CARRERA DE INGENIERÍA ELECTRÓNICA

MEMORIA DEL TRABAJO FINAL

# Control de aires acondicionados a través de una aplicación móvil


**Autor:**
**Diego Sebastian Scicchitano**

Director:
Esp. Ing. Milton Eduardo Sosa


Jurados:
Marcelo Romeo (UNSAM)
Jorge Sindeman (UNSAM)
Gustavo Guitera (Siemens SA)

*Este trabajo fue realizado en la ciudad de Buenos Aires,*
*entre Noviembre de 2021 y Julio de 2022.*

# *Resumen*

Este proyecto fue desarrollado en el contexto de un Proyecto Final Integrador de la carrera de Ingeniería Electrónica. El objetivo del mismo es reducir el uso de energía en la UNSAM.

En la presente memoria se describe el desarrollo e implementación de un sistema, cuya función principal es otorgarle al usuario la capacidad de controlar todos los aires acondicionados de la Universidad Nacional de San Martin desde una aplicación móvil y sin la necesidad de acceder a cada aula. Dicha aplicación reportará el valor de la temperatura de la sala, el estado del cada aire acondicionado y la detección de movimiento en tiempo real al usuario. Además, mostrará una tabla con los datos históricos de cada aire acondicionado en las últimas horas.

# Índice general

# Índice de figuras

# Índice de tablas

# Capítulo 1

# Introducción general

En el presente capítulo se presentará la razón por la cual surge la necesidad de llevar a cabo este proyecto y cuáles son las prestaciones que ofrece el producto.

## 1.1. Problema a resolver

Los directivos de la UNSAM han manifestado que existe un consumo de energía excesivo en la universidad, principalmente en los aires acondicionados. De hecho, empleados de seguridad remarcan que es frecuente que los empleados se vayan de sus oficinas y olviden apagar los aires acondicionados, de esta manera los mismos funcionan durante horas en aulas vacías.

Por otro lado, hay ciertas salas dentro de la universidad para las cuales el personal con acceso a las mismas es restringido. Por lo tanto, a pesar de que el personal de seguridad detecte el uso innecesario del aire acondicionado, no puede acceder al mismo para apagarlo. Por este motivo, la capacidad de comandar el aire acondicionado desde un dispositivo móvil, es una herramienta clave.

## 1.2. Solución propuesta

Con el objetivo de solucionar el uso excesivo de aires acondicionados en la UNSAM, se propuso desarrollar un sistema embebido que permite, de manera remota, controlar todos los aires acondicionados de cada sala mediante una aplicación Android. Es decir, que el personal de seguridad pueda, a través de su celular, ya sea encender o apagar el aire acondicionado o incluso subir o bajar la temperatura del mismo, de esta manera, no habría necesidad de ingresar a la sala, por lo cual no existiría el problema mencionado anteriormente de que existen salas con acceso restringido.

Además, la app permite obtener el estado de los equipos de aire acondicionado, sensar la temperatura ambiente en la sala y detectar movimiento dentro del aula en tiempo real. Es decir, que el usuario podrá navegar a través de la aplicación móvil y saber en tiempo real cuál es la temperatura en la sala, saber si el aire acondicionado está encendido o no y saber cuanto tiempo paso desde que se detectó el último movimiento. Con estos datos, el usuario contara con información suficiente para decidir que acción tomar con el aire acondicionado.

Simultáneamente, cada una hora dichos parámetros serán almacenados en una base de datos, la cual podrá ser visualizada desde la app.

Por otro lado, la aplicación móvil generara advertencias cada vez que haya alguna sala cuyo aire acondicionado este encendido fuera del horario de clases, o bien este encendido y el ultimo movimiento detectado haya sido hace mas de una hora. Desde dichas advertencias, el usuario podrá apagar el aire acondicionado en cuestión.

## 1.3. Objetivos

### 1.3.1. Objetivo General

De acuerdo a lo descrito en las secciones anteriores, el objetivo general del proyecto es diseñar e implementar un sistema embebido que permita, de manera remota, obtener el estado de los equipos de aire acondicionado y controlarlos mediante una aplicación móvil.

### 1.3.2. Objetivos Específicos

El sistema deberá:

- Contar con un sensor de movimiento que permita inferir si hay gente en la sala.
- Contar con un sensor de temperatura que permita conocer la temperatura en la sala.
- Contar con un sensor de corriente no invasivo que permita determinar el consumo de corriente.
- Contar con una aplicación móvil que permita al usuario final visualizar los datos publicados en el servidor. Adicionalmente, esta aplicación permitirá enviar comandos de control al nodo in situ.
- De manera periódica, reportar y almacenar todos los datos históricos en una base de datos de:
  - El estado de los aires acondicionados
  - La temperatura
  - El movimiento
- Desplegar un conjunto de servicios de backend que permita la recuperación y almacenamiento de datos en la base de datos.
- Ser escalable en cuanto a la cantidad de nodos que se deseen desplegar

## 1.4. Alcance

En el presente proyecto se desarrollan los siguientes temas:

- Desarrollo de un programa Arduino cargado en un ESP32 que:
  - Envié comandos al aire acondicionado.
  - Lea sensores de temperatura, corriente y movimiento.
  - Envié y reciba mensajes a través de MQTT.

    - Publique los valores de los parámetros recibidos en una base de datos.
- Desarrollo e implementación de un PCB con toda la electrónica necesaria del sistema.
- Diseño e impresión 3D de una caja contenedora que proteja la electrónica de ambientes hostiles.
- Desarrollo de una aplicación Android que:
    - Haga consultas a bases de datos
    - Envié y reciba mensajes a través de MQTT.
    - Muestre los datos de los sensores en tiempo real.
    - Muestre datos históricos de todos las salas en una tabla.
- Comunicación MQTT entre el ESP32 y la aplicación Android.
- Desarrollo de servicios PHP que permitan la interacción entre el ESP32 y la aplicación Android con las bases de datos.

# Capítulo 2

# Introducción Especifica

En el presente capítulo el objetivo es explicar, sin entrar en detalles, el funcionamiento del sistema. Por lo tanto, se presentarán las herramientas utilizadas y la interacción entre las mismas.

## 2.1. Herramientas utilizadas

### 2.1.1. ESP32

ESP32 es la denominación de una familia de chips SoC[1] de bajo costo y consumo de energía. El mismo está diseñado para aplicaciones móviles, dispositivos electrónicos portátiles e Internet de las cosas (IoT). Cuenta con todas las características de vanguardia de los chips de bajo consumo, incluida la activación de reloj de granularidad fina, múltiples modos de alimentación y escalado dinámico de alimentación. Por ejemplo, en un escenario de aplicación de concentrador de sensores IoT de baja potencia, ESP32 se activa periódicamente solo cuando se detecta una condición específica. El ciclo de trabajo bajo se usa para minimizar la cantidad de energía que gasta el chip [1]. En la Figura 2.1 se muestra el diagrama funcional del ESP32 con todas las funcionalidades que el mismo ofrece.

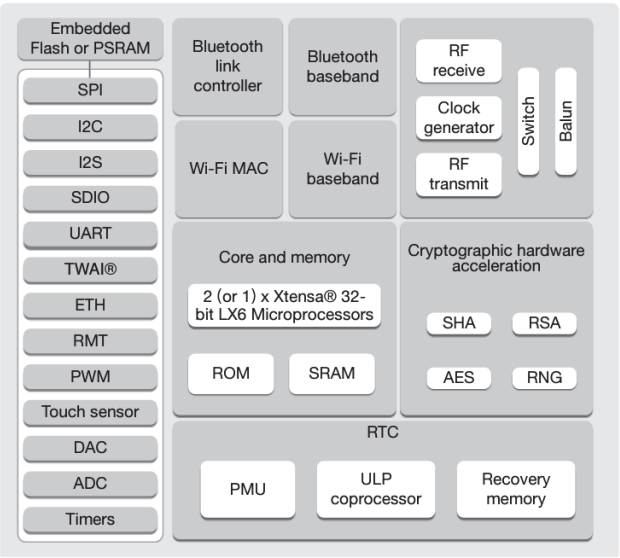



FIGURA 2.1. Diagrama de bloques funcional del ESP32 [1].

[1]SoC: System on a Chip o Sistema en un Chip es una tecnología de fabricación que integra todos o gran parte de los módulos que componen un computador en un único circuito integrado o chip.

En este proyecto, se utilizo la placa de desarrollo Nodemcu ESP32 para la lectura de sensores, reportar las variables del sistema y controlar los equipos de aire acondicionado.

La razón principal por la cual se decidió utilizar un ESP32 en vez de otro microcontrolador, además del bajo costo y consumo, es que el ESP32 tiene integrada la tecnología Wi-Fi, por lo cual ofrece todas las características necesarias para desarrollos IoT inalámbricamente.

### 2.1.2. KiCAD

KiCad es una herramienta de automatización de diseño electrónico (EDA) gratuita y de código abierto. Cuenta con captura esquemática, simulación de circuitos integrados, diseño de placa de circuito impreso (PCB), representación 3D y trazado/exportación de datos a numerosos formatos. KiCad también incluye una biblioteca de componentes de alta calidad con miles de símbolos, huellas y modelos 3D [2].

En este proyecto, se utilizó KiCAD para desarrollar un PCB sobre el cual se montaron el ESP32 y todos sus periféricos.

### 2.1.3. Android Studio

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de apps para Android y está basado en IntelliJ IDEA[2]. [3].

En este proyecto, se utilizará Android Studio para el desarrollo tanto del frontend como del backend de la aplicación móvil.

### 2.1.4. PHP

PHP es un acrónimo recursivo para "*PHP: Hypertext Preprocessor*", originalmente *Personal Home Page*, es un lenguaje interpretado libre, usado originalmente para el desarrollo de aplicaciones presentes y que actuaran en el lado del servidor, capaces de generar contenido dinámico en la *World Wide Web*[3].

Figura entre los primeros lenguajes posibles para la inserción en documentos HTML[4], dispensando en muchos casos el uso de archivos externos para eventuales procesamientos de datos.

El código es interpretado en el lado del servidor por el módulo PHP, que también genera la página web para ser visualizada en el lado del cliente [4].

En este proyecto, se desarrollaron servicios PHP como interfaz entre el ESP32 y la app con las bases de datos, tanto para escribir como para leer en las mismas.

---

[2]IntelliJ IDEA es un entorno de desarrollo integrado para programas informáticos.

[3]La World Wide Web (comúnmente conocida como WWW) es un sistema interconectado de páginas web públicas accesibles a través de Internet.

[4]HTML, siglas en inglés de HyperText Markup Language, hace referencia al lenguaje de marcado para la elaboración de páginas web.

### 2.1.5. MySQL

MySQL es un sistema gestor de bases de datos muy conocido y ampliamente usado por su simplicidad y notable rendimiento. Aunque carece de algunas características avanzadas disponibles en otros SGBD[5] del mercado, es una opción atractiva tanto para aplicaciones comerciales, como de entretenimiento, precisamente por su facilidad de uso y tiempo reducido de puesta en marcha. Esto y su libre distribución en Internet bajo licencia GPL[6] le otorgan como beneficios adicionales contar con un alto grado de estabilidad y un rápido desarrollo [5].

En este proyecto, se utilizó MySQL para administrar las bases de datos de los usuarios que están habilitados para utilizar la app, para almacenar los comandos de los aires acondicionados y para tener un registro histórico de todas las variables del sistema periódicamente.

### 2.1.6. MQTT

MQTT (Message Queuing Telemetry Transport) es un protocolo de mensajería estándar de OASIS[7] para Internet de las cosas (IoT). Está diseñado como un transporte de mensajería de publicación/suscripción extremadamente ligero que es ideal para conectar dispositivos remotos con un espacio de código pequeño y un ancho de banda de red mínimo. MQTT hoy en día se utiliza en una amplia variedad de industrias, como la automotriz, la manufactura, las telecomunicaciones, el petróleo y el gas, etc [6].

En este proyecto se utilizará la comunicación MQTT para que el ESP32 y la aplicación móvil puedan enviar y recibir mensajes entre ellos.

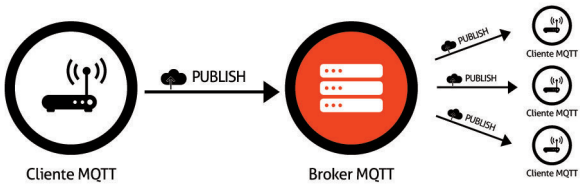



FIGURA 2.2. Diagrama de funcionamiento de la comunicación MQTT.

En la Figura 2.2 se muestra un diagrama de funcionamiento de la comunicación entre dispositivos a través de MQTT, donde un cliente publica el mensaje en el Broker MQTT y los demás dispositivos reciben dicho mensaje.

## 2.2. Funcionamiento del sistema

De acuerdo a los objetivos y especificaciones definidos en el capítulo anterior, el sistema deberá enviar comandos desde la aplicación móvil al aire acondicionado

[5]Un SGBD (Sistema Gestor de Base de Datos) es un conjunto de programas que nos permiten gestionar bases de datos.

[6]Licencia Pública General derecho de autor ampliamente usada en el mundo del software libre.

[7]La Organización para el Avance de los Estándares de Información Estructurada (OASIS) es un consorcio sin fines de lucro que trabaja en el desarrollo de estándares abiertos para ciberseguridad, blockchain, Internet de las cosas (IoT), entre otros

y enviar información de la sala hacia la aplicación móvil, es decir que hay dos direcciones en el flujo de datos los cuales se definen a continuación:

### 2.2.1. Envió de comandos al aire acondicionado

En la Figura 2.3 se muestra el diagrama de funcionamiento con todos los pasos y eventos que suceden desde que el usuario ejecuta un comando en la aplicación móvil, hasta que dicho comando es recibido por el aire acondicionado.

Cada flecha representa un evento distinto. Las mismas están numeradas secuencialmente. A continuación, se describe cada uno de los eventos.

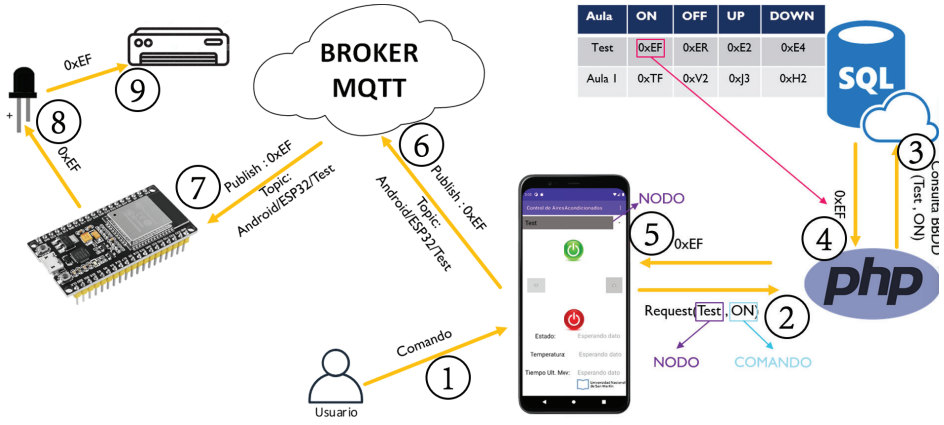



FIGURA 2.3. Diagrama de funcionamiento para el envío de comandos al aire acondicionado.

1. **Comando:** este evento ocurre cuando el usuario interactúa con la aplicación para ejecutar alguna de las cuatro instrucciones posibles (encender, apagar, aumentar temperatura o disminuir temperatura).
2. **Request**[8]**:** cuando el usuario ejecuta alguna instrucción, la app hace un request a un servicio PHP y le envía el nodo al que se debe enviar la instrucción y el comando como parámetros.
3. **Consulta a Base de Datos:** el servicio PHP hace una consulta a una base de datos SQL. La misma contiene el ID de todas los nodos y los comandos hexadecimales correspondientes a las cuatro posibles instrucciones. Entonces el servicio PHP hace la consulta en la cual le envía como parámetros la instrucción que desea ejecutar el usuario y el aula en el que lo desea ejecutar.
4. **Respuesta de Base de Datos:** en este evento, la base de datos devuelve al servicio php el código hexadecimal del comando solicitado en el aula solicitada.
5. **Recepción del comando:** una vez que la base de datos le responde al servicio PHP con el código hexadecimal, este último lo único que hace es enviarle ese código a la app en respuesta al request hecho en el paso 2.
6. **Publicación MQTT:** una vez obtenido el código hexadecimal, la app publica el mismo en el broker de MQTT.

[8]En informática, request es uno de los métodos básicos que utilizan los dispositivos para comunicarse entre sí en una red, en el un dispositivo envía una solicitud de algunos datos y el otro dispositivo responde a la solicitud.

7. **Lectura del comando:** cuando el ESP32 detecta que hay un nuevo mensaje en el broker de MQTT, hace una lectura del mensaje.
8. **Envió del comando al emisor IR:** una vez leído el comando, el ESP32 procede a enviar el código hexadecimal al emisor IR
9. **Ejecución del comando:** el emisor IR que estará apuntando al aire acondicionado ejecutara el comando.

### 2.2.2. Recepción de datos

En la Figura 2.4 se muestra el diagrama de funcionamiento para el reporte de temperatura, movimiento y estado de aires acondicionados de manera periódica. Además, se especifican todos los pasos y eventos que suceden desde que el ESP32 lee los sensores hasta que la aplicación móvil recibe el reporte.

Cada flecha representa un evento distinto. Las mismas están numeradas secuencialmente. A continuación, se describe cada uno de los eventos.

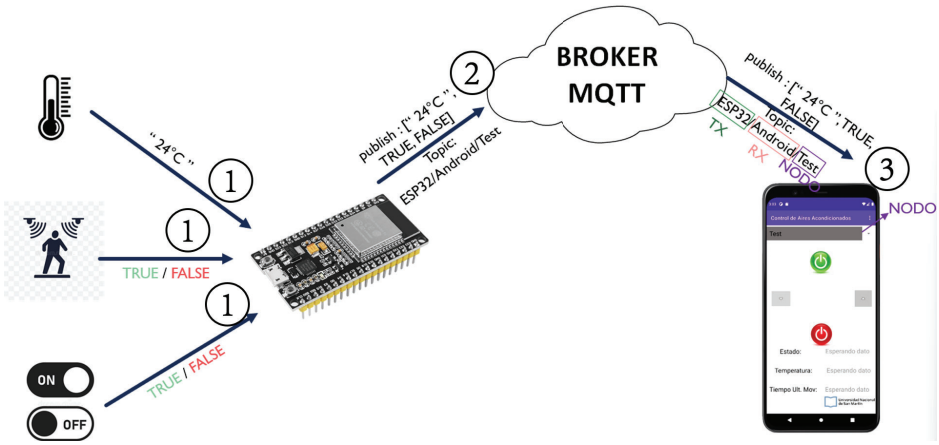



FIGURA 2.4. Diagrama de funcionamiento para la recepción de datos en la app.

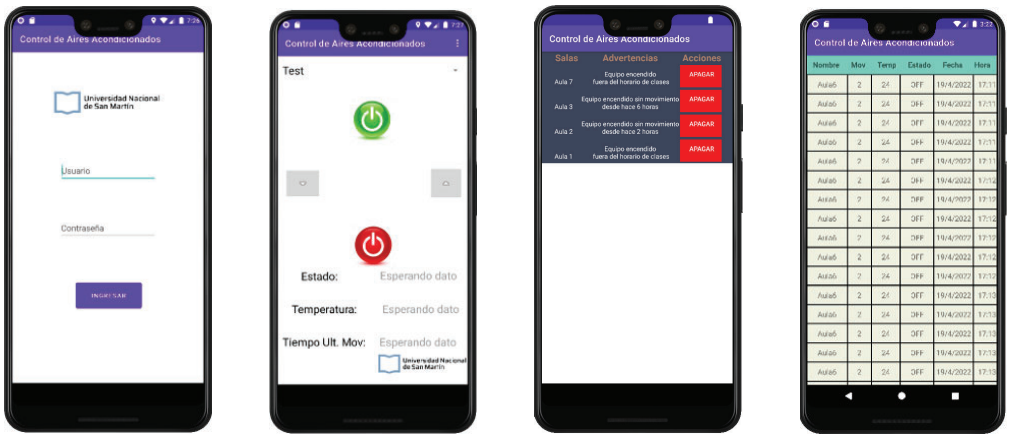
1. **Lectura de sensores:** el primer evento consiste en la lectura de los sensores de temperatura, movimiento y corriente. El ESP32 toma una lectura de los mismos cada 10 segundos.
2. **Publicación MQTT:** cada vez que el ESP32 lee un mensaje lo publica en el broker de MQTT
3. **Lectura MQTT:** cuando la app detecta que hay un nuevo mensaje en el broker de MQTT, hace una lectura del mismo.

# Capítulo 3

# Desarrollo de la Aplicación Móvil

Un método para enviar instrucciones a microcontroladores y recibir reportes de los mismos, es a través de paginas web embebidas, las cuales son mas comunes y sencillas de desarrollar, en este tipo de proyectos, que una aplicación móvil. Sin embargo, la necesidad de desarrollar una aplicación móvil surge de que el personal de seguridad (quienes serán los potenciales usuarios) no siempre tienen acceso a una computadora a través de la cual acceder a la pagina web, en cambio siempre tienen un celular a través del cual pueden usar la aplicación móvil.

En este sentido, se desarrollo una aplicación móvil (publicada en el Repositorio de la Aplicación Android) la cual contiene 4 activities[1] (Figura 3.1), a través de los cuales el usuario puede navegar para comandar los aires acondicionados y obtener reportes y advertencias del estado de cada uno de ellos.




(A) Activity de Inicio de Sesión (B) Activity de Control (C) Activity de Monitoreo (D) Activity de Tabla de Datos

FIGURA 3.1. Capturas de pantalla de los activitys desarrollados en la app.

A continuación, se describe el funcionamiento de los activities.

## 3.1. Activity de Inicio de Sesión

El activity de inicio de sesión (Figura 3.1a) es donde el usuario debe autenticarse para poder hacer uso de los servicios de la aplicación.

[1]Los activities de Android son pantallas de la interfaz de usuario de la aplicación de Android. Una aplicación de Android puede contener una o más activities, es decir, una o más pantallas.

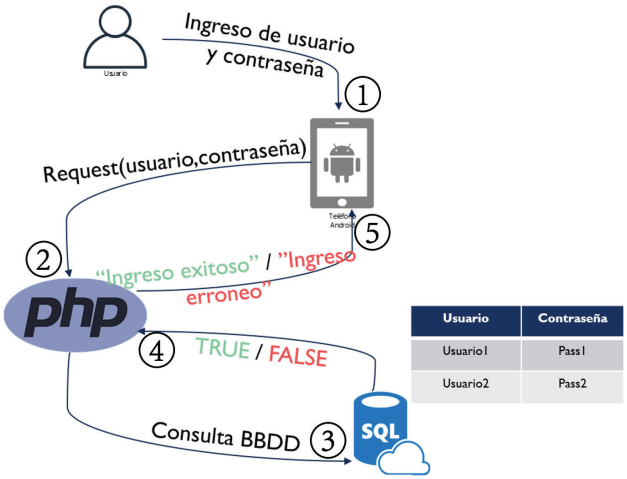



FIGURA 3.2. Diagrama de funcionamiento para el activity de inicio de sesión.

En la Figura 3.2 se muestra el diagrama de funcionamiento donde cada flecha representa un evento distinto y están numeradas secuencialmente. A continuación, se describe cada uno de los eventos.

1. **Ingreso:** el primer evento ocurre cuando el usuario ingresa el usuario y la contraseña y presiona el botón '*Ingresar*'.

2. **Request:** aquí la app hace un request a un servicio PHP y le envia como parámetros el usuario y la contraseña. A continuación se muestra el código desarrollado en PHP:

```
require "DataBase.php";
$db = new DataBase();
if (isset($_GET['username']) && isset($_GET['password'])) {
    if ($db->dbConnect()) {
        if ($db->logIn("login", $_GET['username'], $_GET['password'
    ])) {
            echo "Login Success";
        } else echo "Username or Password wrong";
    } else echo "Error: Database connection";
} else echo "All fields are required";
?>
```

CÓDIGO 3.1. Código de la consulta de usuario y contraseña a la Base de Datos.

3. **Consulta de Base de Datos:** dicho servicio hace una consulta a una base de datos SQL. La misma contiene todas las combinaciones de usuario y contraseña habilitadas para hacer uso de la aplicación.

4. **Respuesta de Base de Datos:** si el usuario y contraseña ingresados por el usuario pertenecen a la base de datos, la misma devolverá 'TRUE'. Caso contrario, devolverá 'FALSE'.

5. **Habilitación o negación de ingreso:** luego el servicio PHP responde el request hecho por la app en el paso 2 y le envía como respuesta 'Ingreso exitoso' o 'Ingreso erróneo' si la respuesta de la base de datos fue 'TRUE' o 'FALSE' respectivamente. En caso de recibir 'Ingreso exitoso' la app permite al usuario hacer uso del resto de la aplicación. Caso contrario, devuelve un mensaje que indica que el usuario y/o contraseña son incorrectos.

## 3.2. Activity de Control

El activity de control (Figura 3.1b) es donde el usuario puede enviar los comandos de encender, apagar, aumentar temperatura o disminuir temperatura. Además, puede visualizar la temperatura de la sala, cuanto tiempo paso desde que se detectó el último movimiento y si el aire acondicionado está encendido o apagado.

En la parte superior de la pantalla de la Figura 3.1b se muestra un menú desplegable en el cual el usuario puede navegar y seleccionar a que aula se desea conectar.

En la Figura 3.3 se muestra el diagrama de funcionamiento donde cada flecha representa un evento distinto y están numeradas secuencialmente. A continuación, se describe cada uno de los eventos.

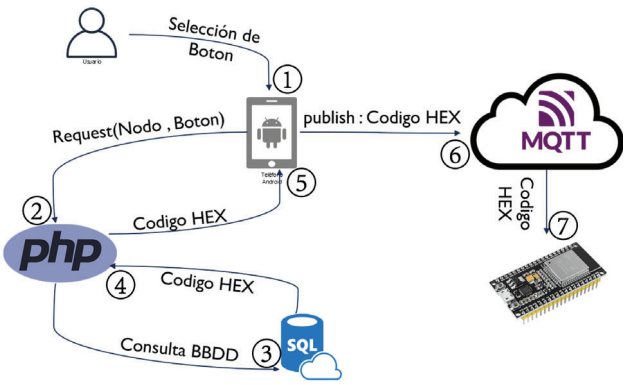



FIGURA 3.3. Diagrama de funcionamiento para el activity de control.

1. **Selección de botón:** este evento ocurre cuando el usuario interactúa con la aplicación para ejecutar alguna de las cuatro instrucciones posibles (Encender, Apagar, Aumentar Temperatura, Disminuir Temperatura).

2. **Request:** cuando el usuario ejecuta alguna instrucción, la app hace un request a un servicio PHP y le envía como parámetros el aula en la que se debe ejecutar la instrucción y el comando. Cuando recibe la respuesta del request, almacena el ID y los códigos hexadecimales en una matriz.

```
1     RequestQueue queue = Volley.newRequestQueue(MainActivity.this);
2     String url = "http://192.168.0.122/android/getProducts.php";
3     System.out.println("URL = " + url);
4     StringRequest stringRequest = new StringRequest(Request.Method.
      GET, url, new Response.Listener<String>() {
5         @Override
6         public void onResponse(String response) {
7             try {
8                 System.out.println(response);
9                 JSONArray array = new JSONArray(response);
10                for(int i = 0; i<array.length(); i++){
11                    JSONObject object = array.getJSONObject(i);
12                    Matriz_BBDD[i][0] = object.getString("ID");
13                    Matriz_BBDD[i][1] = object.getString("IR_ON");
14                    Matriz_BBDD[i][2] = object.getString("IR_OFF");
15                    Matriz_BBDD[i][3] = object.getString("IR_UP");
16                    Matriz_BBDD[i][4] = object.getString("IR_DOWN")
      ;
```

```
17
18                      IDs.add(object.getString("ID").toString());
19
20
21                  }
22
23              }catch (Exception e){
24                  Toast.makeText(getBaseContext(), "Error", Toast.
    LENGTH_SHORT).show();
25              }
26          }
27      }, new Response.ErrorListener() {
28          @Override
29          public void onErrorResponse(VolleyError error) {
30              Toast.makeText(getBaseContext(), "Error BBDD", Toast.
    LENGTH_SHORT).show();
31
32          }
33      });
34
35      queue.add(stringRequest);
36
37  }
```

CÓDIGO 3.2. Código para hacer el request.

3. **Consulta a Base de Datos:** el código desarrollado en PHP hace una consulta a una base de datos SQL. La misma contiene el ID de todas las aulas y los comandos hexadecimales correspondientes a las cuatro posibles instrucciones. Entonces, el servicio PHP hace la consulta y le envía como parámetros la instrucción que desea ejecutar el usuario y el aula en el que lo desea ejecutar.

```
11
12 include('db_connect.php');
13
14 $stmt = $conn->prepare("SELECT ID, IR_ON, IR_OFF, IR_UP, IR_DOWN
    FROM parametros");
15
16 $stmt ->execute();
17 $stmt -> bind_result($ID, $IR_ON, $IR_OFF, $IR_UP, $IR_DOWN);
18
19 $test = array();
20
21 while($stmt ->fetch()){
22
23     $temp = array();
24
25   $temp['ID'] = $ID;
26   $temp['IR_ON'] = $IR_ON;
27   $temp['IR_OFF'] = $IR_OFF;
28   $temp['IR_UP'] = $IR_UP;
29   $temp['IR_DOWN'] = $IR_DOWN;
30
31   array_push($test,$temp);
32   }
33
34   echo json_encode($test);
35
36 ?>
```

CÓDIGO 3.3. Código de la consulta comandos hexadecimales a la Base de Datos.

4. **Respuesta de Base de Datos:** en este evento, la base de datos devuelve al código php el código hexadecimal del comando solicitado en el aula solicitada.
5. **Recepción del comando:** una vez que la base de datos le devuelve al servicio PHP el código hexadecimal, este último lo único que hace es enviarle dicho código a la app en respuesta al request hecho en el paso 2.
6. **Publicación MQTT** una vez obtenido el código hexadecimal, la app publica el mismo en el broker de MQTT.

```
1
2  public void SenderON(View view) {
3      String seleccion = spinner1.getSelectedItem().toString(); //
       Guardo en seleccion el item seleccionado en el spinner
4      String TopicON = ForwardTopic + seleccion;
5      int indice_seleccion = 0;
6      for (int i = 0; i < Matriz_BBDD.length; i++) {
7          for (int j = 0; j < Matriz_BBDD[i].length; j++) {
8              if (Matriz_BBDD[i][j]== seleccion){
9                  indice_seleccion = i;
10             }
11         }
12     }
13     String commandON = Matriz_BBDD[indice_seleccion][1];
14     try{
15         client.subscribe(TopicON,0);
16
17     }catch (MqttException e){
18         e.printStackTrace();
19     }
20     try{
21         MqttMessage message = new MqttMessage((commandON).getBytes
       ());
22         client.publish(TopicON, message);
23     }catch (MqttException e){
24         e.printStackTrace();
25     }
26
27 }
```

CÓDIGO 3.4. Código para la publicación en el broker MQTT del código hexadecimal para encender el aire acondicionado.

7. **Lectura del comando:** cuando el ESP32 detecta que hay un nuevo mensaje en el broker de MQTT, lo lee.
8. **Envió del comando al emisor IR:** una vez leído el comando, el ESP32 procede a enviar el código hexadecimal al emisor IR.

## 3.3. Activity de Monitoreo

Existen dos eventos que claramente pueden indicar un uso indebido del aire acondicionado:

- Que el aire acondicionado este encendido luego de que no se haya detectado movimiento durante horas, ya que esto podría significar que el aire acondicionado está ambientando una sala vacía.

- Que el aire acondicionado este encendido fuera del horario de clases, es decir desde las 22 hs hasta las 8 hs, ya que esto podría indicar que alguna comisión abandono la clase y olvido apagar el aire acondicionado.

El activity de monitoreo (Figura 3.1c) es donde se muestran las aulas en las que suceden alguno de estos dos eventos. Aquí el usuario puede ver todas las salas en las cuales suceden los eventos y la descripción de los mismos. Además, como se ve en la, hay un botón rojo de acceso rápido para apagar directamente el aire acondicionado.

En la Figura 3.4 se muestra el diagrama de funcionamiento. Cada flecha representa un evento distinto. Las mismas están numeradas secuencialmente. A continuación, se describe cada uno de los eventos.

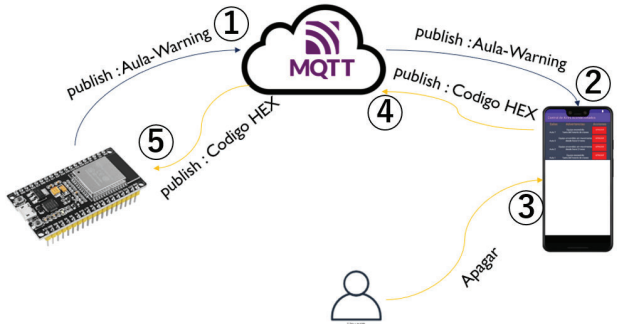



FIGURA 3.4. Diagrama de funcionamiento para el activity de monitoreo.

1. **Publicación del Warining:** cada vez que el ESP32 detecta que el aire acondicionado está encendido, luego de no detectar movimiento por horas o fuera del horario de clases, envía un warning al broker de MQTT donde especifica el aula y la descripción del evento.

2. **Lectura del Warning:** cuando la app detecta el evento, lo lee y lo muestra en una fila de la tabla. A continuación se expone el código implementado para el agregado dinámico de filas en la tabla.

```
1
2  public void init() {
3          Toast.makeText(getBaseContext(), "Mensaje Recibido", Toast.
       LENGTH_SHORT).show();
4          String[] AulasWarnings;
5          System.out.println(Mensaje);
6          String[] Mensaje_split;
7          Mensaje_split = Mensaje.split("-");
8          System.out.println(Mensaje_split[0]);
9          if (Mensaje_split[0].equals("0")) {
10             map.remove(Mensaje_split[1]);
11         } else {
12             map.put(Mensaje_split[1], Mensaje_split[2]);
13         }
14         ArrayList WarningArray = new ArrayList<String>(map.values()
       );
15         ArrayList AulasArray = new ArrayList<String>(map.keySet());
16         System.out.println("Aulas = " + AulasArray);
17         System.out.println("Warnings = " + WarningArray);
18         System.out.println("Size = " + AulasArray.size());
19
20         TableRow tbrow0 = new TableRow(this);
```

```
21          TextView tv0 = new TextView(this);
22          TextView tv1 = new TextView(this);
23          TextView tv2 = new TextView(this);
24          TableLayout stk = (TableLayout) findViewById(R.id.
        table_main);
25
26          stk.removeAllViews();
27
28
29          tv0.setText(" Salas ");
30          tv0.setTypeface(Typeface.DEFAULT_BOLD);
31          tv0.setTextColor(Color.rgb(172, 127, 106));
32          tv0.setTextSize(20);
33          tv0.setGravity(Gravity.CENTER_HORIZONTAL);
34          tv0.setBackgroundResource(R.drawable.rectangulo);
35          tv0.setBackgroundColor(Color.YELLOW);
36          tbrow0.addView(tv0);
37          tv1.setText(" Advertencias ");
38          tv1.setTypeface(Typeface.DEFAULT_BOLD);
39          tv1.setTextColor(Color.rgb(172, 127, 106));
40          tv1.setTextSize(20);
41          tv1.setGravity(Gravity.CENTER);
42          tv1.setBackgroundResource(R.drawable.rectangulo);
43          tv1.setBackgroundColor(Color.YELLOW);
44          tbrow0.addView(tv1);
45          tv2.setText(" Acciones ");
46          tv2.setTypeface(Typeface.DEFAULT_BOLD);
47          tv2.setTextColor(Color.rgb(172, 127, 106));
48          tv2.setTextSize(20);
49          tv2.setGravity(Gravity.CENTER);
50          tv2.setBackgroundResource(R.drawable.rectangulo);
51          tv2.setBackgroundColor(Color.YELLOW);
52          tbrow0.addView(tv2);
53          stk.addView(tbrow0);
54
55
56          for (int i = 0; i <= AulasArray.size(); i++) {
57              flagAux=1;
58              TableRow tbrow = new TableRow(this);
59              TextView t1v = new TextView(this);
60              TableRow separator = new TableRow(this);
61              View line = new View(this);
62              TableRow.LayoutParams separatorLayoutParams = new
        TableRow.LayoutParams(400, 1);
63              separatorLayoutParams.setMargins(20, 0, 0, 0);
64              line.setBackgroundColor(Color.BLUE);
65
66              System.out.println(AulasArray.get(i));
67              System.out.println(WarningArray.get((i)));
68              System.out.println(i);
69
70              t1v.setText("\n" + AulasArray.get(i));
71              t1v.setTextColor(Color.WHITE);
72              t1v.setGravity(Gravity.LEFT);
73              tv0.setBackgroundResource(R.drawable.rectangulo);
74              tbrow.addView(t1v);
75              tbrow.setPadding(0, 10, 0, 10);
76              TextView t2v = new TextView(this);
77              if (WarningArray.get(i).equals("0")) {
78                  t2v.setText("Equipo encendido\n fuera del horario
        de clases");
79              } else {
```

```
80                  t2v.setText("Equipo encendido sin movimiento\ndesde
     hace " + WarningArray.get(i) + " horas");
81              }
82              t2v.setTextColor(Color.WHITE);
83              t2v.setGravity(Gravity.CENTER_HORIZONTAL);
84              tv1.setBackgroundResource(R.drawable.rectangulo);
85              tbrow.addView(t2v);
86              tbrow.setPadding(0, 10, 0, 10);
87              Button t3v = new Button(this);
88              t3v.setOnClickListener(new View.OnClickListener() {
89                  @Override
90                  public void onClick(View v) {
91                      String Topic = ForwardTopic+ AulasArray.get(t3v
    .getId());
92                      Topic = Topic.replace(" ","");
93                      System.out.println(ForwardTopic+ AulasArray.get
    (t3v.getId()));
94
95                      int indice_seleccion = 0;
96                      for (int i = 0; i < Matriz_BBDD.length; i++) {
97                          System.out.println("AulasArray = "+
    AulasArray.get(t3v.getId()));
98                          System.out.println("Matriz = "+ Matriz_BBDD
    [i][0]);
99                          if (Matriz_BBDD[i][0].equals(AulasArray.get
    (t3v.getId()))) {
100                             indice_seleccion = i;
101                         }
102                     }
103                     String commandOFF = Matriz_BBDD[
    indice_seleccion][1];
104                     try{
105                         client.subscribe(Topic,0);
106
107                     }catch (MqttException e){
108                         e.printStackTrace();
109                     }
110                     try{
111                         MqttMessage message = new MqttMessage((
    commandOFF).getBytes());
112                         client.publish(Topic, message);
113                     }catch (MqttException e){
114                         e.printStackTrace();
115                     }
116
117                 }
118             });
119             t3v.setText("Apagar");
120             t3v.setTextColor(Color.WHITE);
121             t3v.setBackgroundColor(Color.RED);
122             t3v.setGravity(Gravity.CENTER_HORIZONTAL);
123             t3v.setId(i);
124             tbrow.addView(t3v);
125             tbrow.setPadding(0, 10, 0, 10);
126             stk.addView(tbrow);
127             tv2.setBackgroundResource(R.drawable.rectangulo);
128             flagAux = 0;
129         }
130
131
```

```
132     }
```

CÓDIGO 3.5. Código para el el agregado dinámico de filas en la tabla.

3. **Apagar:** con dicha información del evento, el usuario puede decidir apagar el aire acondicionado. Para esto lo único que debe hacer es presionar el botón rojo 'Apagar'.
4. **Publicación del comando:** cuando el usuario presiona el botón rojo, la app hace la consulta en la base de datos como se describió en el activity de control y luego envía el código hexadecimal al broker MQTT.
5. **Lectura del comando:** el ESP32 lee el código hexadecimal y lo ejecuta.

## 3.4. Activity de Tabla de Datos

Uno de requerimientos mencionados en el Capítulo 1 fue *"Desplegar un conjunto de servicios de backend que permita la recuperación y almacenamiento de datos en la base de datos "*. En este sentido, el ESP32 publica en una base de datos cada una hora los siguientes datos:

- Aula
- Tiempo desde el último movimiento
- Temperatura
- Estado del aire acondicionado
- Fecha
- Hora

El objetivo del Activity de Tabla de Datos (Figura 3.1a) es que la app obtenga dicha información de la base de datos y la muestre en una tabla.

En la Figura 3.5 se muestra el diagrama de funcionamiento. Cada flecha representa un evento distinto. Las mismas están numeradas secuencialmente. A continuación, se describe cada uno de los eventos:

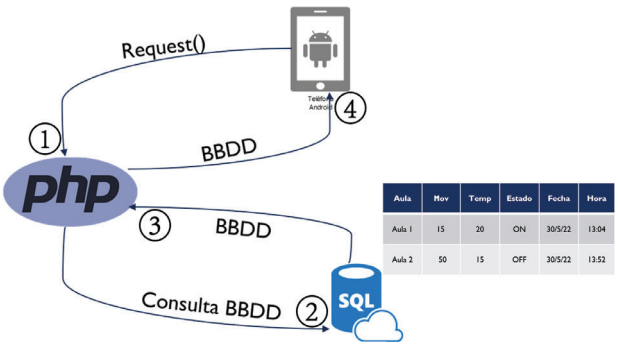



FIGURA 3.5. Diagrama de funcionamiento para el activity de Tabla de Datos.

1. **Request:** la app hace un request a un servicio PHP cuyo código se muestra a continuación:

```
1  private void sendGetRequest(Context context){
2          RequestQueue queue = Volley.newRequestQueue(DataTable.this)
       ;
3          String url = "http://172.16.232.38/Data_Table/getData.php";
4          System.out.println("URL = " + url);
5          StringRequest stringRequest = new StringRequest(Request.
       Method.GET, url, new Response.Listener<String>() {
6              @Override
7              public void onResponse(String response) {
8                  try {
9                      System.out.println(response);
10                     JSONArray array = new JSONArray(response);
11                     for(int i = 0; i<array.length(); i++){
12                         JSONObject object = array.getJSONObject(i);
13                         Matriz_BBDD[i][0] = object.getString("ID");
14                         System.out.println(Matriz_BBDD[i][0]);
15                         Matriz_BBDD[i][1] = object.getString("
       Movimiento");
16                         System.out.println(Matriz_BBDD[i][1]);
17                         Matriz_BBDD[i][2] = object.getString("
       Temperatura");
18                         System.out.println(Matriz_BBDD[i][2]);
19                         if (object.getString("Estado")=="Encendido"
       ){
20                             Matriz_BBDD[i][3] = "ON ";
21                         }else{
22                             Matriz_BBDD[i][3] = "OFF";
23                         }
24                         System.out.println(Matriz_BBDD[i][3]);
25                         Matriz_BBDD[i][4] = object.getString("Fecha
       ");
26                         System.out.println(Matriz_BBDD[i][4]);
27                         Matriz_BBDD[i][5] = object.getString("Hora"
       );
28                         System.out.println(Matriz_BBDD[i][5]);
29                         tabla_lista.add(new ModeloTabla(Matriz_BBDD
       [i][0],Matriz_BBDD[i][1],Matriz_BBDD[i][2],Matriz_BBDD[i][3],
       Matriz_BBDD[i][4],Matriz_BBDD[i][5]));
30                         recycler_view = findViewById(R.id.
       recycler_view);
31                         recycler_view.setHasFixedSize(true);
32                         recycler_view.setLayoutManager(new
       LinearLayoutManager(context));
33                         System.out.println("Funcion");
34                         adapter = new TablaAdapter(context,
       tabla_lista);
35                         recycler_view.setAdapter(adapter);
36                     }
37
38
39                 }catch (Exception e){
40                     Toast.makeText(getBaseContext(), "Error", Toast
       .LENGTH_LONG).show();
41                 }
42             }
43         }, new Response.ErrorListener() {
44             @Override
45             public void onErrorResponse(VolleyError error) {
46                 Toast.makeText(getBaseContext(), "error", Toast.
       LENGTH_LONG).show();
47
48             }
49         });
```

```
50
51
52
53          queue.add(stringRequest);
54
55      }
```

CÓDIGO 3.6. Código para hacer el request.

2. **Consulta a Base de Datos:** el código desarrollado en PHP hace una consulta a una base de datos SQL. La misma contiene la información publicada por el ESP32 descrita anteriormente. A continuación, se exhibe el código desarrollado en PHP:

```
1  <?php
2  $servername = "localhost";
3  $username = "root";
4  $password = "";
5  $dbname = "test";
6
7  // Create connection
8  $conn = new mysqli($servername, $username, $password, $dbname);
9  // Check connection
10 if ($conn->connect_error) {
11   die("Connection failed: " . $conn->connect_error);
12 }
13
14 $sql = "INSERT INTO monitoreo (ID, Movimiento, Temperatura, Estado,
       Fecha,Hora)
15 VALUES ('".$_GET["ID"]."', ".$_GET["Movimiento"].", ".$_GET["
       Temperatura"].", '".$_GET["Estado"]."', '".$_GET["Fecha"]."', '
       ".$_GET["Hora"]."')";
16
17 if ($conn->query($sql) === TRUE) {
18   echo "Escritura Exitosa";
19 } else {
20   echo "Error: " . $sql . "<br>" . $conn->error;
21 }
22
23 $conn->close();
24 ?>
```

CÓDIGO 3.7. Código de la consulta a la Base de Datos.

3. **Respuesta de Base de Datos:** en este evento la base de datos devuelve al código PHP la información solicitada.

4. **Recepción del comando:** una vez que la base de datos le devuelve al código PHP la información, este último lo único que hace es enviarle la misma a la app en respuesta al request hecho en el paso 2.

# Capítulo 4

# Desarrollo del Hardware

## 4.1. Lista de Materiales

| Elemento | Cantidad |
|---|---|
| ESP32 | 1 |
| Sensor de Temperatura LM35 | 1 |
| Sensor de Corriente SCT013 | 1 |
| Emisor IR KY-005 | 1 |
| LED | 4 |
| Conversor AC-DC | 1 |
| Sensor de movimiento HC-SR501 | 1 |

TABLA 4.1. Tabla de componentes

## 4.2. Componentes

En la Figura 4.1 se muestra el diagrama de conexión de todos los dispositivos electrónicos del sistema. El mismo está compuesto por una placa de desarrollo ESP32, la cual contiene toda la inteligencia y la lógica del sistema, y periféricos que se comunican con el ESP32 los cuales se describen a continuación:

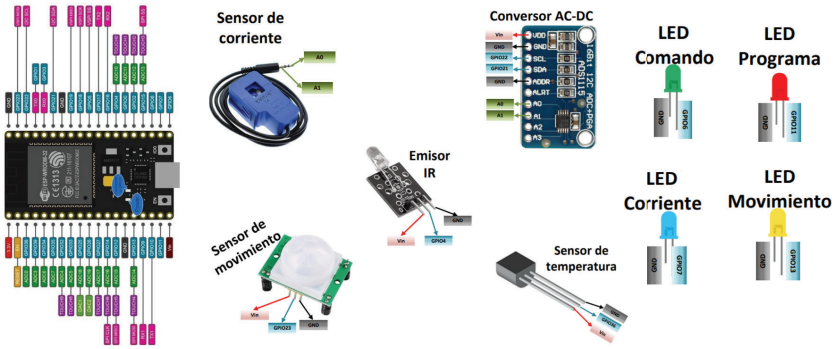



FIGURA 4.1. Diagrama de conexión de todos los dispositivos electrónicos del sistema.

1. **Sensor de Corriente:** el sensor de corriente mide la corriente en el cable de fase de alimentación del aire acondicionado. De esta manera, es posible inferir si el aire acondicionado está encendido o apagado. El sensor utilizado es el SCT-013, el mismo es no invasivo, es decir que no es necesario cortar el cable para medir la corriente, lo cual facilita su instalación.

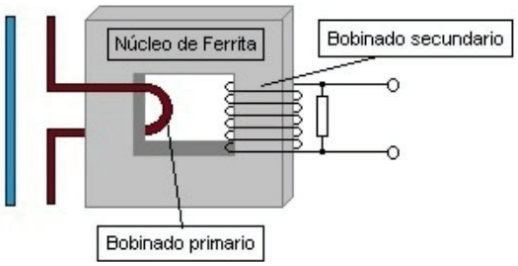



FIGURA 4.2. Imagen ilustrativa del funcionamiento del sensor de movimiento PIR HC-SR501 [7].

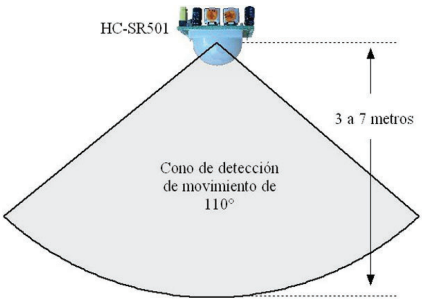
Como se muestra en la Figura 4.2, el SCT-013 funciona como un transformador, donde el cable a medir funciona como el bobinado primario y típicamente se pone con una sola vuelta, luego el bobinado secundario está dentro del sensor y tiene 2000 vueltas. De esta manera, la corriente de salida del sensor es 2000 veces menor a la que circula por el cable.

2. **Conversor AC-DC:** el sensor de corriente entrega una señal AC como salida. La función del conversor ADS1115 [**2**] es acondicionar dicha señal a una comunicación I2C la cual es compatible con el ESP32.

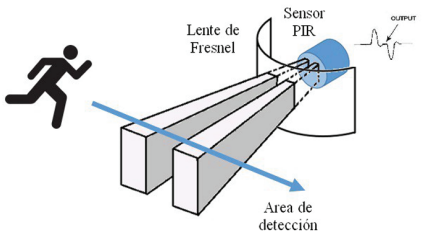
3. **Sensor de Movimiento:** el sensor de movimiento PIR HC-SR501 permite inferir si hay gente en la sala o si la misma está vacía. El mismo permite detectar movimiento hasta 7 metros de distancia y tiene un ángulo de detección de 110° (Figura 4.3a).

   A pesar de ser un sensor de movimiento, el mismo no mide el movimiento directamente, sino que mide ondas infrarrojas. Las moléculas de nuestro cuerpo y de todo lo que nos rodea a la temperatura ambiente tienen sus enlaces atómicos en continuo movimiento. Esto provoca que nuestros cuerpos emitan ondas electromagnéticas (luz) en la franja del infrarrojo y a mayor temperatura, mayor es la radiación emitida. De aquí el nombre de sensores PIR (Passive Infrared).

   El sensor PIR HC-SR501 tiene un lente de Fresnel, el cual es un encapsulado semiesférico hecho de polietileno de alta densidad cuyo objetivo es permitir el paso de la radiación infrarroja en el rango de los 8 y 14 micrones. El lente detecta radiación en un ángulo con apertura de 110° y, adicionalmente, concentra la energía en la superficie de detección del sensor PIR, de esta manera permite una mayor sensibilidad del dispositivo [8].




(A) Cono de detección y alcance del sensor PIR HC-SR501.




(B) Funcionamiento del sensor PIR HC-SR501.

FIGURA 4.3. Sensor de movimiento PIR HC-SR501

Como se muestra en la Figura 4.3b el PIR HC-SR501 cuenta con dos sensores infrarrojos detectores y la señal diferencial entre ambos es lo que activa la alarma de movimiento.

4. **Emisor IR:** el sensor emisor infrarrojo, también llamado diodo emisor infrarrojo, es un módulo KY-005 que convierte la energía eléctrica en luz infrarroja a una frecuencia de 38KHz y una longitud de onda de 940 nm, esto se encuentra fuera del espectro detectable por humanos. El mismo tiene como función enviar los comandos al aire acondicionado. A continuación, se describen sus especificaciones:
   - Voltaje de funcionamiento: 5V
   - Corriente alimentación: 30 a 60 mA DC
   - Consumo de energía: 90 mW
   - Temperatura de funcionamiento: -25°C a 80°C
   - Frecuencia: 38 kHz
   - Peso: 2 g
   - Longitud de onda: 940nm
   - Corriente directa 30 – 60 mA
5. **Sensor de Temperatura:** el sensor de temperatura LM35 [**5**] permite medir la temperatura ambiente en la sala. La serie LM35 son dispositivos de temperatura de circuito integrado de precisión con un voltaje de salida linealmente proporcional a la temperatura en grados centígrados. El dispositivo LM35 tiene una ventaja sobre los sensores de temperatura lineales calibrados en Kelvin, ya que no se requiere que el usuario reste un voltaje constante grande de la salida para obtener una escala centígrada conveniente [9]. La razón por la cual se sensa este parámetro, es para saber si la temperatura es demasiado alta o demasiado baja como para que realmente sea necesario el uso del aire acondicionado. A continuación, se describen sus especificaciones:
   - Voltaje de funcionamiento: 4V a 30V
   - Rango de temperatura: -55°C a 150°C
   - Factor de conversión: Lineal 10 mV/°C
   - Incertidumbre: 0,5°C a 25°C
6. **LED Comando:** el LED de comando es un led que se enciende cada vez que el ESP32 recibe un comando por MQTT
7. **LED Programa:** el LED de programa cambia de estado cada 3 segundos. De esta manera, es posible inferir si el programa se está ejecutando o si por alguna razón se detuvo.
8. **LED Corriente:** el LED de corriente se enciende cuando el equipo de aire acondicionado está encendido y se apaga cuando el mismo está apagado.
9. **LED Movimiento:** el LED de movimiento permanece encendido siempre que se detecte movimiento y permanece apagado mientras no se detecte movimiento.

## 4.3. Producto Final

El producto final esta constituido por un PCB montado dentro de una caja contenedora, la cual fue diseñada a medida para la conexión de la placa con el exterior y el correcto funcionamiento de los sensores.

### 4.3.1. PCB

En la Figura 4.4 se muestra un render del PCB diseñado en KiCAD con todos los componentes de la Figura 4.1 montados como indica el esquematico de la Figura A.1.

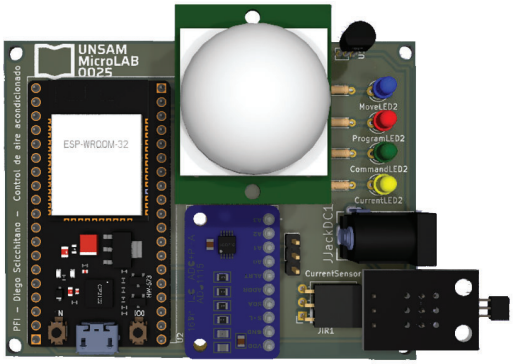



FIGURA 4.4. Render del PCB diseñado en KiCAD con todos los componentes integrados.

Tanto del esquemático, como del PCB se puede observar que el equipo cuenta con un conector '*Jack Power*' a través del cual el equipo se alimenta con 5V. De esta manera, el equipo es '*Plug-and-Play*'[1], es decir que solo es necesario alimentarlo con 5V para que comience a funcionar.

### 4.3.2. Caja Contenedora

Para evitar que la placa quede expuesta al ambiente, la misma fue fijada dentro de una caja impresa en una impresora 3D Creality Ender 5 Plus. En la Figura 4.5 se muestran renders del producto final.

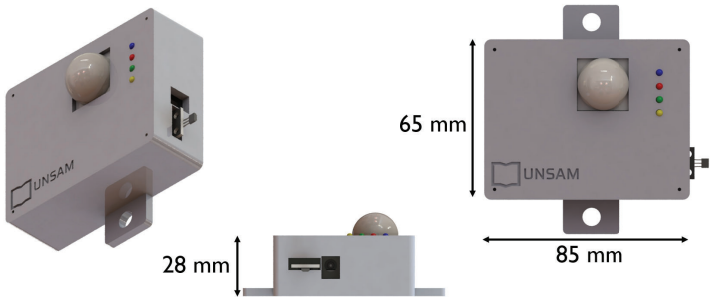



FIGURA 4.5. Render de la caja contenedora diseñada para proteger la componentes electrónicos.

[1]Plug-and-Play (PnP) La frase puede traducirse como "conectar y usar" y refiere a las características de aquellos dispositivos que pueden utilizarse en una computadora sin la necesidad de configuración.

Como se puede observar de dicha figura, el producto puede ser fijado a la pared. Además, los LEDs indicativos quedan a la vista, al igual que el sensor de movimiento y el emisor IR los cuales necesariamente deben están fuera de la caja para su correcto funcionamiento. Por último, el jack de alimentación también está al exterior para poder alimentar el equipo desde una fuente switching.

# Capítulo 5

# Desarrollo del firmware

Dado que para cumplir los requerimientos era necesario leer sensores y enviar comandos a un emisor infrarrojo, el uso de un microcontrolador es necesario. El principal motivo de usar un ESP32, es que el mismo tiene la capacidad de establecer conexión WI-FI, lo cual es necesario para la comunicación MQTT.

Si bien, el objetivo de este capítulo es describir el desarrollo del firmware, en el Repositorio del Proyecto se podrá encontrar el código fuente del mismo.

## 5.1. Flujo de funcionamiento

Como se muestra en la Figura 5.1, el código desarrollado en Arduino se divide en 4 partes, la primera es el setup, el cual se ejecuta por única vez al iniciar el programa. Una vez ejecutado el setup, el loop se ejecuta cíclicamente de manera indefinida. El loop puede ser interrumpido por el callback de comandos o el callback de ID.

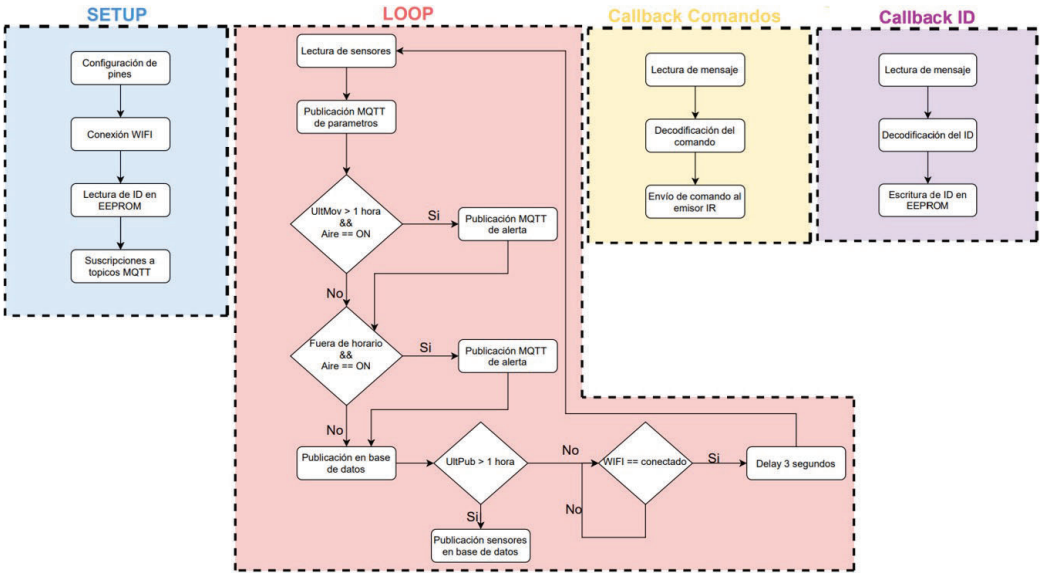



FIGURA 5.1. Diagrama de flujo de alto nivel del firmware.

### 5.1.1. Setup

El siguiente flujo de tareas se ejecuta solo una vez al iniciar el sistema:

- **Configuración de pines:** El primer paso es configurar los pines del ESP32 que se van a utilizar y definir si serán de entrada o salida.

- **Conexión WIFI:** En esta tarea, el ESP32 iniciara la conexión WIFI. El programa no avanzara hasta que no se establezca la conexión, ya que no podría ni reportar los parámetros a la aplicación móvil, ni recibir comandos.
- **Lectura de ID en EEPROM:** El ESP32 lee el ID que tiene cargado en la EEPROM.
- **Suscripción a tópicos MQTT:** El ultimo paso es suscribirse a los tópicos MQTT en los que recibirá mensajes.

### 5.1.2. Loop

El siguiente flujo de tareas se ejecuta de manera indefinida siempre que el sistema este funcionando:

- **Lectura de sensor:** El primer paso es leer los sensores de temperatura, movimiento y corriente.
- **Publicación MQTT de parámetros:** Una vez realizada la lectura de parámetros, se procede a publicar los mismos en el broker MQTT.
- **Primera verificación de uso indebido:** El objetivo de esta tarea es verificar si el aire acondicionado esta encendido mientras no se detecta movimientos desde hace mas de una hora. Si esto ocurre, se publica una alerta en el broker MQTT y se avanza a la siguiente tarea. Si no ocurre, se avanza sin publicar en el broker MQTT.
- **Segunda verificación de uso indebido:** El objetivo de esta tarea es verificar si el aire acondicionado esta encendido fuera del horario de clases. De ser así, se publica una alerta en el broker MQTT y se avanza a la siguiente tarea. Si no ocurre, se avanza sin publicar en el broker MQTT.
- **Publicación en base de datos:** Cada una hora se publica en la base de datos, los valores obtenidos de la lectura de los sensores.
- **Verificación de conexión WIFI:** En esta tarea, se verifica que el ESP32 tenga conexión WIFI. Si no tiene conexión no avanzara a la siguiente tarea hasta restablecer la conexión. Si la conexión esta establecida, se avanza a la siguiente tarea.
- **Delay:** Se esperan 3 segundos antes de comenzar de nuevo el loop.

### 5.1.3. Callback Comandos

El siguiente flujo de tareas se ejecuta cada vez que el usuario decide ejecutar un comando en el aire acondicionado:

- **Lectura de mensaje:** Se hace la lectura del mensaje publicado en el broker de MQTT.
- **Decodificación del mensaje:** Se decodifica el mensaje para convertirlo en un código hexadecimal.
- **Envió de comando al emisor IR:** Se envía la señal señal infrarroja al emisor IR para ejecutar el comando en el aire acondicionado.

### 5.1.4. Callback ID

El siguiente flujo de tareas se ejecuta cada vez que el usuario decide cambiar el ID del sistema:

- **Lectura de mensaje:** Se hace la lectura del mensaje publicado en el broker de MQTT.
- **Decodificación del mensaje:** Se decodifica el mensaje para extraer el ID.
- **Envió de comando al emisor IR:** Se escribe el ID deseado en la memoria EEPROM del ESP32.

## 5.2. Lectura de sensores

A continuación, se describe la programación para la lectura de los sensores descritos en el Capítulo 4:

### 5.2.1. Sensor de Corriente:

El sensor de corriente mide la corriente en el cable de fase de alimentación del aire acondicionado. De esta manera, es posible inferir si el aire acondicionado está encendido o apagado.

Dado que el sensor de corriente entrega una señal AC como salida, es necesario utilizar el conversor ADS1115 para acondicionar dicha señal a una comunicación I2C, la cual es compatible con el ESP32.

Para la lectura del conversor, se utiliza la librería *Adafruit_ADS1015.h*. A continuación se exhibe el código utilizado para la lectura de la corriente:

```
#include <Adafruit_ADS1015.h>

Adafruit_ADS1115 ads;
const float FACTOR = 30; //30A/1V
const float multiplier = 0.0625F;
int CurrentON = 1; //Corriente RMS para la cual se considera encendido
    el equipo

void printMeasure(String prefix, float value, String postfix)
{
 Serial.print(prefix);
 Serial.print(value, 3);
 Serial.println(postfix);
}


float getCorriente()
{
 float voltage;
 float corriente;
 float sum = 0;
 long tiempo = millis();
 int counter = 0;

 while (millis() - tiempo < 1000)
 {
   voltage = ads.readADC_Differential_0_1() * multiplier;
   corriente = voltage * FACTOR;
```

```
28     corriente /= 1000.0;
29
30     sum += sq(corriente);
31     counter = counter + 1;
32    }
33
34  corriente = sqrt(sum / counter);
35  return(corriente);
36 }
37
38 void setup() {
39   Serial.begin(115200);
40
41   ads.setGain(GAIN_TWO);        //  2 .048V  1 bit = 0.0625mV
42   ads.begin();
43
44 }
```

CÓDIGO 5.1. Pseudocódigo de la lectura de corriente.

Como se puede ver en el Código 5.1, el conversor tiene como salida el valor RMS de corriente del sensor. Cuando la lectura de corriente presenta un valor RMS mayor a 1A, se considera que el aire acondicionado está encendido y se enciende el LED de corriente como se muestra en el Código 5.1:

```
1 const int CurrentLedPin = 15;
2 if(currentRMS>=CurrentON){
3   EstadoAire="Encendido";
4   digitalWrite(CurrentLedPin,HIGH);
5 }else{
6   EstadoAire="Apagado";
7   digitalWrite(CurrentLedPin,LOW);
8 }
```

CÓDIGO 5.2. Pseudocódigo para determinar el estado del aire acondicionado.

### 5.2.2. Sensor de Movimiento

El sensor de movimiento permite inferir si hay gente en la sala o si la misma está vacía.

El hecho de que no se detecte movimiento en un determinado momento no significa necesariamente que existe un uso indebido del aire acondicionado. Es decir, si no se detecta movimiento, puede significar que la última comisión se fue y olvido apagar el aire acondicionado, lo cual sería un uso indebido, o bien podría significar que están en recreo. En este sentido, es más útil saber hace cuanto tiempo no se detecta movimiento, que simplemente saber si hay o no movimiento en un determinado momento.

De esta manera, si no se detecta movimiento hace 3 horas, es altamente probable que haya un consumo innecesario, pero si no se detecta movimiento hace 3 minutos, existe la posibilidad de que la comisión esté en recreo y vuelvan al aula en unos minutos.

A continuación, se exhibe el código para obtener el tiempo transcurrido desde la última vez que se detectó movimiento y para encender el LED de movimiento cada vez que se detecta movimiento:

```
unsigned long lastMove = 0;
unsigned long TimeMove = 0;

int MoveState = 0;
const int MovePin =  33;
const int MoveledPin =  13;       // the number of the LED pin


void setup() {
  pinMode(MoveledPin, OUTPUT);
  pinMode(MovePin, INPUT);

  Serial.begin(115200);
}
void loop() {
      MoveState = digitalRead(MovePin);
      if (MoveState == HIGH) {
        // turn LED on:
        digitalWrite(MoveledPin, HIGH);
        lastMove = now;
      } else {
        // turn LED off:
        digitalWrite(MoveledPin, LOW);
      }
    now = millis();
    if (now - lastMsg > 3000) {
        TimeMove = now-lastMove;
    }
}
```

CÓDIGO 5.3. Pseudocódigo para obtener el tiempo transcurrido desde la última vez que se detectó movimiento.

### 5.2.3. Sensor de Temperatura

El sensor de temperatura LM35 [5] permite medir la temperatura ambiente en la sala. Es un parámetro que ayuda a saber si la temperatura es demasiado alta o demasiado baja como para que realmente sea necesario el uso del aire acondicionado.

En el Código 5.4 se exhibe la lectura del sensor de temperatura.

```

float Temp; // Variable para almacenar el valor obtenido del sensor (0 a
     1023)
int pinLM35 = 36; // Variable del pin de entrada del sensor (A0)

void loop() {
    Temp = analogRead(pinLM35);
    Temp = (Temp/ 4095) * 500; //El 4095 = 2^12 ya que 12 bits es la
    resolucion del conversor ADC del ESP32 y luego el 500 es por 5V
}
```

CÓDIGO 5.4. Pseudocódigo para medir temperatura.

## 5.3. Emisor IR

El emisor tiene como función enviar los comandos al aire acondicionado. Los comandos se reciben a través de MQTT desde la aplicación móvil, el ESP32 lo

único que hace es enviar dichos comandos al emisor IR. Por este motivo, el envío del comando se ejecuta en el callback de MQTT como se muestra en el Código 5.5

```
void callback(char* topic, byte* payload, unsigned int length) {
  digitalWrite(CommandLedPin, HIGH);
  /*delay(100);
  digitalWrite(CommandLedPin, LOW);*/
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");

  int i = 0;
  for (i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
    TopicESP32[i] = (char)payload[i];
  }
  Serial.println();
  for (int j = i; j < 15; j++) {
    TopicESP32[j] = NULL;
  }

  String msgMQTT = (char *)payload;

   msgMQTT = msgMQTT.substring(0, length);
   Serial.print("Payload:");
   Serial.println(msgMQTT);
   codeMQTT=msgMQTT.toInt();
   Serial.println(codeMQTT,HEX);
   irsend.sendLG(codeMQTT, 28);


}
```

CÓDIGO 5.5. Pseudocódigo para enviar comandos al emisor IR.

## 5.4. Comunicación MQTT

La comunicación MQTT es similar al antiguo método de comunicación mediante cartas por correo. Las cartas que se envían por correo tienen un mensaje y la dirección del domicilio al cual se quiere enviar dicho mensaje. En la comunicación MQTT, además de especificar el mensaje, también se debe especificar el tópico el cual, al igual que la dirección del domicilio, cumple la función de filtro para discriminar quien debe recibir el mensaje. Es decir, los mensajes serán recibidos solo por aquellos dispositivos que estén suscritos a ese tópico.

En el presente trabajo, cada ESP32 tiene un ID asignado. De esta manera, cada ESP32 estará suscrito a los tópicos que le correspondan según el ID que tenga asignado. Por ejemplo, para recibir comandos, el ESP32 del AulaX estará suscrito al tópico denominado '*Comandos/AulaX*', entonces solo recibirá los comandos que se le envíen a través de dicho tópico.

En el código, la comunicación MQTT se usa para la recepción de comandos, el envío de los parámetros, el envío de warnings ante eventos no deseados y para cambiar el ID del sistema. Para cada una de estas funcionalidades, se utiliza un tópico distinto y las mismas se describen a continuación.

### 5.4.1. Recepción de comandos

Para la recepción de comandos es necesario conocer el tópico a través del cual recibimos los mensajes. El tópico tiene una dirección raíz (*BackwardTopic* en el código) a través de la cual se envían todos los códigos hexadecimales, pero a esa dirección raíz se le concatena el ID del ESP32 al cual se le envía dicho comando (línea 5 del Código 5.6).

```
1 #include <PubSubClient.h>
2 const char* BackwardTopic = "Recepcion/de/comandos/";
3
4 strcpy(SubscribeTopic,BackwardTopic);
5 strcat(SubscribeTopic,(const char*) TopicESP32);
6
7 client.subscribe(SubscribeTopic);
```

CÓDIGO 5.6. Pseudocódigo para la recepción de comandos.

### 5.4.2. Envió de parámetros

Cada 15 segundos el ESP32 envía por MQTT el estado de los tres sensores (temperatura, movimiento y corriente).

En el caso del movimiento, el sistema reporta el tiempo transcurrido desde la ultima vez que se detecto movimiento, y en el caso de la corriente envía *'ON'* si la corriente es mayor a $1A$, caso contrario envía *'OFF'*.

Al igual que en la recepción de comandos, todos los ESP32 tienen la misma dirección raíz a través de la cual envían los parámetros, pero a esa dirección raíz se le concatena el ID del ESP32 que indica de que aula son esos parámetros (linea 5 del Código 5.7).

```
1 #include <PubSubClient.h>
2 const char* ForwardTopic = "ESP32/Android/";
3
4 strcpy(PublishTopic,ForwardTopic);
5 strcat(PublishTopic,(const char*) TopicESP32);
6
7 client.publish(PublishTopic, (char*) TimeMoveString.c_str());
```

CÓDIGO 5.7. Pseudocódigo para la recepción de comandos.

### 5.4.3. Envió de warnings

Como se menciono anteriormente, hay dos eventos que indican una alta probabilidad de uso indebido de los aires acondicionados:

- Que el aire acondicionado este encendido luego de que no se haya detectado movimiento durante horas, ya que esto podría significar que el aire acondicionado esta ambientando una sala vacía.
- Que el aire acondicionado este encendido fuera del horario de clases, es decir desde las 22hs hasta las 8hs, ya que esto podría indicar que alguna comisión abandono la clase y olvido apagar el aire acondicionado.

El ESP32 esta programado para capturar estos eventos y enviar un warning al broker de MQTT con el tópico raíz para el envió de warnings concatenado con el ID del ESP32. A continuación se muestra el código:

```
1
2  if((TimeMove/1000 > 3600) && (EstadoAire == "Encendido")){    //3600sec =
       1h
3    WarningState = "1 - ";
4    WarningState.concat(String(TopicESP32));
5    WarningState.concat(" - ");
6    WarningState.concat(TimeMoveHora);
7    client.publish(WarningTopic,(char*) WarningState.c_str()); //Cuando no
       se detecta movimiento por x tiempo y el aire esta encendido, se
      envia separado por un guion, el nodo que tiene el error y codigo de
      error que debe interpretar la APP de android
8    Serial.println("Warning1 Enviado");
9  }else{
10   Serial.println(String(rtc.getTime("%H").toInt()));
11   if(((rtc.getTime("%H").toInt()>=21) || (rtc.getTime("%H").toInt()<=6))
       && (EstadoAire == "Encendido")){
12     WarningState = "1 - ";
13     WarningState.concat(String(TopicESP32));
14     WarningState.concat(" -0");
15     client.publish(WarningTopic,(char*) WarningState.c_str()); //Este
      warning se dispara cuando el aire esta encendido fuera del horario
      de clases, se envia separado por un guion, el nodo que tiene el
      error y codigo de error que debe interpretar la APP de android
16     Serial.println("Warning2 Enviado");
17   }else{
18       if(WarningState == "1 - "){
19         WarningState = "0 - ";
20         WarningState.concat(String(TopicESP32));
21         WarningState.concat(" - 0");
22         client.publish(WarningTopic,(char*) String(TopicESP32).c_str());
      //Este warning se dispara cuando el aire esta encendido fuera del
      horario de clases, se envia separado por un guion, el nodo que tiene
       el error y codigo de error que debe interpretar la APP de android
23         Serial.println("Warning0 Enviado");
24       }
25   }
26 }
```

CÓDIGO 5.8. Pseudocódigo para la recepción de comandos.

### 5.4.4. Cambio de ID

El ID no es solo un nombre que se le asigna al nodo. Como se vio en las 3 comunicaciones MQTT anteriores, el ID del nodo se utiliza para identificar el nodo del cual provienen los parámetros o los warnings o bien para que la app especifique a que ESP32 le envía los comandos hexadecimales.

Por este motivo, si por alguna razón se decide cambiar el equipo del *'Aula X'* al *'Aula Y'*, sería necesario cambiar el ID del aula anterior por el ID del aula nueva en la cual estará el equipo, para especificar correctamente de donde viene la información y porque además, en la nueva aula, tal vez el aire acondicionado tenga otro protocolo de comunicación, y los comandos hexadecimales están almacenados por aula en una base de datos, entonces si no se actualiza el ID del ESP32 el mismo recibirá los códigos hexadecimales que corresponden al aire acondicionado del aula anterior.

En este sentido, el ID del ESP32 es configurable a través de MQTT. A continuación, se muestra el código:

```
1 #include <PubSubClient.h>
```

```
#include "EEPROM.h"

const char* ParamTopic = "UNSAM/Aires/Parametros/";
int addr=0;

void callback(char* topic, byte* payload, unsigned int length) {
  digitalWrite(CommandLedPin, HIGH);
  /*delay(100);
  digitalWrite(CommandLedPin, LOW);*/
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");

  int i = 0;
  for (i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
    TopicESP32[i] = (char)payload[i];
  }
  Serial.println();
  for (int j = i; j < 15; j++) {
    TopicESP32[j] = NULL;
  }

  String msgMQTT = (char *)payload;

  if (strcmp(topic,ParamTopicFinal) == 0){
    Serial.print("TopicESP32 : ");
    Serial.println(String(TopicESP32));
    for (int k = 0; k < EEPROM_SIZE; k++) {
      EEPROM.write(addr, TopicESP32[k]);
      addr += 1;
    }
   EEPROM.commit();
    addr=0;
  }


}


strcpy(SubscribeTopic,BackwardTopic);
strcat(SubscribeTopic,(const char*) TopicESP32);

client.subscribe(SubscribeTopic);
```

CÓDIGO 5.9. Pseudocódigo para el cambio de ID.

Como se puede observar en la línea 31 del Código 5.9, el ID del ESP32 se escribe en la EEPROM del microcontrolador, ya que es una variable que cuando cambia su valor debe mantener los cambios incluso cuando se reinicia el equipo.

Por este motivo, el código, cada vez que inicia, lee la EEPROM para almacenar el ID del ESP32 en la variable TopicESP32 como se muestra a continuación:

```
#include "EEPROM.h"

int addr=0;


void setup() {

  EEPROM.begin(EEPROM_SIZE);
  // reading byte-by-byte from EEPROM
```

```
  for (int i = 0; i < EEPROM_SIZE; i++) {
    byte readValue = EEPROM.read(i);

    if (readValue == 0) {
        break;
    }

    char readValueChar = char(readValue);
    Serial.print(readValueChar);
    TopicESP32[i] = readValueChar;
  }
}
```

CÓDIGO 5.10. Pseudocódigo para la lectura de la EEPROM.

## 5.5. Publicación en Base de Datos

Con el objetivo de almacenar un registro histórico de todos los parámetros del sistema, el ESP32 publica dichos parámetros en una base de datos cada una hora.

En este sentido, el ESP32 hace un request a un servicio PHP cada una hora enviándole el valor de los parámetros:

```
#include <HTTPClient.h>
String host = "http://localhost/write_data/write_data.php?";
void write_BBDD(int Mov,int Temp,String Estado){
  HTTPClient http;
  Serial.println(host + "ID=" + String(TopicESP32) + "&Movimiento=" +
    Mov + "&Temperatura=" + Temp + "&Estado=" + Estado + "&Fecha=" + rtc
    .getDay() + "/" + rtc.getMonth() + "/" + rtc.getYear() + "&Hora=" +
    rtc.getTime());

  http.begin(host + "ID=" + String(TopicESP32) + "&Movimiento=" + Mov +
    "&Temperatura=" + Temp + "&Estado=" + Estado + "&Fecha=" + rtc.
    getDay() + "/" + rtc.getMonth() + "/" + rtc.getYear() + "&Hora=" +
    rtc.getTime());
  int httpCode = http.GET();

  if (httpCode > 0) {

      String payload = http.getString();
      Serial.println(httpCode);
      Serial.println(payload);
    }

  else {
    Serial.println("Error on HTTP request");
  }

  http.end();
}
```

CÓDIGO 5.11. Pseudocódigo para el request de escritura en base de datos.

A continuación se muestra el servicio PHP para la publicación de los parámetros del sistema

```
<?php
$servername = "localhost";
$username = "root";
```

```
$password = "";
$dbname = "test";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

$sql = "INSERT INTO monitoreo (ID, Movimiento, Temperatura, Estado,Fecha
    ,Hora)
VALUES ('".$_GET["ID"]."', ".$_GET["Movimiento"].", ".$_GET["Temperatura
    "].", '".$_GET["Estado"]."', '".$_GET["Fecha"]."', '".$_GET["Hora"].
    "')";

if ($conn->query($sql) === TRUE) {
  echo "Escritura Exitosa";
} else {
  echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>
```

CÓDIGO 5.12. Pseudocódigo para el request de escritura en base de datos.

# Capítulo 6

# Conclusiones

Hay 3 factores importantes a la hora de describir un proyecto a nivel global, los cuales se pueden representar a través del triangulo de la gestión de proyectos que se muestra en la Figura 6.1.


FIGURA 6.1. Triangulo de la gestión de proyectos.

- Todo proyecto está sujeto a un **plazo**, a una duración. La mayoría de los proyectos tienen una fecha límite para la que el proyecto deberá estar concluido. Además, el proyecto posiblemente disponga de una serie de hitos intermedios (o puntos intermedios de control) por cumplir en cuanto a fechas.
- El **objetivo** es el conjunto de características, funciones, especificaciones y calidad final que tiene el producto una vez terminado.
- Los **costos** son todos aquellos gastos en los que se incurre para realizar una tarea, un trabajo o un proyecto determinado. Las dos principales clases de costos se conocen como costos directos e indirectos.

## 6.1. Conclusiones generales

En esta sección se resaltan los resultados de los 3 factores contemplados en el triangulo de la gestión de proyectos, con el objetivo de evaluar los objetivos, plazos y costos incurridos a lo largo del proyecto.

A continuación se describen cada uno de estos temas y el grado de cumplimiento de los mismos.

### 6.1.1. Grado de cumplimiento de los requerimientos

En la Figura 6.2 se muestra una imagen de la caja contenedora con el PCB y todos los componentes soldados. De la misma, se puede observar que la caja contenedora está orientada de manera tal que el emisor IR apunta en la dirección en la que se encuentra el aire acondicionado.

FIGURA 6.2. Imagen del set-up para la verificación del correcto funcionamiento del sistema.

Para medir la corriente, se colocó el sensor de corriente en el cable de fase del aire acondicionado como se muestra en la Figura 6.3. De la misma, se puede observar que no fue necesario cortar el cable ni hacer ningún tipo de cambio sobre el mismo, ya que el sensor es no invasivo.

FIGURA 6.3. Instalación de sensor de corriente.

Los servicios PHP y las bases de datos están corriendo en una notebook, la cual simula ser el servidor.

De esta manera, como se exhibe en el vídeo demostrativo de funcionalidad del sistema, se ha podido verificar el correcto funcionamiento de todas las funcionalidades del sistema, las cuales se enumeran a continuación:

1. **Medición de temperatura:** se comparó la temperatura que reportaba el sistema con la temperatura que entregaba una termocupla para verificar que la medición sea correcta.
2. **Detección de movimiento:** se verificó que el LED de movimiento se encienda solo cuando hay movimiento.

3. **Medición de corriente:** se verificó que el LED de corriente se encienda cuando el aire acondicionado está encendido y se apague cuando el aire acondicionado está apagado.
4. **Recepción de comandos:** se verificó que cada vez que se envíe un comando desde la aplicación Android, dicho comando sea correctamente recibido por el ESP32.
5. **Ejecución de comandos:** se verificó que cada vez que se recibe un comando, el mismo se ejecute correctamente en el aire acondicionado. Es decir, que si el usuario envía el comando 'ON', el aire acondicionado se debe encender.
6. **Inicio de sesión:** se verificó que la app solo habilite el acceso a los usuarios que estén en la base de datos.
7. **Lectura de sensores en la app:** se verificó que los valores de los sensores que muestra la app sean los mismos que recibe el ESP32.
8. **Escritura en la BBDD:** se verificó que, tanto el ESP32 como la aplicación Android, escriban en la base de datos correctamente.
9. **Lectura de la BBDD:** se verificó que la app lea correctamente la base de datos.
10. **Advertencias de eventos:** se verificó que la app muestre las advertencias cada vez que en aire acondicionado está encendido fuera del horario de clases o que permanezca encendido luego de que haya pasado más de una hora desde que se detectó el último movimiento.
11. **Comunicación MQTT:** se verificó que la comunicación MQTT entre el ESP32 y la aplicación Android funcione correctamente.

En la Tabla 6.1 se muestra un resumen de todas las funcionalidades testeadas y su resultado. De la misma, se puede observar que el sistema funciona correctamente.

| ID | Funcionalidad | Resultado |
|---|---|---|
| 1 | Medición de temperatura | OK |
| 2 | Detección de movimiento | OK |
| 3 | Medición de Corriente | OK |
| 4 | Recepción de comandos | OK |
| 5 | Ejecución de comandos | OK |
| 6 | Inicio de sesión | OK |
| 7 | Lectura de sensores en la app | OK |
| 8 | Escritura en la BBDD | OK |
| 9 | Lectura de la BBDD | OK |
| 10 | Advertencias de eventos | OK |
| 11 | Comunicación MQTT | OK |
| 12 | Cambio de ID del ESP32 | OK |

TABLA 6.1. Tabla de Requerimientos

### 6.1.2. Riesgos

Antes de iniciar el proyecto, se hizo un análisis de riesgos enumerando todos los posibles factores que puedan eventualmente generar un costo adicional, un

retraso en la planificación del proyecto y/o un efecto sobre el desempeño del sistema. Dichos factores se describen en la Figura 6.4.

| Riesgos | Justificación | Severidad | Ocurrencia | RPN | Mitigacion de riesgos | Severidad* | Ocurrencia* | RPN* |
|---|---|---|---|---|---|---|---|---|
| Restricciones de acceso a la universidad. | Si bien el proyecto esta planeado para ser desarrollado desde mi domicilio propio, la etapa de testeo final es necesario hacerla en la universidad. | 6 | 7 | 42 | Se compraran todos los componentes necesarios para llevar a cabo la etapa de desarrollo en el domicilio propio. | 3 | 3 | 9 |
| La universidad no proveerá el servicio de fabricación de PCBs. | Si bien hay otros fabricantes de PCB, los mismos son significativamente mas costosos, mientras que la universidad las fabrica gratuitamente para estudiantes. | 9 | 4 | 36 | Hay otras alternativas de proveedores para la fabricación de PCB. | 8 | 2 | 16 |
| El borker de MQTT deje de ser gratuito. | En principio, se utilizara el broker HiveMQ, el cual es gratuito, pero el mismo podria dejar de serlo. | 7 | 4 | 28 | Se desarrollara un broker de MQTT propio o se pagara la licencia de un broker no gratuito. | 6 | 2 | 12 |
| Dependecia del conversor ADS1115. | El conversor ADS1115, convierte la señal AC del sensor de corriente a I2C para que pueda ser recibida por el ESP32. | 6 | 4 | 24 | Se buscaron alternativas en el exterior que son mas costosas. | 5 | 3 | 15 |
| Problemas de resolución en áreas sin experiencia. | En este proyecto se desarrollaran herramientas en las cuales no se tiene experiencia, como por ejemplo el desarrollo de aplicaciones Android, desarrollo de base de datos, diseño 3D, entre otros. | 8 | 7 | 56 | Antes del comienzo del proyecto, se hizo una evaluación de la factibilidad del proyecto y se consulto a otros profesionales que tenian experiencia. | 5 | 5 | 25 |

FIGURA 6.4. Análisis de riesgos realizado antes de comenzar el proyecto

A continuación se detallan los resultados y los impactos que produjeron cada uno de los riegos:

1. **Restricciones de acceso a la universidad:** desde febrero de 2022 la UNSAM dicta las clases presencialmente, por lo cual no ha presentado ningún impacto perjudicial para el proyecto.

2. **La Universidad proveerá el servicio de fabricación de PCB:** la UNSAM cuenta con un servicio de fabricación de PCB gratuito para alumnos en el MicroLab. Sin embargo, desde hace algunos meses el CNC de la UNSAM no funciona, por este motivo no es posible fabricar el PCB en el MicroLab.

   Se buscaron distintos proveedores y se fabrico el PCB con un fabricante externo a la universidad. Esto genero un impacto significativo para el proyecto en cuanto a costos y tiempos.

3. **El broker de MQTT deje de ser gratuito:** por el momento, el broker HiveMQ de MQTT sigue siendo gratuito.

4. **Dependencia del conversor ADS1115:** por el momento, Adafruit aun fabrica los conversores ADS1115.

5. **Problemas de resolución en áreas de no competencia:** si bien este riesgo fue al que se le asignó el mayor RPN, no presento ningún riesgo durante el desarrollo del proyecto y todas las áreas en las que se consideraba que no se tenía experiencia, fueron resueltas satisfactoriamente.

### 6.1.3. Planificación

Antes de iniciar el proyecto, se presentó un diagrama de Gantt, en el cual se estimó que el proyecto se finalizaría el 30 de junio del 2022. Todas las tareas fueron completadas en tiempo y forma, excepto, como se mencionó anteriormente, la fabricación del PCB, el cual se fabrico 2 semanas después de lo previsto. De esta

manera, las tareas de desarrollo se finalizaron un mes después, el 30 de julio del 2022.

### 6.1.4. Costos

En la Figura 6.5 se muestra la tabla de costos realizada antes de comenzar el proyecto.

| Item | Cantidad2 | Costo Unitario | Costo Final |
|---|---|---|---|
| Sensor de corriente | 2 | $ 1,450.00 | $ 2,900.00 |
| ESP32 | 2 | $ 1,100.00 | $ 2,200.00 |
| Sensor de temperatura | 2 | $ 140.00 | $ 280.00 |
| Placa PCB virgen | 2 | $ 150.00 | $ 300.00 |
| Emisor IR | 2 | $ 200.00 | $ 400.00 |
| Tira de pines hembra | 2 | $ 100.00 | $ 200.00 |
| Tira de pines macho | 2 | $ 100.00 | $ 200.00 |
| Protoboard | 2 | $ 350.00 | $ 700.00 |
| LEDs | 20 | $ 10.00 | $ 200.00 |
| Juego de cables | 3 | $ 100.00 | $ 300.00 |
| Sensor de movimiento | 2 | $ 320.00 | $ 640.00 |
| | | Costo Final | $ 8,320.00 |

FIGURA 6.5. Análisis de costos directos realizado antes de comenzar el proyecto

Los materiales utilizados durante el proyecto fueron los planificados en la Figura 6.5, por lo cual no hubo materiales que se hayan tenido que agregar a los costos previstos.

Sin embargo, antes de comenzar el proyecto, se planifico fabricar el PCB de manera gratuita en la UNSAM. Como se describió anteriormente, esto no fue posible. Entonces, para fabricar el PCB se tuvo que incurrir en un gasto adicional de $10.000, convirtiendo el costo final planificado de $8.320 en un costo final de $18.320, lo cual significo un incremento de un 120 %.

De todas formas, el gasto adicional se debe a que se prefirió fabricar el PCB en un fabricante externo, aunque esto significara un costo adicional, para poder avanzar con el desarrollo en vez de esperar a que la UNSAM vuelva a fabricar PCBs, pero una vez que la UNSAM vuelva a fabricar PCBs, el costo del producto volverá a ser el de la Figura 6.5.

## 6.2. Trabajo Futuro

### 6.2.1. Cambio de ID del ESP32 desde la app

Como se dijo anteriormente, el ID del nodo se utiliza para identificar el nodo del cual provienen los parámetros o los warnings o bien para que la app especifique a que ESP32 le envía los comandos hexadecimales.

En este sentido, si por alguna razón se decide cambiar el equipo del Aula A al Aula B, sería necesario cambiar el ID del aula anterior ('Aula A') por el ID del aula nueva ('Aula B') en la cual estará el equipo para especificar correctamente de donde viene la información y porque además, en la nueva aula, tal vez el aire acondicionado tenga otro protocolo de comunicación, y los comandos hexadecimales están almacenados por aula en una base de datos, entonces si no se actualiza el ID del ESP32, el mismo recibirá los códigos hexadecimales que corresponden al aire acondicionado del aula anterior.

El cambio de ID se hace a través de MQTT, Sin embargo, no hay ninguna interfaz gráfica desarrollada para hacerlo. Actualmente, el ID se cambia enviando el mensaje al ESP32 desde el broker de MQTT. Una mejora funcional para el sistema sería desarrollar una interfaz gráfica en la app que permita cambiar el ID de una manera más visual y desde la app.

### 6.2.2. Capacidad de utilizar la app desde redes externas

Una de las principales prestaciones que ofrecen los sistemas de demótica es la posibilidad de controlar y obtener información de cosas que no están en nuestro entorno.

Si bien la comunicación entre la app y el ESP32 es a través de MQTT, es decir que ambos pueden comunicarse sin la necesidad de estar en la misma red, la app obtiene los comandos hexadecimales de una base de datos e incluso para iniciar sesión la app debe hacer una consulta a una base de datos.

Para acceder a estas bases de datos es necesario estar en la misma red. Esto implica la limitación de controlar y obtener información de los aires acondicionados desde cualquier lugar que no esté en la red de la universidad. Para resolver esta limitación se proponen dos alternativas.

- La primera alternativa es que cada vez que la app inicia, descargue las bases de datos y la guarde localmente. De esta manera, cada vez que se inicie la app actualizara la base de datos. Entonces, la app siempre intentará acceder a la base de datos remota, pero si no puede acceder a esa base de datos por alguna razón, utilizará la base de datos local. De esta manera, no sería necesario estar en la misma red para enviar comandos.

  Sin embargo, existen el riesgo de que si hay un cambio en las bases de datos, los comandos hexadecimales que se envíen no estén actualizados y las instrucciones enviadas por el usuario no se reflejen efectivamente en el aire acondicionado, o lo que sería peor, que un usuario sea eliminado de la base de datos, pero que el mismo pueda ingresar a la app porque tiene una versión anterior de la base de datos.

- La segundo alternativa sería hacer que todas las bases de datos sean remotas en vez de locales. De esta manera, las mismas podrían ser accedidas desde otras redes externas a la facultad.

  Cabe destacar que esta alternativa no tendría el problema mencionado en la primera ya que las bases de datos estarían actualizadas siempre y cuando haya acceso a internet. Pero si la app no tiene acceso a internet, no se podría iniciar sesión y hacer uso de la misma.

### 6.2.3. Actualización inalámbrica de firmware

Es común que las necesidades de los clientes cambien, por ejemplo, que además de querer controlar el aire acondicionado con el emisor infrarrojo, se quiera controlar el encendido o apagado de estufas con un relé. En este caso sería necesario agregar funcionalidades al sistema y agregar líneas de código. Luego, alguna persona debería ir al lugar en el cual está instalado el equipo y actualizar el firmware del ESP32.

En este sentido, sería una mejor opción poder actualizar el firmware de manera inalámbrica ya que todos estos costos pueden eliminarse haciendo que los dispositivos actualicen '*silenciosamente*' su firmware en segundo plano, o durante las horas de inactividad operativa, como la mitad de la noche

En la actualidad existen muchas soluciones entre las que elegir. Por ejemplo, OTA[1] es un método de actualización de firmware, donde tanto la aplicación como el SBL[2] ejecutan el proceso de actualización sin la intervención de personal técnico. En algunos métodos, el firmware actualizado sobrescribe el firmware anterior. Si el firmware recibido es defectuoso, es posible que el sistema deje de funcionar.

### 6.2.4. Desarrollo de un broker propio

Para el desarrollo del proyecto se utilizó un cliente de websocket gratuito. Este método es poco seguro, ya que cualquier persona que conozca el tópico a través del cual se comunican los dispositivos del sistema, puede interferir en la comunicación de los mismos. En este sentido, sería más seguro desarrollar un broker MQTT con usuario y contraseña o utilizar un broker comercial para tener mayor seguridad.

### 6.2.5. Modo Recepción

Si se desea instalar el equipo en un aula, es necesario cargar en la base de datos los comandos hexadecimales manualmente. En este sentido, sería más cómodo para el usuario, tener una funcionalidad en el ESP32, en la cual el mismo tenga un receptor infrarrojo que reciba la señal de un determinado control remoto y decodifique la señal a un código hexadecimal, de esta manera recibiría los comandos y los publicaría en la base de datos automáticamente con el ID que tiene guardado en la memoria flash.

[1] La sigla en inglés 'OTA' se refiere a over-the-air o *'por el aire'* en español

[2] El Secondary Boot Loader (SBL) o cargador de arranque secundario se utiliza para reprogramar y/o actualizar el software.

# Apéndice A

# Esquemático

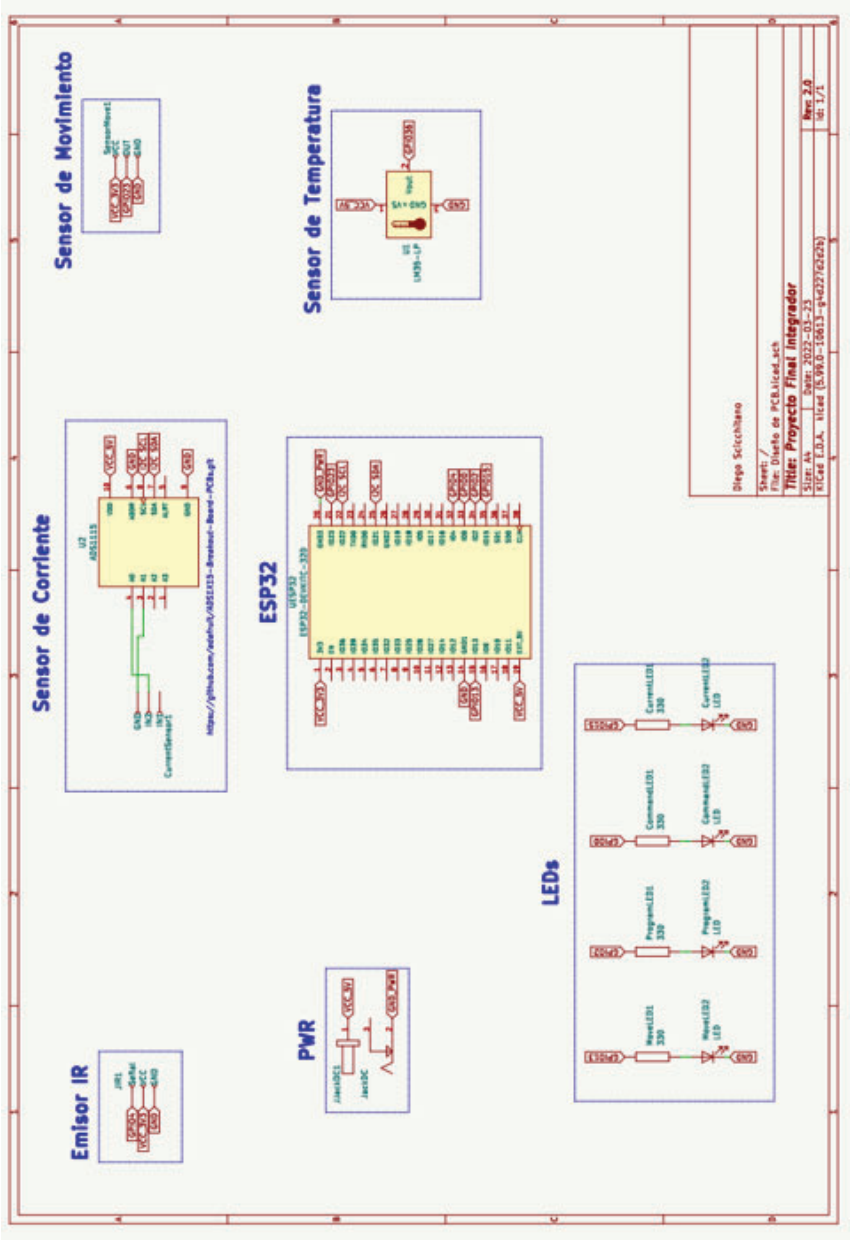



FIGURA A.1. Esquemático del PCB diseñado en KiCAD.

# Bibliografía

[1] https://espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf/.
[2] https://docs.kicad.org/6.0/en/introduction/introduction.html.
[3] https://developer.android.com/studio/intro?hl=es-419.
[4] M. A. Arias. «Introducción a PHP. IT Campus Academy». En: 2013.
[5] Ginestà M. G. Mora Ó. P. Santillán L. A. C. «Bases de datos en MySQL». En: Universitat oberta de Catalunya, 2014.
[6] https://mqtt.org/.
[7] https://naylampmechatronics.com/blog/51_tutorial-sensor-de-corriente-ac-no-invasivo-sct-013.html.
[8] https://puntoflotante.net/MANUAL-DEL-USUARIO-SENSOR-DE-MOVIMIENTO-PIR-HC-SR501.pdf.
[9] https://www.ti.com/lit/ds/symlink/lm35.pdf?&ts=1589289309000.