiente de comida haciendo como si fuera el sensor de peso.

En el pin numero 2 conectamos el sensor de temperatura y humedad encargado de registrar esos parámetros de la sala y mantener al usuario informado en todo momento.

6. Ingeniería de detalle

6.1. Controlador

Nos permitirá recoger y almacenar la información recogida por los sensores, y enviar las órdenes recibidas por la aplicación móvil a los actuadores. Hay ciertos actuadores que no requieren una unidad de control, pero en este caso los actuadores que se usaran si requieren de una placa de control que nos permitirá procesar la información que reciben según lo que el usuario desea controlar.

El tipo de control que se usara es una placa de circuitos integrados, que permitirá conectarse a la red Wifi y al ordenador. Existen diferentes placas integradas: Arduino y Raspberry Pi.

6.1.1. Placa de Arduino

La placa de Arduino es del tipo de controlador de una sola placa, como se podrá observar. La programación de esta placa es abierta y fácil.

Hay varios modelos de Arduino (Arduino uno, uno rev2, nano, Yun, etc) dependiendo de la necesidad del proyecto se usará una u otra. En este proyecto se hará uso de la placa Arduino uno rev2.

Esta placa lleva integrada red Wifi y permite la conexión con aplicaciones móviles.

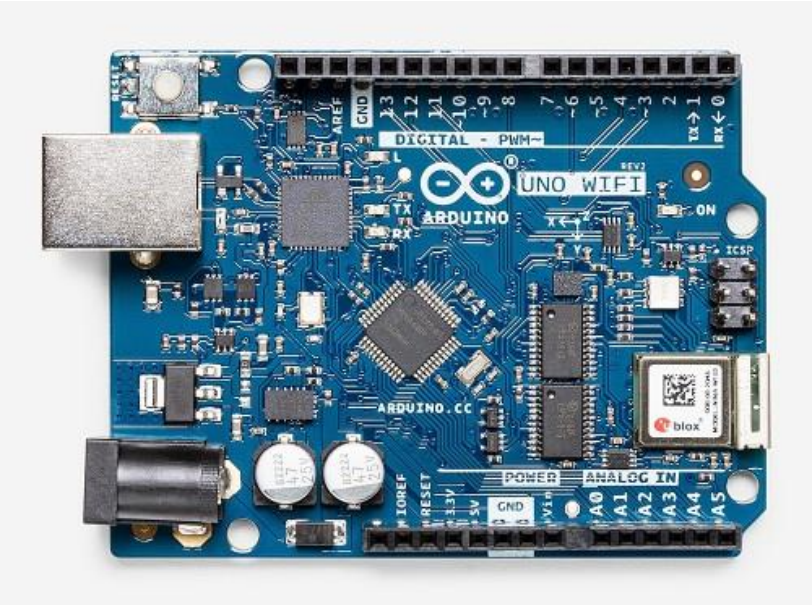


Imagen 2 Placa Arduino Uno rev2. Fuente: https://store.arduino.cc/arduino-uno-wifi-rev2

Características de la Placa

Microcontrolador	ATmega4809
Voltaje de operación	5V
Voltaje de entrada	7- 12 V
Pines digitales I/O	14 – 5 PWM[1] Output
Pines PWM[1] Digitales I/O	5
Pines analógicos entradas	6
Corriente por I/O Pin	20 mA en pines I/O; 50 mA en el pin 3,3V
Memoria Flash CPU	48 KB
Memoria SRAM	6,144 Bytes
Memoria EEPROM	256 Bytes
Velocidad de reloj	16 MHz
Comunicación Wireless	u- blox NINA- W102
Elemento de seguridad	LSM6DS3TR
Sensor de inercia	LSM6DS3TR
LED_BUILTIN	25
Longitud	68.6 mm
Ancho	53,4 mm
Peso	25 g

Tabla 1 Característica de la placa de Arduino. Fuente:https://stroe.arduino.cc/arduino-uno-wifi-rev2

6.1.2. Componentes extras para la placa

Para poder poner la placa en funcionamiento se requieren de los siguientes elementos:

- Fuente de alimentación de 7 a 12 V
- Cable micro-USB
- La propia placa puede guardar el sistema operativo, pero si se requiere o le falta más capacidad se le podrá añadir una micro-SD, si la programación va por micro-SD no hace falta conectar la placa al ordenador.
- La conexión a internet será por Wifi.

6.1.3. Manipulación de la placa de Arduino

El microprocesador que lleva el Arduino no se puede cambiar si se hace un mal uso de el mismo, durante la programación. Hay que ser cuidadosos cuando se vaya a trabajar con el dispositivo,

- Manipular correctamente la tensión en los pines del Arduino para evitar sobre voltajes.
- Conectar el voltaje correctamente en el pin de entrada, para evitar sobre voltaje.
- Seguir las recomendaciones del fabricante a la hora de conectar actuadores y sensores en el Arduino para evitar sobre corrientes.
- Nunca conectar directamente el Arduino sobre: motores para alimentarlos, cargas inductivas (tipo relay), ni superar la cantidad de leds que se pueden alimentar.

6.1.4. Software del Arduino

El software usado en Arduino es un IDE también conocido como Integrated Development Environment. Es un programa informático compuesto por un conjunto de programas de acceso libre y gratuito.

6.2. Software de almacenamiento de datos

Se hará uso de la plataforma de Firebase de Google ya que es una plataforma gratuita de almacenamiento de datos e información en tiempo real. Es gratuita hasta cierta capacidad de

información que se puede guardar y descargar y del número de usuarios que se registren en la aplicación.

Siguiendo los siguientes pasos haremos la integración de la base de Firebase en nuestra aplicación y nuestra placa de Arduino.

- Paso 1

  Registraremos nuestra aplicación según nos marca Firebase.

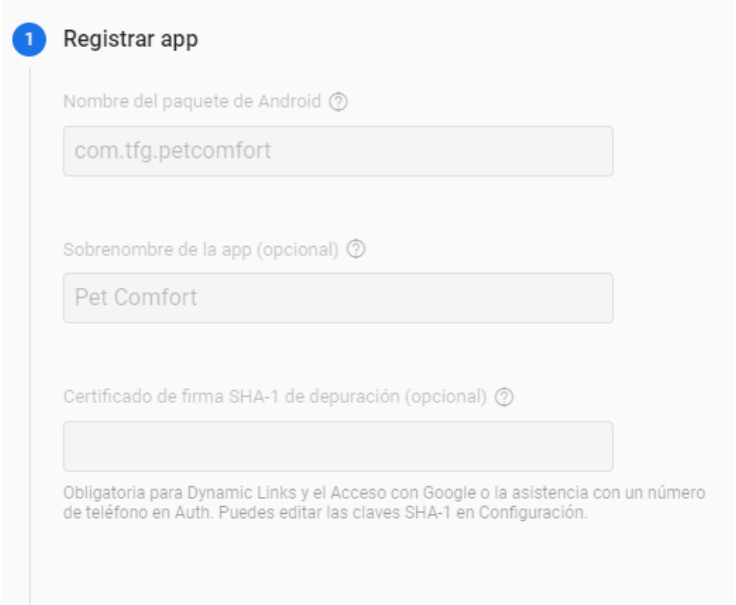


Imagen 3 Registro App. Fuente de Firebase

- Paso 2:

  Descargaremos el fichero que nos proporciona Firebase, que luego integraremos en nuestra aplicación tal y como nos indica Firebase.

Imagen 4 Descarga fichero google-services.json. Fuente de Firebase

- Paso 3:

  Agregaremos los códigos proporcionados por Firebase si la aplicación no los tiene integrados. Seguiremos las instrucciones que nos proporciona Firebase para saber dónde y cómo integrarlos sin que de posibles errores a la hora de compilar el proyecto.

Imagen 5 Código a integrar en la App. Fuente de Firebase

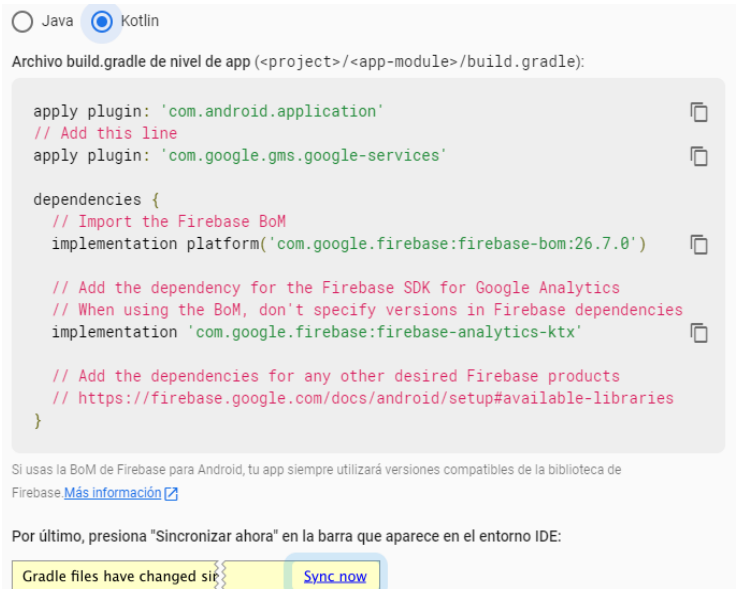


Imagen 6 Código a integrar en la App. Fuente de Firebase

- Paso 4:

  Revisar que estén todos los pasos anteriores bien integrados, y estará todo listo. Sincronizaremos los nuevos códigos en la App para que sean funcionales a la hora de programar la App.

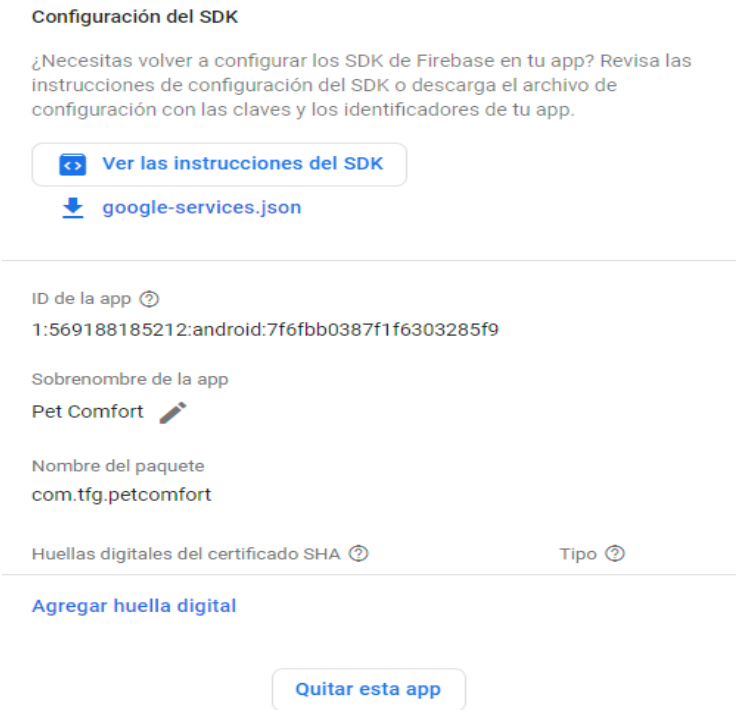


Imagen 7 Verificación de registro de la App. Fuente de Firebase

7. Recepción y control mediante App

Una vez este configurado el Arduino para hacer de control sobre los actuadores necesitaremos un control externo como en este caso será una aplicación, para poder observar y controlar lo que los sensores detectan y esa información es procesada y enviada por el Arduino a la base de datos de Firebase donde posteriormente se envía a nuestra aplicación.

7.1. Pantalla de Registro

En la búsqueda de información se ha podido encontrar el código de cómo programar una Login Activity.

Siguiendo los pasos cree la pantalla de registro, primero se creará la interfaz gráfica de la Activity. Una vez finalizado el diseño gráfico pasaremos a la parte de programación de esa pantalla para hacerla funcional, ya que en esa pantalla los elementos con botón no hacen ninguna acción.

7.1.1. Diseño gráfico

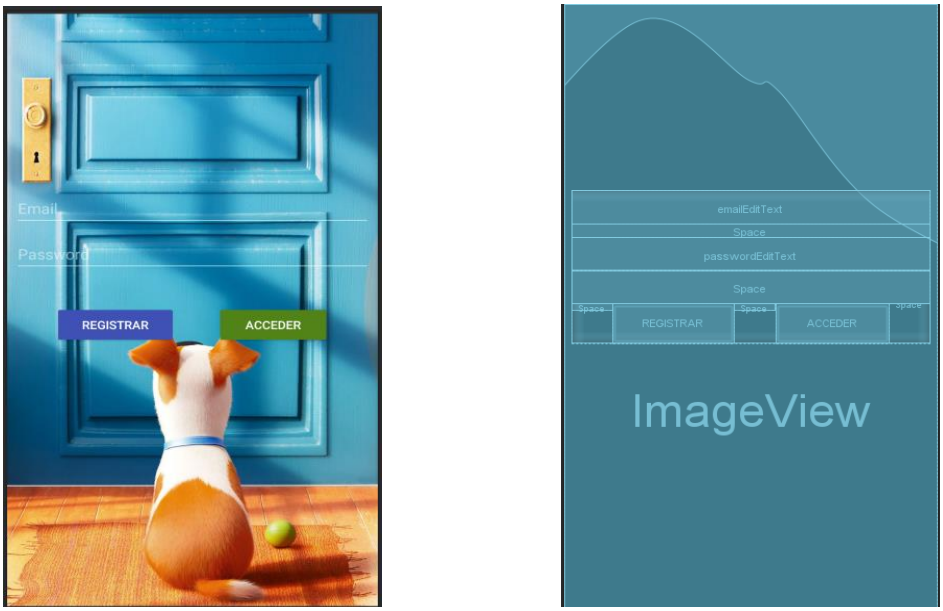


Imagen 8 Diseño gráfico de la pantalla de Registro. Fuente del Fondo de la pantalla en google.

7.1.2. Programación de la pantalla

En la parte de programación permitirá que los elementos que hay en la pantalla gráfica sea funcional y operativa para que cuando el usuario haga uso de ello. Haciendo que los elementos como los dos botones y los dos componentes donde irán el registro de correo y contraseña. Junto en esta pantalla estará enlazada a la base de datos de Firebase para poder registrar y guardar el usuario que se registre.

```
class LoginActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_login)

        // Setup

        setup()
    }

    private fun setup(){
        title= "Login Screen"

        registerButton.setOnClickListener {  it: View!
            if( emailEditText.text.isNotEmpty() && passwordEditText.text.isNotEmpty()){

                FirebaseAuth.getInstance().createUserWithEmailAndPassword(emailEditText.text.toString(),
                    passwordEditText.text.toString()).addOnCompleteListener {  it: Task<AuthResult!>

                        if( it.isSuccessful){
                           showMenu( email: it.result?.user?.email?:"")
                        } else {
                            showAlert()

                        }
                }
            }
        }
    }
```

Imagen 9 Programa aplicado al botón de registro

En la imagen 8 se puede observar que dentro de la clase LoginActivity se ha agregado un setup que permitirá ejecutar los comandos siguientes relacionados a nuestra pantalla gráfica.

En la imagen8 se puede observar que se ha creado una función para el comando setup, dentro de esa función se configurara los botones que hay en la interfaz gráfica y también los espacios de registro de correo y contraseña. Los parámetros para el botón de registro una vez configurado llamara una serie de funciones que permitirán al usuario registrarse, siempre y cuando cumpla con los requisitos, ya que si el usuario no rellena ningún campo o no pone un correo correctamente o no pone contraseña el usuario no podrá registrarse dando una notificación de fallo al registrarse. Si el usuario una vez se ha registrado correctamente cuando pulse al botón de acceder se le permitirá la entrada al menú de la aplicación, pero si el usuario no pone correctamente su usuario o contraseña saltara un error de que no puede acceder debido a algún error en la contraseño o en el email. Como se puede observar en la imagen 9.

```
    loginButton.setOnClickListener { it: View!
        if( emailEditText.text.isNotEmpty() && passwordEditText.text.isNotEmpty()){

            FirebaseAuth.getInstance().signInWithEmailAndPassword(emailEditText.text.toString(),
                passwordEditText.text.toString()).addOnCompleteListener { it: Task<AuthResult!>

                if( it.isSuccessful){
                    showMenu( email: it.result?.user?.email?:"")
                } else {
                    showAlert1()

                }
            }
        }
    }

}
```

Imagen 10 Programa aplicado al botón de Acceder

```
private fun showAlert(){

    val builder = AlertDialog.Builder( context: this)
    builder.setTitle("Error")
    builder.setMessage("Usuario ya registrado")
    builder.setPositiveButton( text: "Aceptar", listener: null)
    val dialog: AlertDialog = builder.create()
    dialog.show()
}

private fun showMenu(email: String) {

    val menuIntent = Intent( packageContext: this, MenuActivity::class.java).apply { this: Intent
    }

    startActivity(menuIntent)
}
private fun showAlert1(){

    val builder = AlertDialog.Builder( context: this)
    builder.setTitle("Error")
    builder.setMessage("Usuario no registrado o clave incorrecta")
    builder.setPositiveButton( text: "Aceptar", listener: null)
    val dialog: AlertDialog = builder.create()
    dialog.show()
}
```

Imagen 11 Programa aplicado a las alertas

En la imagen 10 se puede observar el programa de las alertas en caso de que el usuario no haya rellenado su registro. En esta imagen también está la función que permite mediante un Intent abrir otra pantalla o Activity como se le conoce en Android Studio.

7.2. Pantalla de menú

Para la pantalla del menú no se ha requerido de mucha programación compleja ya que es una pantalla de selección de diferentes acciones. Consiste en un conjunto de botones en formato imagen y un solo botón de cerrar sesión.

7.2.1. Diseño gráfico de la pantalla de menú

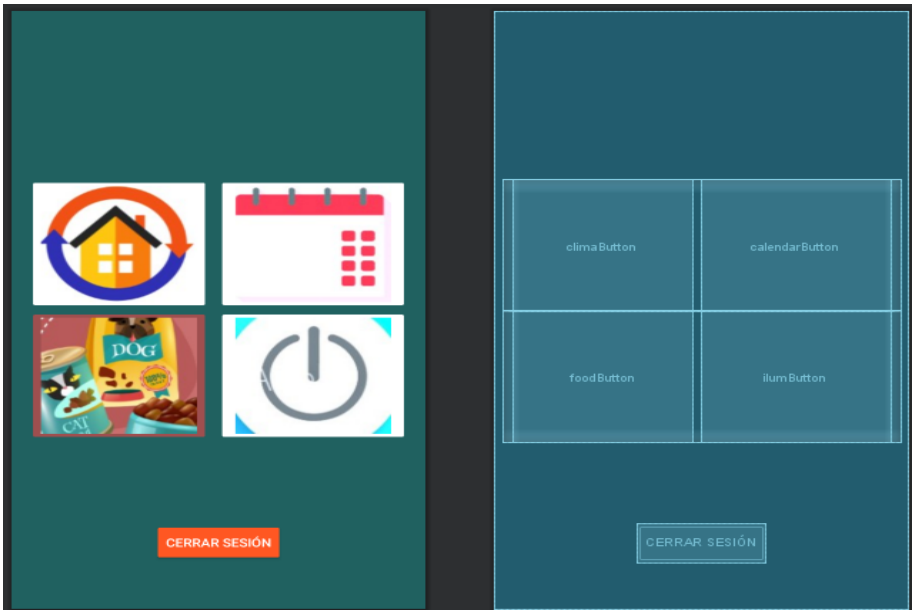


Imagen 12 Diseño gráfico de la interfaz de menú. Imágenes sacadas de Freepick

7.2.2. Programa de la pantalla del menú

```
class MenuActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_menu)

        // setup
        setup()
    }
    private fun setup(){

        logOutButton.setOnClickListener { it: View!
            FirebaseAuth.getInstance().signOut()
            onBackPressed()
        }
        calendarButton.setOnClickListener { it: View!
            calendarshow()
        }
        foodButton.setOnClickListener { it: View!
            foodshow()
        }
        climaButton.setOnClickListener { it: View!
            climashow()
        }
        ilumButton.setOnClickListener { it: View!
            ilumishow()
        }
    }
```

Imagen 13 Programa para activar los botones

En la imagen 12 se ha escrito las funciones que permiten hacer funcionales los botones de selección de nuevas pantallas. Cuando el usuario pulse uno de esos botones permitirá el acceso a nuevas pantallas mediante un intent.

```
    private fun calendarshow(){
        val calendarIntent = Intent( packageContext: this,calendarActivity::class.java).apply { this: Intent

        }
        startActivity(calendarIntent)
    }

    private fun foodshow(){
        val foodIntent = Intent( packageContext: this,foodActivity::class.java).apply { this: Intent

        }
        startActivity(foodIntent)
    }
    private fun climashow(){
        val climaIntent = Intent( packageContext: this,climaActivity::class.java).apply { this: Intent

        }
        startActivity(climaIntent)
    }

    private fun ilumishow(){
        val ilumiIntent = Intent( packageContext: this,ilumActivity::class.java).apply { this: Intent

        }
        startActivity(ilumiIntent)
    }
}
```

Imagen 14 Funciones Intent

7.3. Pantalla de temperatura y humedad

En esta pantalla se diseñó mediante una pantalla de control que permite controlar diferentes fragment que permite pasar de una fragment a otra sin tener que pulsar ningún botón solo con arrastrar la pantalla hacia un lateral.

7.3.1. Pantalla de clima

```
class climaActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_clima)
        val sectionsPagerAdapter = SectionsPagerAdapter( context: this, supportFragmentManager)
        val viewPager: ViewPager = findViewById(R.id.view_pager)
        viewPager.adapter = sectionsPagerAdapter

    }
}
```

Imagen 15 Programa pantalla control de los Fragment

Esta parte de programación no tiene diseño gráfico como tal ya que es una pantalla que permite controlar diferentes fragment dentro de esta. Mediante la función de sectionPagerAdapter que esta función está configurada en una clase aparte como se puede ver en la imagen 15.

```
class SectionsPagerAdapter(private val context: Context, fm: FragmentManager)
    : FragmentPagerAdapter(fm) {

    override fun getItem(position: Int): Fragment {
        return when (position) {
            0 -> TemperatureFragment()
            1 -> HumidityFragment()

            else -> throw IllegalStateException("Unexpected position $position")
        }
    }

    override fun getCount(): Int {
        return 2
    }

}
```

Imagen 16 Programa de la pantalla que permite el cambio entre fragments

En esta configuración se establece los parámetros que permiten que dentro de la pantalla clima que es la principal tenga dos fragmentos independientes entre sí como se verá en las imágenes siguientes en esta clase no tiene interfaz gráfica.

7.3.2. Fragmento de la pantalla Clima: Temperatura

En este fragment está la parte gráfica de nuestra pantalla que nos mostrara la temperatura y su valor a medida que pasa el tiempo. En esta interfaz tiene dos botones que permitirán al usuario entrar en la pantalla que controlara si se quiere encender o apagar el sistema de calefacción o ventilación si se requiere. Si el usuario no quiere saber cómo está la temperatura puede cancelar el flujo de datos en la temperatura mediante el botón que hay al lado del valor de la temperatura.

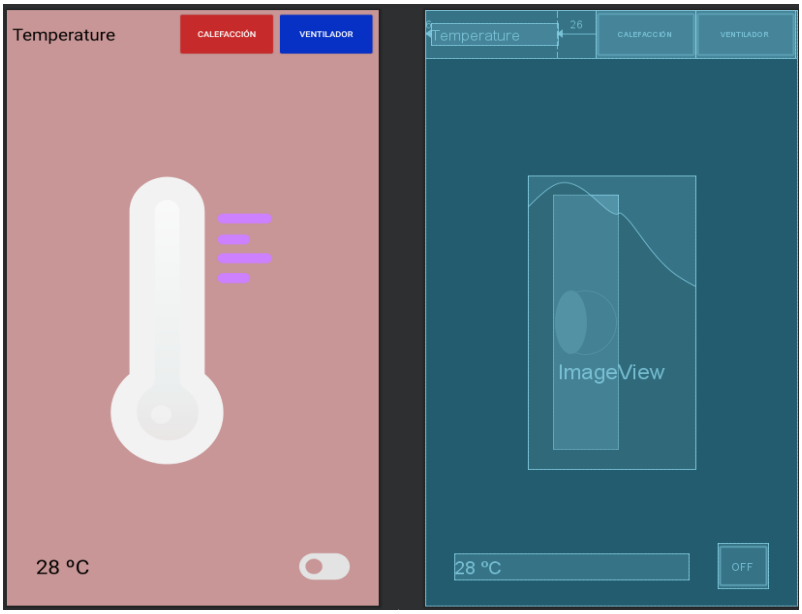


Imagen 17 Interfaz gráfica del fragment Temperatura

```
class TemperatureFragment : Fragment(){
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)


    }


    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        return  inflater.inflate(R.layout.fragment_temperature, container,  attachToRoot: false)


    }
    override fun onActivityCreated(savedInstanceState: Bundle?) {
        super.onActivityCreated(savedInstanceState)


        val database = Firebase.database
        val myRef = database.getReference()
        var dht11 = 0

        myRef.addValueEventListener(object : ValueEventListener {
            @SuppressLint( ...value: "SetTextI18n")
            override fun onDataChange(dataSnapshot: DataSnapshot) {
                val value: String = dataSnapshot.child( path: "temperature").getValue().toString()
                dht11  = dataSnapshot.child( path: "state").getValue().toString().toInt()

                temperatureProgressBar.progress = value.toFloat().toInt()
                temperatureTextView.text = value + " ºC"

                if (dht11 == 0){
                    dht11ToggleButton.isChecked = false
                    dht11 = 1024
                }else{
                    dht11ToggleButton.isChecked = true
                    dht11 = 0
                }
```

Imagen 18 Programación de la fragment Temperatura

En esta imagen 17 se puede observar cómo se describe los parámetros que se usaran para poder recibir la información que capta el sensor que luego es enviado al Arduino que se encargar de subir esos datos a la base de datos de Firebase. Una vez esos datos están en la base este programa se encargará de buscar en que sitio están guardados esos datos y los descargara para mostrarlo en la pantalla gráfica. Si no hubiera nada en la base de datos de Firebase saltara un mensaje de error de que no existe ningún valor mediante la función onCancelled de la imagen 18 se puede observar. También en esta parte del fragment se configurarán dos botones que permitirán al usuario acceder independientemente a la pantalla donde estará ubicado los botones de puesta en marcha de los dispositivos de calefacción y ventilación.

```
            override fun onCancelled(error: DatabaseError) {
                Log.w( tag: "TAG",  msg: "Failed to read value.", error.toException())
            }

        })

        dht11ToggleButton.setOnCheckedChangeListener { buttonView, isChecked ->
            if (isChecked) {
                myRef.child( pathString: "state").setValue(dht11)
            } else {
                myRef.child( pathString: "state").setValue(0)
            }
        }

        caleButton.setOnClickListener(View.OnClickListener {  it: View!
            var cale = Intent(activity,caleActivity::class.java)
            activity!!.startActivity(cale)
        })
        refriButton.setOnClickListener (View.OnClickListener {  it: View!
            var refri = Intent(activity,refriActivity::class.java)
            activity!!.startActivity(refri)
        })
    }

}
```

Imagen 19 Programación del fragment Temperatura

7.3.3. Control sistema Ventilación

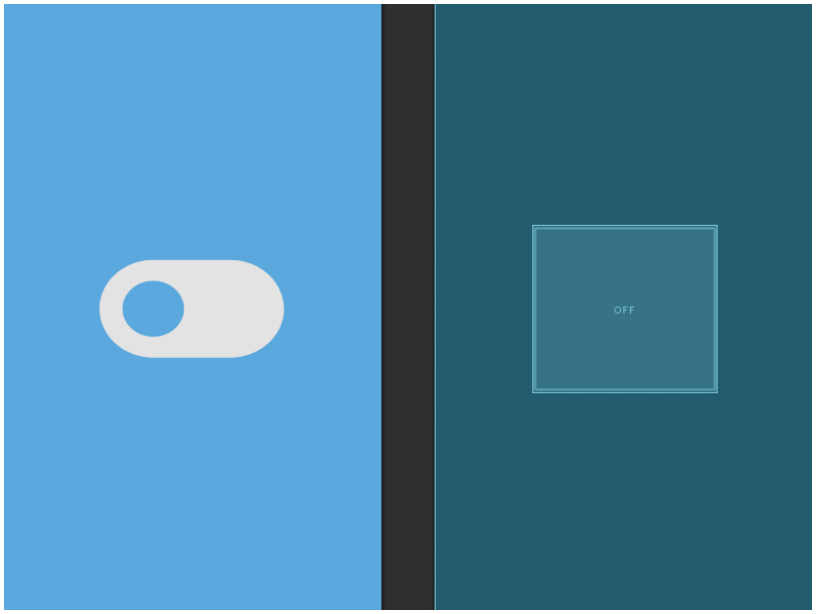


Imagen 20 Pantalla gráfica botón control ventilación

```
class refriActivity : AppCompatActivity() {
    var led1: Any? = 0

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_refri)

        val database = Firebase.database
        val myRef = database.getReference( path: "led1")

        myRef.addValueEventListener(object : ValueEventListener{
            override fun onDataChange(dataSnapshot: DataSnapshot) {
               led1= dataSnapshot.getValue()
                if (led1.toString().toInt()==0){
                    refritoggleButton.isChecked = false
                    led1 = 1024
                }else{
                    refritoggleButton.isChecked = true
                    led1 = 0
                }
            }

            override fun onCancelled(error: DatabaseError) {
                Log.w( tag: "TAG", msg: "Failed to read value",error.toException())
            }
        })
        refritoggleButton.setOnCheckedChangeListener { buttonView, isChecked ->
            if (isChecked){
                myRef.setValue(led1)
            }else{
                myRef.setValue(0)
            }
        }
    }
}
```

Imagen 21 Programa control de ventilación

En la imagen 19 se puede observar que en la pantalla solamente hay un toggle boton ese botón nos permitirá mediante el programa que se puede observar en la imagen 20 controlar la puesta en marcha del sistema de ventilación o apagarlo. El sistema será el mismo que para el control de sistema de calefacción que se puede observar en la imagen 21 y 22.

7.3.4. Control sistema de calefacción

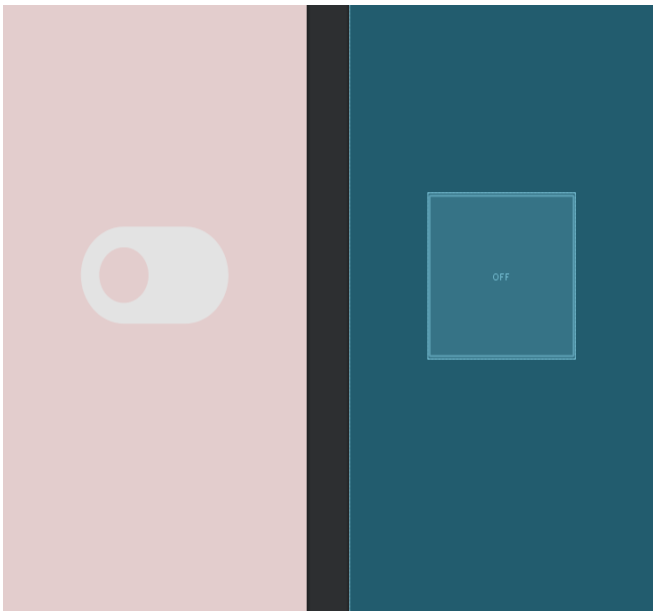


Imagen 22Pantalla Grafica sistema calefacción

```
var led: Any? = 0

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_cale)

    val database = Firebase.database
    val myRef = database.getReference( path: "led")


    myRef.addValueEventListener(object : ValueEventListener{
        override fun onDataChange(datSnapshot: DataSnapshot) {
            led = datSnapshot.getValue()
            if(led.toString().toInt()==0){
                caleToggleButton.isChecked = false

                    led = 1024

            }else{
                caleToggleButton.isChecked = true

                    led = 0

            }

        }

        override fun onCancelled(error: DatabaseError) {
            Log.w( tag: "TAG", msg: "Failed to read Value",error.toException())
        }
    })

    caleToggleButton.setOnCheckedChangeListener { buttonView, isChecked ->
        if (isChecked){
            myRef.setValue(led)
        }else{
            myRef.setValue(0)
        }
    }
}
```

Imagen 23 Pantalla programa del sistema de calefacción

7.3.5. Fragment de la pantalla Humedad

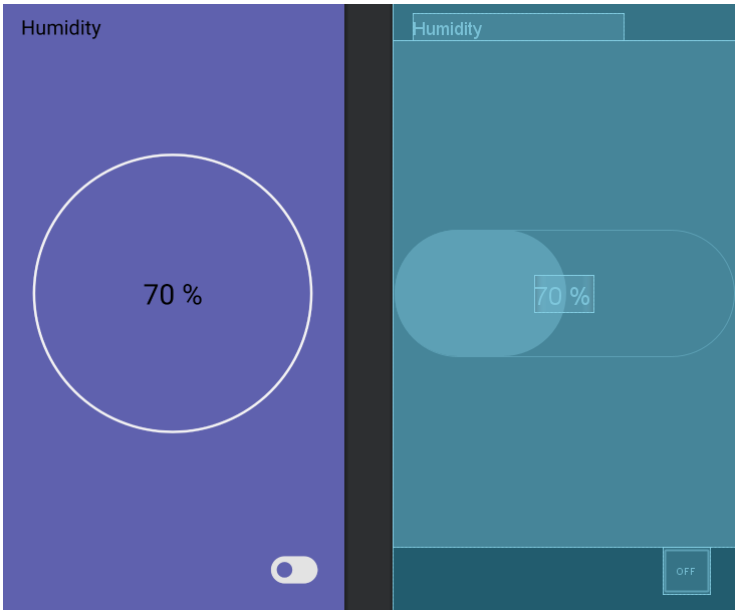


Imagen 24 Pantalla gráfica del sistema de Humedad

```
class HumidityFragment : Fragment() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(R.layout.fragment_humidity, container,  attachToRoot: false)
    }

    override fun onActivityCreated(savedInstanceState: Bundle?) {
        super.onActivityCreated(savedInstanceState)

        val database = Firebase.database
        val myRef = database.getReference()
        var dht11 = 0

        myRef.addValueEventListener(object : ValueEventListener {
            @SuppressLint( ...value: "SetTextI18n")
            override fun onDataChange(dataSnapshot: DataSnapshot) {
                val value: String = dataSnapshot.child( path: "humidity").getValue().toString()
                dht11  = dataSnapshot.child( path: "state").getValue().toString().toInt()

                humidityProgressBar.progress = value.toInt()
                humidityTextView.text = value + " %"

                if (dht11 == 0){
                    dht11ToggleButton.isChecked = false
                    dht11 = 1024
                }else{
                    dht11ToggleButton.isChecked = true
                    dht11 = 0
                }


            }
```

Imagen 25 Programa pantalla Humedad

```
        override fun onCancelled(error: DatabaseError) {
            Log.w( tag: "TAG",  msg: "Failed to read value.", error.toException())
        }
    })

    dht11ToggleButton.setOnCheckedChangeListener { buttonView, isChecked ->
        if (isChecked) {
            myRef.child( pathString: "state").setValue(dht11)
        } else {
            myRef.child( pathString: "state").setValue(0)
        }
    }
}
```

Imagen 26 Programa de la pantalla Humedad

En la imagen 23 está la parte grafica de nuestra pantalla que nos mostrara la humedad y su valor a medida que pasa el tiempo. En este fragment solo es una pantalla de información a la humedad del ambiente constante. En las imágenes 24 y 25 se puede observar el tipo de programa que se usa para enlazar los datos de Firebase con Android y que directorio tiene que acceder para obtener estos valores.

7.4. Pantalla control de luz

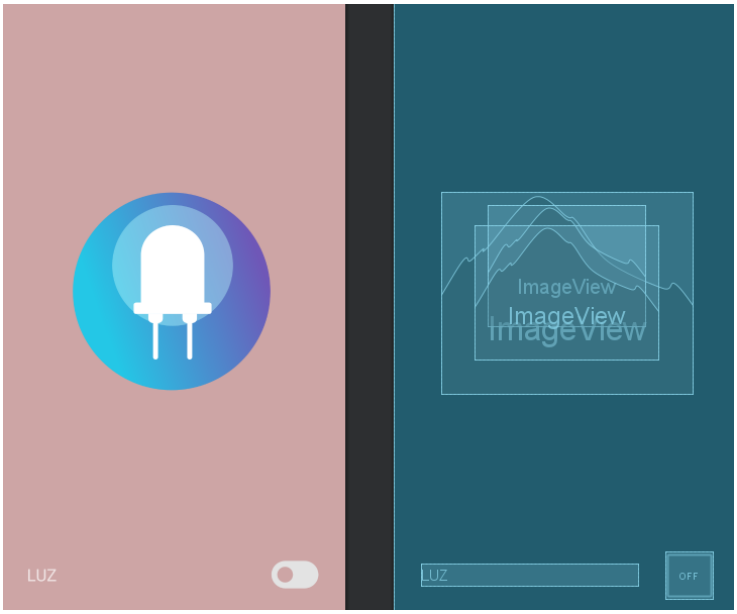
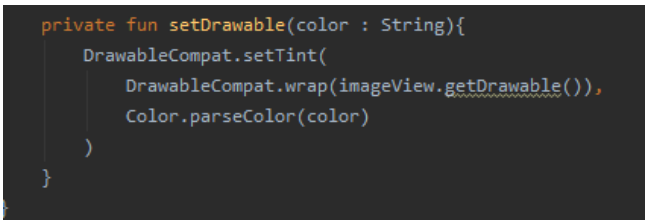


Imagen 27 Pantalla gráfica del control de luz

```
class ilumActivity : AppCompatActivity() {

    var luz: Any? = 0

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_ilum)

        val database = Firebase.database
        val myRef = database.getReference( path: "luz")
        setDrawable("#00BFFFFF")

        myRef.addValueEventListener(object : ValueEventListener{
            override fun onDataChange(dataSnapshot: DataSnapshot) {
                luz = dataSnapshot.getValue()

                if (luz.toString().toInt()==0){
                    luzToggleButton.isChecked = false
                    setDrawable("#00BFFFFF")
                    luz = 1024
                }else{
                    luzToggleButton.isChecked = true
                    setDrawable("#66BFFFFF")
                    luz = 0
                }

            }

            override fun onCancelled(error: DatabaseError) {
                Log.w( tag: "TAG", msg: "Failed to read value",error.toException())
            }
        })
        luzToggleButton.setOnCheckedChangeListener { buttonView, isChecked ->
            if (isChecked){
                myRef.setValue(luz)
            }else{
                myRef.setValue(0)
```

Imagen 28 Programa de control de luz



```
private fun setDrawable(color : String){
    DrawableCompat.setTint(
        DrawableCompat.wrap(imageView.getDrawable()),
        Color.parseColor(color)
    )
}
```

Imagen 29 Programa de control de luz

En esta pantalla tenemos la pantalla gráfica como se puede observar en la imagen 26 y su correspondiente programa en las imágenes 27 y 28 que nos permitirán controlar el encendido y apagado de la luz. EL control se hace mediante un botón de tipo toggle y en la imagen 28 se ha configurado una función de dibujo que permite cambiar el color de fondo cuando el usuario activa y desactiva la luz. Esta pantalla no tiene mucha complicación ya que consiste en el uso de un botón que envía y recibe la información dependiendo de si está o no encendida la luz mediante la base de datos de Firebase.

7.5. Pantalla Calendario

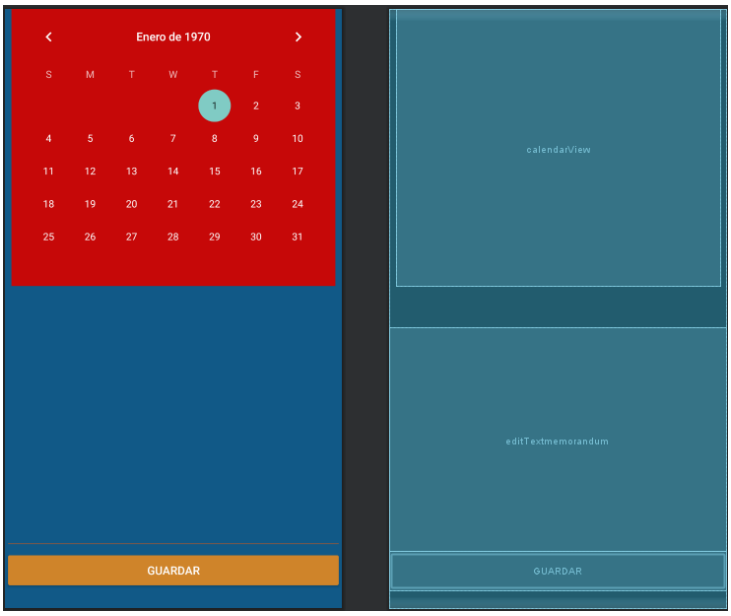


Imagen 30 Pantalla gráfica de calendario y Registro de eventos.

```
class calendarActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_calendar)

        val archivos = fileList()
        if (ArchivoExiste(archivos, NombreArchivo: "memorandum.txt")) {
            try {
                val archivo = InputStreamReader(openFileInput( name: "memorandum.txt"))
                val br = BufferedReader(archivo)
                var linea = br.readLine()
                var memorandumCompleta = ""
                while (linea != null) {
                    memorandumCompleta = """
                $memorandumCompleta$linea

                """.trimIndent()
                    linea = br.readLine()
                }
                br.close()
                archivo.close()
                editTextmemorandum.setText(memorandumCompleta)
            } catch (e: IOException) {
            }
        }
    }
    private fun ArchivoExiste(archivos: Array<String>, NombreArchivo: String): Boolean {
        for (i in archivos.indices) if (NombreArchivo == archivos[i])
            return true
        return false
    }
```

Imagen 31 Programa de Registro de eventos

```
fun Guardar(view: View?) {
    try {
        val archivo = OutputStreamWriter(openFileOutput( name: "memorandum.txt", MODE_PRIVATE))
        archivo.write(editTextmemorandum.getText().toString())
        archivo.flush()
        archivo.close()
    } catch (e: IOException) {
    }
    Toast.makeText( context: this,  text: "Evento Guardado correctamente", Toast.LENGTH_SHORT).show()
    finish()
}
```

Imagen 32 Programa del botón Guardar evento

En la imagen 29 se puede observar el calendario y una multiline text esta ventana nos permitirá registrar y guardar diferentes eventos en un orden. El calendario ha sido incorporado mediante el código propio de Android Studio. En la imagen 30 se puede observar que se ha establecido unos parámetros que permitirán al usuario registrar su evento y por consiguiente mediante la función de la imagen 31 permitirá que el botón de guardar sea operativo que una vez se pulse se mostrara una notificación al usuario verificando su registro.

7.6. Pantalla Dispensador de agua y comida

En la pantalla del dispensador de comida se han creado dos botones como se muestran en la imagen 32 un botón será para el control del agua y otro para el de la comida. A lo que respecta en la pantalla de control consiste en crear una configuración y unas funciones que permitan al usuario acceder a la siguiente pantalla cuando se pulse uno de los dos botones. Como se puede ver la configuración de la imagen 33.

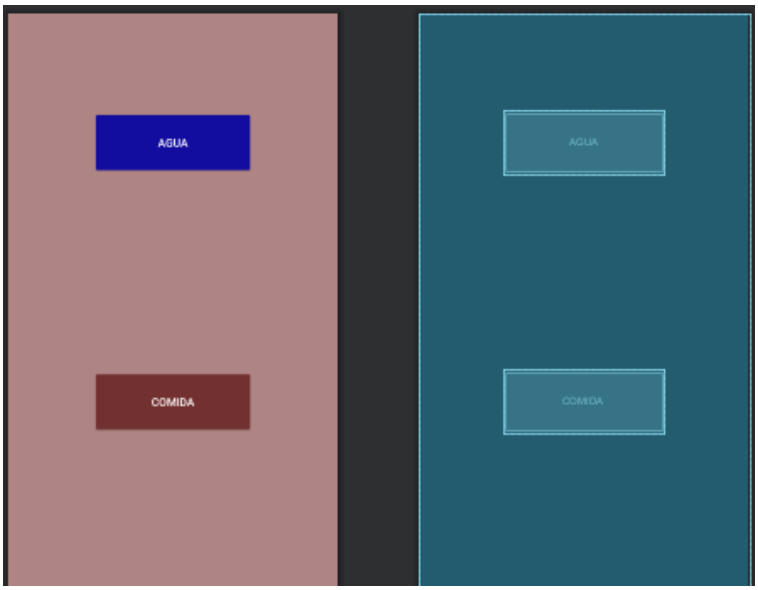


Imagen 33 Pantalla grafica selección de dispensador

```
class foodActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_food)
        setup()

    }

    private fun setup() {

        aiguaButton.setOnClickListener {  it: View!
            aiguashow()
        }

        comidaButton.setOnClickListener {  it: View!
            comidashow()
        }
    }

    private fun aiguashow (){
        val aiguaIntent = Intent( packageContext: this, aiguaActivity::class.java).apply {  this: Intent

        }
        startActivity(aiguaIntent)
    }
    private fun comidashow (){
        val comidaIntent = Intent( packageContext: this, ComidaActivity::class.java).apply {  this: Intent

        }
        startActivity(comidaIntent)
    }
}
```

Imagen 34 Programa Botones de comida y Agua

7.6.1. Pantalla de Agua

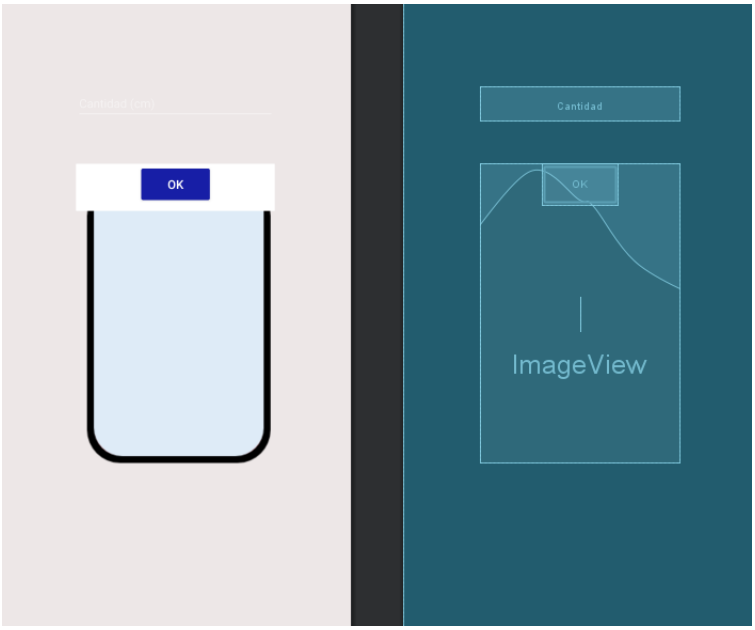


Imagen 35 Interfaz gráfica del dispensador de Agua

```
class aiguaActivity : AppCompatActivity() {
    var statusFb = 0
    var status = 0
    var max: String? = null
    var maxI = 0
    var sdata: DatabaseReference? = null
    var statusKorr = 0



    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(com.tfg.logactivity.R.layout.activity_aigua)

        sdata = FirebaseDatabase.getInstance().reference

        ok_btn.setOnClickListener(View.OnClickListener {  it: View!
            max = Cantidad.text.toString()
            maxI = max!!.toInt()
            sdata!!.addValueEventListener(object : ValueEventListener{
                override fun onDataChange(dataSnapshot: DataSnapshot) {
                    statusFb = dataSnapshot.child( path: "cantidad").value.toString().toInt()
                    statusKorr = maxI - statusFb
                    status =  (statusKorr.toDouble()/maxI.toDouble() * 100).toInt()
                    Log.d( tag: "statPc",Integer.toString(status))
                    sensorData.setText(Integer.toString(status) + "%")
                    if (status >= 95 && status <= 100){
                        imgData.setImageResource(com.tfg.logactivity.R.drawable.i100)
                    }else if ( status >= 90 && status <95){
                        imgData.setImageResource(com.tfg.logactivity.R.drawable.i95)
                    }else if (status >= 80 && status < 90) {
                        imgData.setImageResource(com.tfg.logactivity.R.drawable.i90)
                    } else if (status >= 70 && status < 80) {
                        imgData.setImageResource(com.tfg.logactivity.R.drawable.i80)
                    } else if (status >= 55 && status < 70) {
                        imgData.setImageResource(com.tfg.logactivity.R.drawable.i70)
                    } else if (status >= 50 && status < 55) {
                        imgData.setImageResource(com.tfg.logactivity.R.drawable.i55)
                    } else if (status >= 40 && status < 50) {
                        imgData.setImageResource(com.tfg.logactivity.R.drawable.i50)
```

Imagen 36 Programa Funcionamiento de la Pantalla de Agua

```
                    } else if (status >= 30 && status < 40) {
                        imgData.setImageResource(com.tfg.logactivity.R.drawable.i40)
                    } else if (status >= 20 && status < 30) {
                        imgData.setImageResource(com.tfg.logactivity.R.drawable.i30)
                    } else if (status >= 15 && status < 20) {
                        imgData.setImageResource(com.tfg.logactivity.R.drawable.i20)
                    } else if (status >= 10 && status < 15) {
                        imgData.setImageResource(com.tfg.logactivity.R.drawable.i15)
                    } else if (status > 0 && status < 10) {
                        imgData.setImageResource(com.tfg.logactivity.R.drawable.i10)
                    } else if (status < 0) {
                        imgData.setImageResource(com.tfg.logactivity.R.drawable.i0)
                        sensorData.setText("")
                    } else {
                        imgData.setImageResource(com.tfg.logactivity.R.drawable.i0)
                        sensorData.setText("")
                    }
                }
                override fun onCancelled(error: DatabaseError) {
                }
            })
        })
    }
}
```

Imagen 37 Programa de Funcionamiento de la Pantalla de Agua

En la pantalla de a tendremos un control bastante preciso del nivel de agua gracias al sensor ultrasónico que tiene conectado el Arduino. El Programa consiste en que el usuario pone la profundidad del recipiente y le da al botón de aceptar y como el sensor está transmitiendo constantemente le enviara la información a nuestra aplicación indicando en todo momento el nivel del agua como se puede ver en la imagen 35 y 36 esta descrito que cada diez saltos nos representen una imagen diferente dependiendo de si está lleno o vacío.

7.6.2. Pantalla de Comida

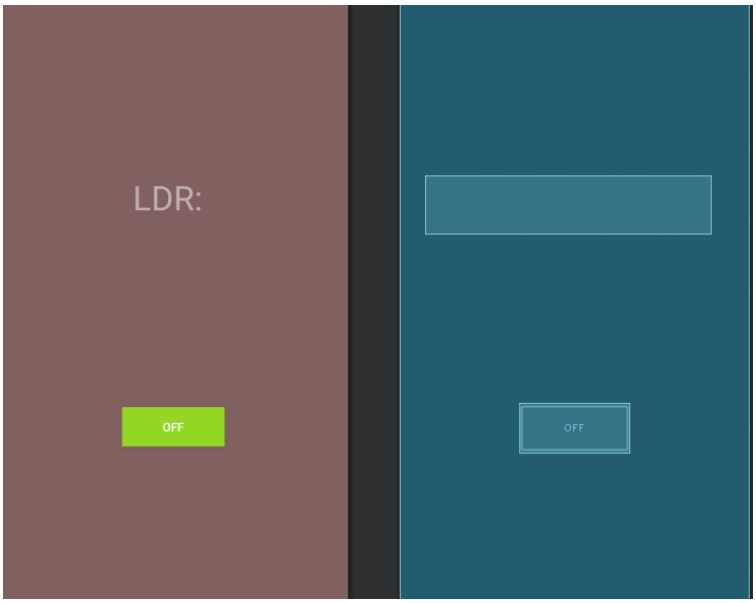


Imagen 38 Interfaz Gráfica dispensador de Comida

```
class ComidaActivity : AppCompatActivity() {
    var valueLdr: String? = null
    var valueLed1: String? = null
    var databaseReference: DatabaseReference? = null

    override fun onCreate(savedInstanceState: Bundle?) {
            super.onCreate(savedInstanceState)
            setContentView(R.layout.activity_comida)


        databaseReference = FirebaseDatabase.getInstance().getReference()

        databaseReference!!.addValueEventListener(object : ValueEventListener{
            override fun onDataChange(@NonNull dataSnapshot: DataSnapshot) {

                valueLdr = dataSnapshot.child( path: "Node1/ldr").getValue().toString()
                ldr.setText("LDR : $valueLdr")

                valueLed1 = dataSnapshot.child( path: "Node1/ldr").getValue().toString()
                if (valueLed1!!.toInt() > 10) LdrBtn.setChecked(false) else LdrBtn.setChecked(true)
            }

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

        LdrBtn.setOnCheckedChangeListener(object : CompoundButton.OnCheckedChangeListener {
            override fun onCheckedChanged(buttonView: CompoundButton?, isChecked: Boolean) {
                if (isChecked) {
                    val led1Ref = FirebaseDatabase.getInstance().getReference( path: "Node1/led1")
                    led1Ref.setValue(1)
                } else {
                    val led1Ref = FirebaseDatabase.getInstance().getReference( path: "Node1/led1")
                    led1Ref.setValue(0)
                }
            }
        })

        }
}
```

Imagen 39 Programa Control sistema LDR y LED

En la pantalla de comida solo tendremos en cuenta el sensor de resistencia lumínica o también conocido como LDR (Light depended Resistor), este sensor nos permitirá simular el sensor de tipo balanza, ya que con el podremos saber si hay luz o no sería lo mismo que decir si pesa o no, el usuario podrá saber en todo momento el estado de la comedora en cuestión, si el usuario desea poner comida activara en este caso una luz LED simulando un motor de apertura y cuando llegue al peso deseado el usuario lo podrá apagar.

8. Programación y recepción de datos en Arduino

Se programará el Arduino para que haga de central de control donde recibirá la información del ambiente y la enviara a la base de datos de Firebase que a la vez esta base enviara esos registros a la aplicación donde el usuario podrá observar los diferentes cambios que se producen en la estancia y decidir si enviar una señal para que nuestra placa actúe directamente sobre los actuadores si el usuario lo desea. Para conseguir configurar correctamente nuestra placa e integrar algunos sensores a la placa y a la base de datos de Firebase se agregará una

librería según se muestra la imagen 39. Hay que asegurarse que estén todas las librerías que se van usar estén correctamente instaladas a la placa sino a la hora de compilar el programa a la placa saltara un error.

8.1. Integración de librerías

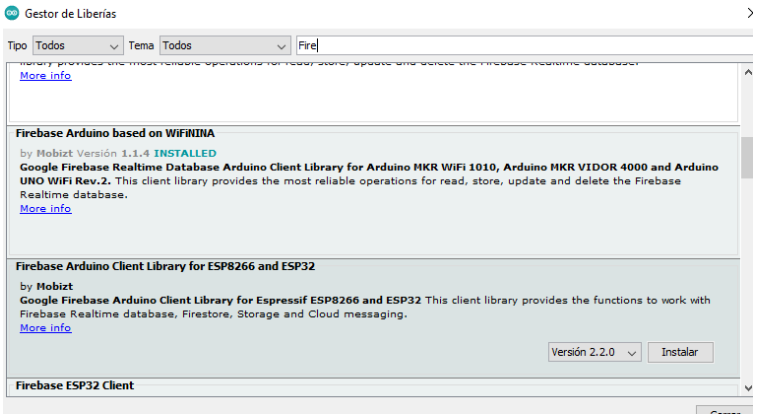


Imagen 40 Integración librería de Firebase

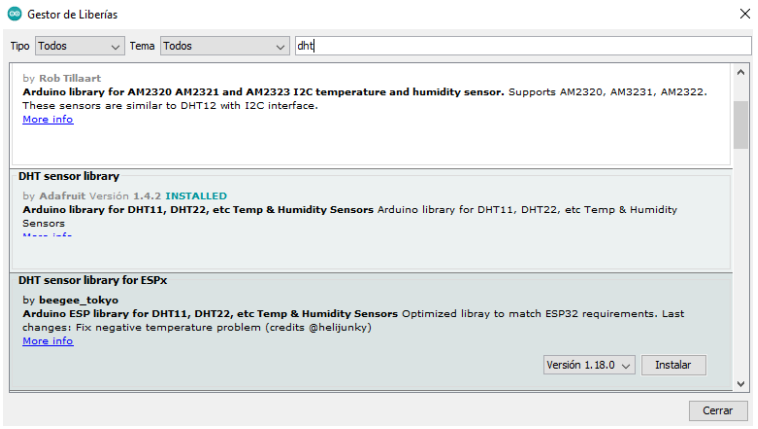


Imagen 41 Librería sensor DHT11

En la imagen 40 se puede observar que se ha integrado la librería del sensor DHT11 este sensor es conocido como el sensor de temperatura y humedad, gracias a este sensor podemos leer estos parámetros que luego son enviado a la base de datos mediante a la librería de la imagen 36. Los componentes como los actuadores que en este caso son leds no hace falta integrar nada a la librería. ya que su función depende de la señal de salida que reciban.

8.2. Programación de la placa de Arduino

Como se ha mencionado antes el software usado es un libre y gratuito y bastante fácil de entender si no se está muy acostumbrado a la programación. El primer paso a realizar será configurar la conexión a la red wifi y el enlace a la base de datos de Firebase. Como se muestra en la imagen 41.

```
#include <WiFiNINA.h>
# include "Firebase_Arduino_WiFiNINA.h"
# include <DHT.h>

// Pin de conexion

# define DHT11PIN 2

// Tipo de sensor
# define DHTTYPE DHT11
DHT dht(DHT11PIN, DHTTYPE);



//Direccion del host
# define FIREBASE_HOST "pet-comfort-default-rtdb.firebaseio.com"

# define FIREBASE_AUTH "[redacted]

//Configuracion WIFI
# define WIFI_SSID "MOVISTAR_2467"
# define WIFI_PASSWORD "942sYjYdSfhT8Dr2JuhG"

// LED
int LEDR = 12;
int LEDB = 11;
int LEDF = 13;


FirebaseData firebaseData;



void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  WiFi.begin (WIFI_SSID, WIFI_PASSWORD);
  while (WiFi.status() !=WL_CONNECTED){
    delay(500);
    Serial.print(".");
    }
    Serial.println("");
    Serial.println(" Conectado");
    Serial.println(WiFi.localIP());
    Firebase.reconnectWiFi(true);
```

Imagen 42 Configuración placa de Arduino

En esta imagen se puede observar que se define los elementos a usar para que la placa pueda acceder a la librería una vez estén instalados. Luego se define los pines de salida para que sean funcionales una vez la placa reciba órdenes. Se define los parámetros que permitirán enlazar la placa con la base de datos de Firebase se define primero la ubicación de registro de datos y luego se pone el código de autenticación personal de Firebase como se puede ver esta borrada ya que es un parámetro que el usuario no puede observar para evitar que alguien pueda acceder y modificar nuestra configuración. También se definirá el nombre y la clave de la red wifi a conectar la placa. En el void setup es donde se ejecutar a la configuración haciendo que la

placa se conecte a la base de datos y a la red wifi como se puede observar en la imagen 41, se ha podido observar su funcionamiento mediante el monitor serial.

```
Firebase.begin(FIREBASE_HOST,FIREBASE_AUTH,WIFI_SSID,WIFI_PASSWORD);

pinMode(LEDR,OUTPUT);
pinMode(LEDB,OUTPUT);
pinMode(LEDF,OUTPUT);

dht.begin();

}

void loop() {
  // put your main code here, to run repeatedly:
  if (Firebase.setFloat(firebaseData, "/temperature", dht.readTemperature())){

  //Success, then read the payload value return from server
  //This confirmed that your data was set to database as float number

  if (firebaseData.dataType() == "float")
    Serial.println(firebaseData.floatData());

} else {
  //Failed, then print out the error detail
    Serial.println(firebaseData.errorReason());
}
 if (Firebase.setFloat(firebaseData, "/humidity", dht.readHumidity())){



  if (firebaseData.dataType() == "float")
    Serial.println(firebaseData.floatData());

} else {
  //Failed, then print out the error detail
    Serial.println(firebaseData.errorReason());
}
```

Imagen 43 Configuración del sensor DHT11

En la imagen 42 se puede observar que en el sistema de void loop permitirá ejecutar la configuración del sensor en bucle leyendo continuamente la temperatura y la humedad del ambiente y enviando los datos a la base de datos de Firebase. En la imagen 43 se puede ver como se ha configurado la recepción de la señal para las luces led simulando los diferentes actuadores donde la placa recoge la información registrada en la base de datos

```
//LED

Firebase.getInt(firebaseData,"/led");
int dato=firebaseData.intData();
analogWrite(LEDR,dato);
Serial.println(dato);

Firebase.getInt(firebaseData,"/led1");
int dato1=firebaseData.intData();
analogWrite(LEDB,dato1);
Serial.println(dato1);

Firebase.getInt(firebaseData,"/luz");
int dato2=firebaseData.intData();
analogWrite(LEDF,dato2);
Serial.println(dato2);
```

Imagen 44 Configuración Actuadores LED

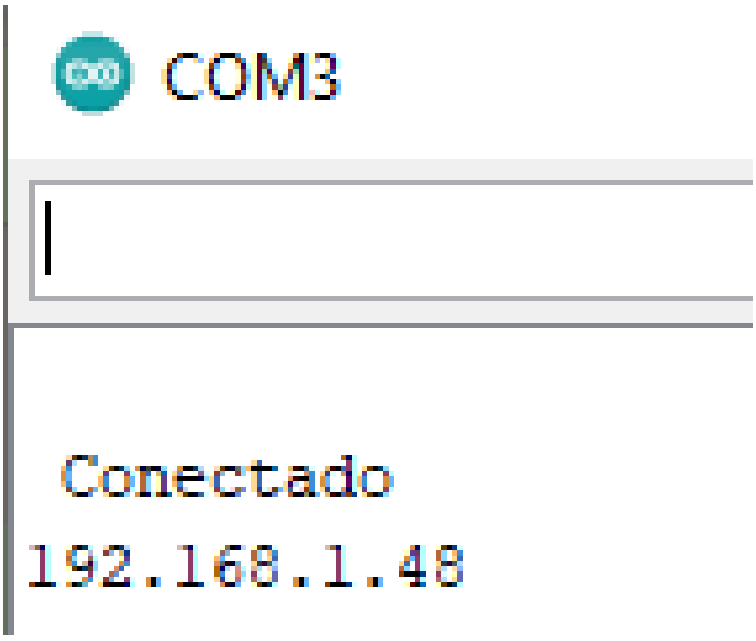


Imagen 45 Verificación de que la placa se conectó a internet

Una vez conectado a la base de datos de Firebase y a los sensores y actuadores se probará que el programa se compile correctamente y observaremos cómo evoluciona en el monitor serial de Arduino.

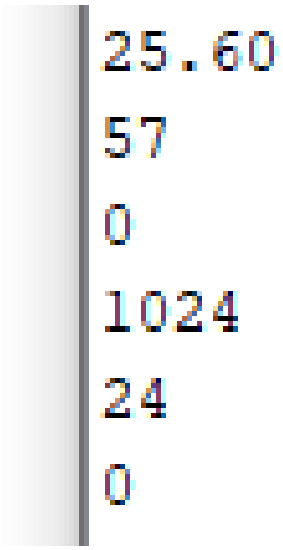


Imagen 46 Verificación de los componentes conectados a la placa

El primer valor que se puede observar es el correspondiente a la temperatura el segundo valor al de la humedad el tercero al sistema de calefacción que en este caso esta nombrado como led, el cuarto valor corresponde al sistema de iluminación que su nombre es luz el quinto parámetro se repite con el de temperatura ya que varía en el tiempo y el ultimo valor corresponde al sistema de refrigeración que equivale al valor led1. El valor de la humedad no varía tanto como el de la temperatura por eso el propio sensor no varía mucho ese parámetro debido a su sensibilidad.

8.3. Componentes Arduino

Imagen 47 Led RGB. Fuente Elegoo

Este es el actuador que nos permitirá simular el encendido del sistema de calefacción en luz roja y en el color azul simulará la puesta en marcha del sistema de ventilación. Cada vez que el

usuario enciende un sistema u otro se encenderá dicha luz también permite tener las dos luces led encendidas a la vez simulando que el usuario dejo conectado los dos sistemas a la vez, si eso sucede se verá un color rosado en los Led.

Imagen 48 LED 2 colores. Fuente Elegoo

Este led de 2 colores se usará para simular el encendido y apagado del sistema de iluminación.

Imagen 49 Sensor temperatura y humedad. Fuente Elegoo

Este será el sensor que nos permitirá saber la temperatura y la humedad de la sala en todo momento ya que registra los datos de forma continua, permitiendo así el usuario saber cómo va variando esa temperatura y humedad. Este sensor es conocido como sensor DHT11.

Imagen 50 Sensor Ultrasónico. Fuente Elegoo

Este sensor también es conocido como HC-SR04. Este sensor medirá la distancia del recipiente donde está ubicado el líquido en cuestión, permitiendo saber en todo momento la capacidad del recipiente y mantener al usuario informado para saber el estado del recipiente.

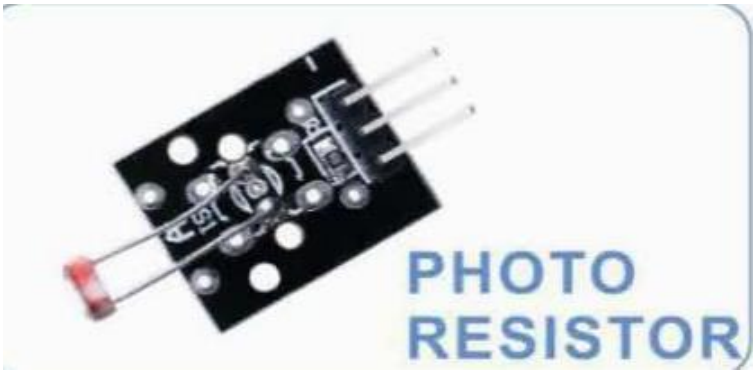


Imagen 51 Foto Resistencia. Fuente Elegoo

Este sensor nos permitirá llevar a cabo la simulación del dispensador de comida, ya que simulara un sensor de peso dependiendo de la resistencia lumínica que tenga indicara la cantidad de comida que habría en el plato y así el usuario saber si abrir o no el motor de dispensador, la función de motor estará dado por una luz LED de tipo SMD que se mostrara en la siguiente imagen.

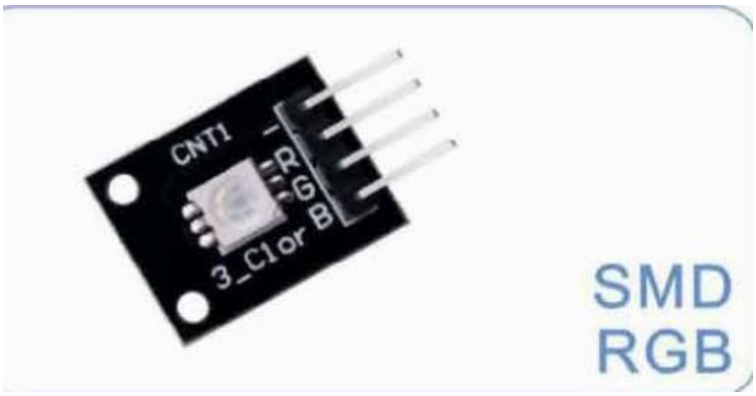


Imagen 52 LED tipo SMD. Fuente Elegoo

9. Análisis del impacto ambiental

El consumo diario de mi ordenador de sobremesa es de unos 750 W x 8 horas/día con un total de potencia consumida por día de unos 6 kWh. El uso suele ser intensivo durante la semana ya que es el ordenador que uso tanto para trabajar como para otras actividades lúdicas. En cambio, el de mi portátil es de 120 W x 5 horas/día con un total de 600 W

La placa de Arduino su microcontrolador tiene un consumo de 5 voltios y una corriente de 40 mA en continua, los demás componentes de la placa pueden trabajar de 7 a 12 voltios, dependiendo del número de actuadores y sensores estén conectados a dicha placa. Eso influirá en el consumo y rendimiento de la placa.

El consumo del dispositivo móvil depende del uso que se le dé, si el dispositivo requiere estar conectado a la red muy seguido o no.

Los sensores y actuadores usados en este proyecto tienen un consumo de entre 3.3 a 5 voltios. La cantidad de corriente suministrada depende de cada componente usado. No lo tengo en cuenta ya que el consumo de cada componente usado es tan pequeño que casi no se nota en el consumo diario.

La generación de CO2 de cada dispositivo depende de la energía que use será mayor o menor, según nos indica la Comisión Europea de que un ordenador libera entre 52 y 234 gramos de CO2 teniendo en cuenta una tensión de entre 80 y 360 V. Según estudios de Universidades y empresas han determinado que un dispositivo de gama alta o normal genera entre 47 y 95 kilogramos de CO2 en una hora.

Aparatos	Potencia
Ordenador Sobremesa	6000 W
Arduino Uno microcontrolador	(5 V * 40 mA) = 0,2 W
Arduino Uno demás componentes de la placa	(7-9 V) * 40 mA = 0,36 W
Portátil	600 W
Consumo Total	**6600,56 W**
Generación CO2	
Ordenador Sobremesa	52 a 234 gr CO2

Dispositivo móvil	47 kilogramos en un ahora

Tabla 4 Consumo y generación de los materiales usados en este proyecto

Para llevar a cabo el reciclaje de componentes electrónicos se llevarían a una planta de reciclaje dedicada a la reutilización de dichos componentes o a la fundición de estos mismos.

10. Planificación Diagrama de Gantt

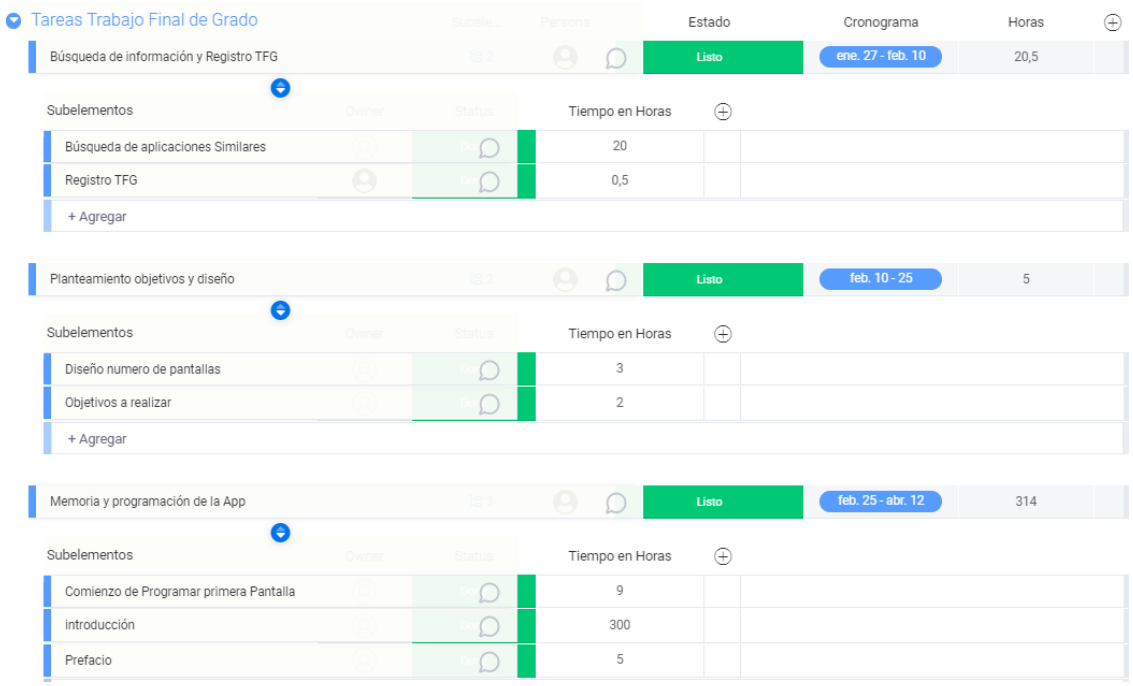


Imagen 52 Tiempo y subtemas realizados. Fuente usado Monday

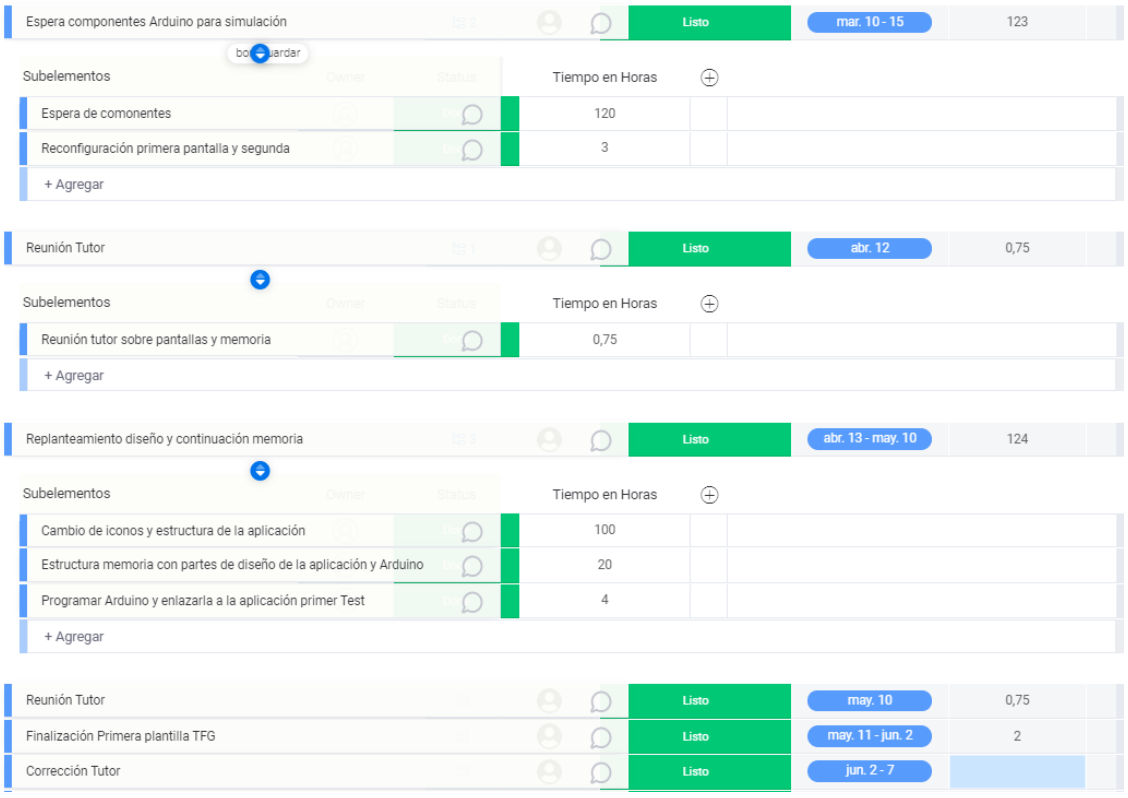


Imagen 53 Tiempo y subtemas realizados. Fuente usado Monday

Imagen 54 Tiempo y subtemas realizados. Fuente usado Monday

11. Costes de montaje

Coste de Material (Hardware)	
Nombre material	Coste
Arduino UNO Wifi REV2; 8,2x6x2,6 cm;40gr.	47,07 €
Elegoo 37v1 Kit de Módulos de Sensores con tutorial	27,99 €
Elegoo 120 Piezas de Cable DuPont, 40 pines M-H, 40 pines H-H, 40 pines M-M	6,99 €
Ordenador de mesa	1.480 €
Portátil	600 €
Programador	30 €/h (30€/h * 600h = 18.000 €)
Total coste Material	20.162,05 €
Coste de Software	
Dominio de Firebase	Gratuito
Uso de Software IDE	Gratuito
Uso software Android Studio	Gratuito

Tabla 2 Coste de materiales

En la tabla 2 tenemos los costes de lo que nos costara crear este sistema para poder simularlo. Este sería el sistema básico sin uso de actuadores como los motores y electroválvulas.

En los gastos de obtención de los materiales no se ha tenido en cuenta el coste del transporte ya que era gratis. En los costes de Firebase como se ha mencionado anteriormente son gratuitos hasta un límite de usuarios registrados en la app y de uso de registro de datos y descarga de dichos datos. El coste del instalador va determinado por el número de horas que esta tarde en montar el sistema en la estancia, cobrando así 15 hasta 70 euros la hora. El coste del programador depende del número de horas que tarde en crear un nuevo programa y este operativo. En este país dependiendo de la dificultad y experiencia se le puede pagar desde 15 a 90 euros la hora. En la tabla 2 solo he tenido en cuenta el coste del material usado y de los equipos que me han permitido llevar a cabo dicho proyecto.

Coste de Material (Hardware)	
Nombre material	Coste
Arduino UNO Wifi REV2; 8,2x6x2,6 cm;40gr.	47,07 €
Elegoo 37v1 Kit de Módulos de Sensores con tutorial	27,99 €
Elegoo 120 Piezas de Cable DuPont, 40 pines M-H, 40 pines H-H, 40 pines M-M	6,99 €
Electroválvula, 1Pc 12V G3/ 4. Electroválvula de Latón válvulas de solenoide para agua	17,39 €
HALJIA, sensor de peso célula de carga, Sensor de pesaje (5kg)+ HX711	9,99 €
Ventilador Turbo potente para mesa y suelo, regulable en 3 velocidades. Honeywell HT900E4	26,30 €
Ordenador de mesa	1.480 €
Portátil	600 €
Mano de Obra (solo montaje)	15 €/h (15€/h *6h = 90 €)
Programador	30 €/h (30€/h * 600h = 18.000 €)
Coste Total	22.518,32 €
Coste de Software	
Uso de Software IDE	Gratuito
Uso software Android Studio	Gratuito

Tabla 3 Costes Montaje real

Si el prototipo se lleva a cabo con sus componentes y materiales reales el costo de dichos materiales seria mayor como se puede observar en la tabla 3 el coste de software se mantiene gratuito siempre y cuando el número de usuarios registrados sea menor a 10000. Y el uso de subida y bajada de datos no supere los 10 GB, entonces Firebase aplicaría una tarifa por el uso de su plataforma. En la tabla 3 se puede observar lo que costaría desarrollar la aplicación y su placa de control con los elementos a usar en el montaje de la sala incluyendo el tiempo de trabajo del programador y del instalador.

12. Conclusiones

En conclusión, en este proyecto he conseguido llevar a cabo el diseño y simulación de una aplicación de control de una sala para mascotas desde el control de luz, climatización, dispensador de comida y bebida y poner un calendario con posibilidad de registrar un evento. Al principio tuve problemas a la hora de programar ya que algunas pantallas que el programa que busca para programar o saber cómo se hace estaba en java y tenía que ir buscando información de cómo ponerlo en Kotlin y que fuera funcional, ya que el propio software de Android te permite el cambio hay códigos que hay que cambiar a mano, ya que el mismo software no sabe exactamente que uso tendrá. En el prototipo he conseguido con lo que tenía a mano diseñar un sistema bastante parecido a lo que sería el prototipo real. He usado sensores diferentes a lo que realmente tendría, pero la forma de configurarlos sería igual. Lo mismo pasaría con los actuadores.

13. Web grafía

Para a llevar a cabo este proyecto use video tutoriales de cómo programar las pantallas y de cómo enlazar la placa de Arduino con Firebase. Mire en blogs de algunos problemas relacionado con alguna actualización de los softwares usados.

[1] GOOGLE: [Consulta: 9 junio 2021].Disponible a : <Firebase console (google.com)>

[2] AMAZON: [Consulta: 1 mayo 2021].Disponible a: <Arduino UNO WiFi REV2 [ABX00021]: Amazon.es: Bricolaje y herramientas>

[3] GOOGLE: [Consulta: 9 junio 2021].Disponible a: <Instalación y configuración en Android | Firebase Realtime Database (google.com)>

[4] FREEPIK: [Consulta 9 junio 2021].Disponible a: <Descarga gratis Vectores, Fotos de Stock y PSD | Freepik>

[5] ARDUINO: [Consulta 9 junio 2021].Disponible a: <Software | Arduino>

[6] DEVELOPERS: [Consulta 9 junio 2021].Disponible a: <Download Android Studio and SDK tools | Android Studio>

[7] GITHUB: [Consulta 9 junio 2021].Disponible a: <GitHub: Where the world builds software · GitHub>

[8] TUTORIALESPDF: [Consulta 25 marzo 2021]. Disponible a: <Kotlin - Tutoriales en PDF>

[9] MONDAY: [Consulta 7 junio 2021]. Disponible a: <GANTT (monday.com)>

[10] ELEGOO:[Consulta 7 junio 2021]. Disponible a:< Arduino Kits User Support – ELEGOO Official>

ANEXOS

I. Fotografías montaje

I. FOTOGRAFIAS MONTAJE

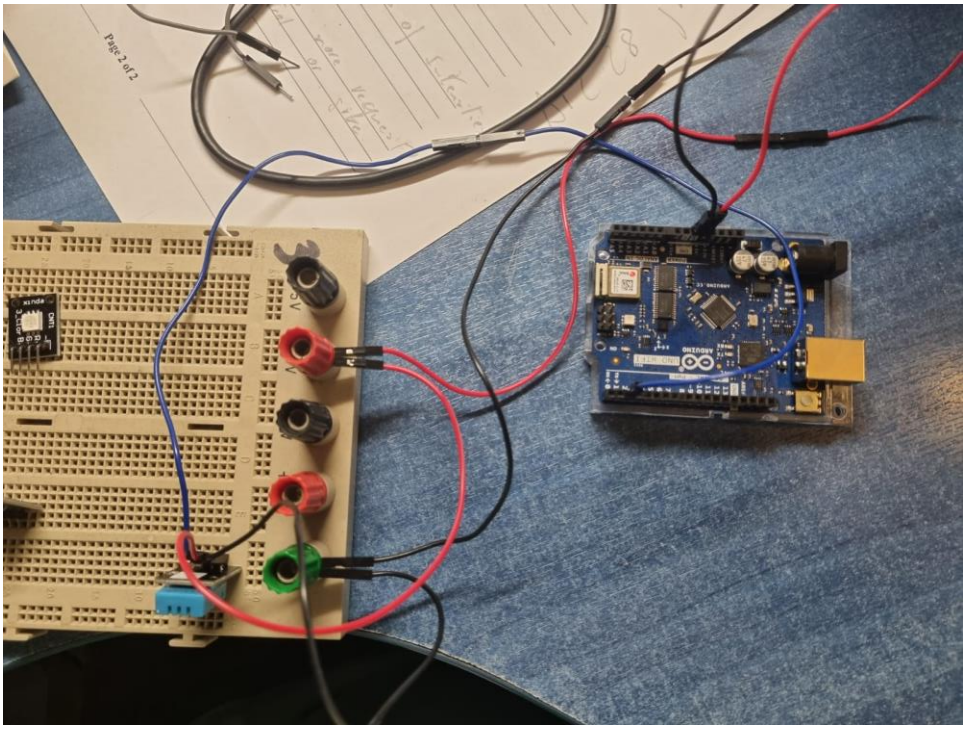


Fig. 1 Montaje y conexión sensor DHT11

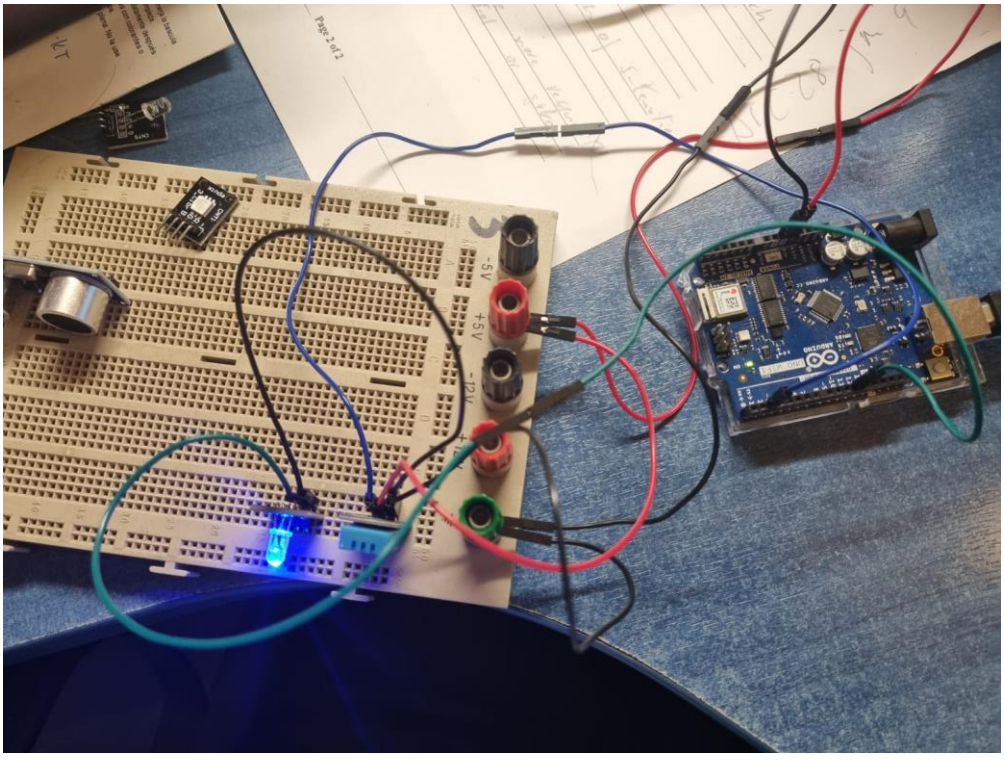


Fig. 2 Montaje y Conexión LED Sistema ventilación y calefacción.

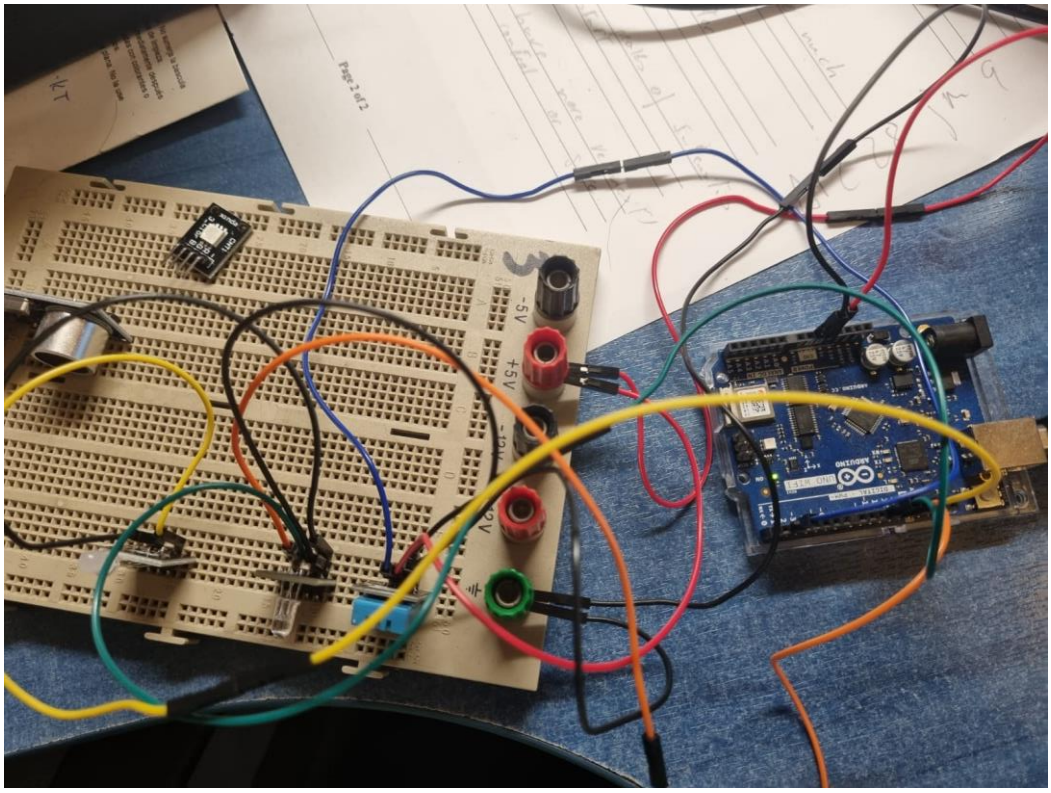


Fig. 3 Montaje y conexión LED 2 colores

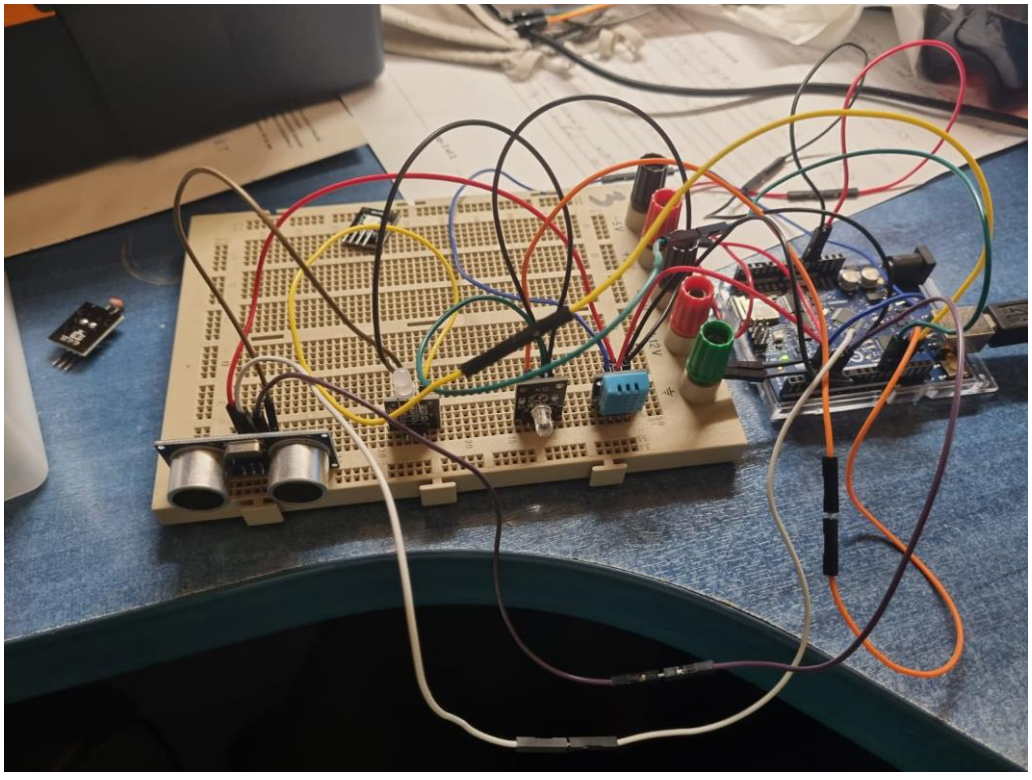


Fig. 4 Montaje y conexión sensor Ultrasonidos

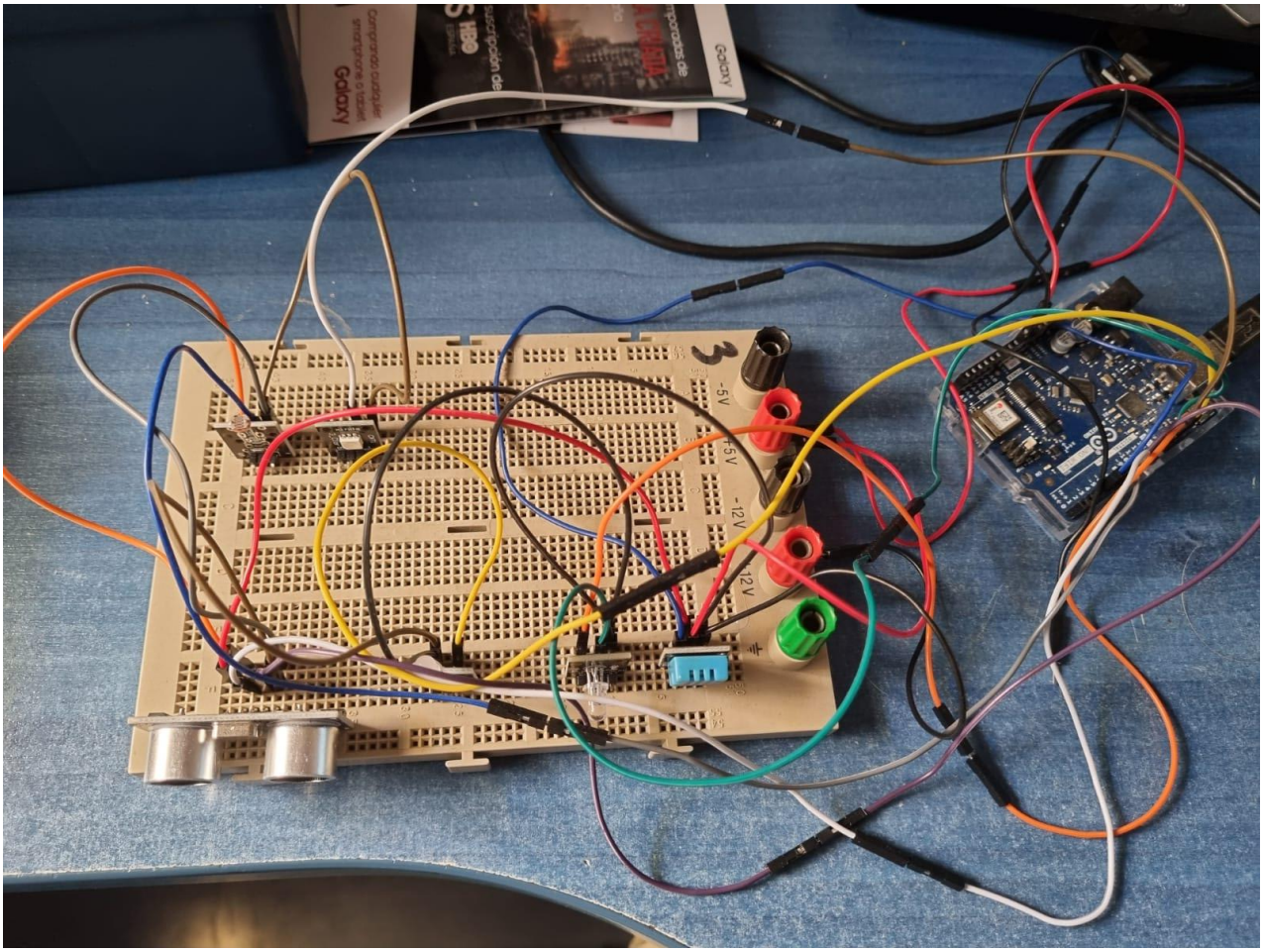


Fig. 5 Montaje y conexión LED SMD y resistencia LDR